QuTech Central Controller: A Quantum Control Architecture for a Surface-17 Logical Qubit

Miguel D. Serrão M. Moreira

Delft University of Technology

TU Delft Delft University of Technology

Challenge the future

QuTech Central Controller: A Quantum Control Architecture for a Surface-17 Logical Qubit

---

A thesis presented
by

Miguel Duarte Serrão Morato Moreira

to
The Faculty of Electrical Engineering,
Mathematics and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Engineering.

To be defended publicly on May 10, 2019 at 14h.

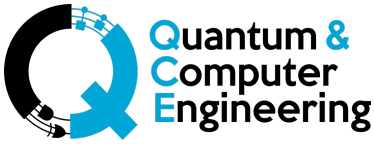
Delft University of Technology
The Netherlands

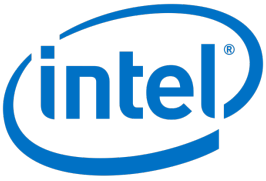
May 2019

Composition of the thesis committee:

Prof.dr. Koen L. M. Bertels	Quantum Computer Architectures Laboratory
Prof.dr.ir. Leonardo DiCarlo	QuTech Roadmap Leader
Dr. Carmina Almudever	Quantum Computer Architectures Laboratory

The work described in this thesis was carried out in the Quantum Computer Architectures Laboratory and the DiCarlo Laboratory, within QuTech. This work was supported by Intel Corporation and the Delft University of Technology (TU Delft).





Student Number: 4740661
Registration Number: Q&CE-QCA-MSc-2019-05
An electronic version of this thesis is available at http://repository.tudelft.nl/

Keywords: Quantum Computing, Computer Architectures, Systems Architecture
Cover: Photograph of the cryostat of a dilution refrigerator used to perform quantum computing experiments, by Jacob de Sterke (DiCarlo Laboratory, QuTech)

Dedicated to

my parents and Leonor

Abstract

The goal of this thesis is the design and development of the QuTech Central Controller, a system conceived to serve as the hardware/software interface of a quantum computer. This system represents an evolution of the QuMA microarchitecture to control a Surface-17 superconducting quantum processor, even though several architectural mechanisms are used to ensure the compatibility of the design with different quantum hardware technologies. In addition to an expansion of the control microarchitecture, the QuTech Central Controller represents an evolution of the overall system architecture, making use of a different hardware infrastructure to overcome previous scalability limitations.

The main contributions of this thesis are a proposed centralized microarchitecture capable of controlling up to 17 qubits, the implementation of this microarchitecture in a device called the QuTech Central Controller and its testing in dynamic quantum information processing experiments with superconducting qubits.

Acknowledgments

As I approach the finish line of my degree, I would like to thank all the people who have helped me both personally and scientifically along the way, people without whom this project would never have been possible.

I would like to express my very great appreciation to **Professor Koen Bertels** who welcomed me into his research group and created the best conditions for me to do this work. I greatly appreciate the freedom you have given me to find my own path and the guidance and support you always offered when I needed. I also want to take a moment to thank **Dr. Carmina Almudever** for her continued support during the writing of this thesis and for her encouragement and kindness throughout my work at the Quantum and Computer Engineering department.

I would like to offer my special thank you to **Professor Leonardo DiCarlo**, who has fostered me in his research group for the development of the majority of the practical work described in this thesis. I still remember my first impression of Leo during the Fundamentals of Quantum Information class. It was his unwavering enthusiasm and intuitive explanations of the principles of quantum information that finally drew me to this field. Thank you for supporting my work and for believing in me!

I also want to take a moment to thank my colleagues at the Quantum and Computer Engineering Laboratory and the DiCarlo Laboratory.

I am particularly grateful for the assistance given by **Xiang**. It was his seminal work on QuMA and eQASM that introduced me to the field of quantum computer architecture and his outstanding contribution to the group, CC-Light, formed the basis for the work presented in this thesis. I wish to thank him for all the thought-provoking discussions we had on control architectures and the future of this field, and for his support when I was still getting my bearings in the department. I wish you the best of success in your research undertakings in China and I hope we can work together again in the near future.

Many thanks to **Vieri** for his initial work on expanding the QuMA core and

his help during this project; but most of all for being the friend I could always rely on during these years in Delft. You have accompanied me for the great duration of this degree, always putting up with my pessimism during so many of the projects we worked on together. I still remember those late evenings spent in panic programming clusters of servers, and (slightly more fondly) all the barbecues you held. I believe it was very important to have someone who always saw the glass half full. I wish you the best of luck for the completion of you Double Masters. You will certainly give a better physicist than me!

I thank **Leon**, who introduced me to the Quantum and Computer Architectures Laboratory when I first arrived to Delft, for his friendship and support. It was you who motivated me to pursue this thesis when everything seemed uncertain! Thank you for your always cheerful presence in the office and for sharing some of the best memories I have of the Quantum and Computer Engineering department. Best of luck for your PhD in Duke!

I also wish to acknowledge the help provided by **Imran**, who I have fondly known since his days as an assistance to the Advanced Computing Systems course, **Hans**, for all the discussions on Compilers and Programming Languages, **Nader**, for his help with my first project for the department, **Abid**, for his friendship and the time we could share outside the stressful environment of the lab, **Lingling**, for her kindness and all the interesting conversations we had on Quantum Error Correction, **Savvas**, for his companionship and all his great suggestions on Greek cuisine, **Diogo**, my fellow countryman and friend from the harsh times spent studying for the Quantum Hardware course, **Daniel**, for his camaraderie and all the great late-night conversations fueled by beer, **Amitabh**, my peer in the work into microarchitectures for quantum control, and **Aritra**, who works hard to ensure that algorithms for genome sequencing may one day make use of the infrastructure being developed in the lab for the good of humanity. It was truly a pleasure to work alongside you all. I wish you good luck for your future endeavours and hope we can meet again in the future.

Recently, I have had the pleasure of also collaborating with **Adriaan**, whom I must thank for his help in testing the QCC in experiments with superconducting qubits and for most of the images presented in the chapter dedicated to these tests. I feel I have learnt a great deal from you on the physics behind quantum computing and the software tools used to program these devices. I hope we can continue to work together in the future!

I also wish to thank **Filip** and **Florian** for their early help in testing the QCC in quantum control experiments, for putting up with my incessant questions and

for contributing to the great environment in the DiCarlo Lab. I wish you lots of luck at Intel, Florian!

I thank **Niels** and **Jules** for their friendship and help during my work at the DiCarlo Laboratory. I wish you the best success with QBlox! I guess we still have to decide on how many crates this last sentence will earn me... hopefully we get to continue to work together in the future!

I extend a big thank you to **Luc**, **Ramiro**, **Brian**, **Thijs**, **Slava**, **Xavier**, **Thomas**, **Nandini** and **Alessandro** for their kindness in welcoming me to the DiCarlo Lab and for the great environment that I always experienced there.

To the technical team of the DiCarlo Lab, I owe the most sincere gratitude and recognition.

I would like to offer my special thank you to **Jacob** for his unwavering support, his motivation during stressful times and his kindness. I feel I have learnt a great deal from you on electrical design and, especially, on how to be a better engineer, through thorough examination of problems and pursuit of sound solutions. Your knowledge, passion and kindness are invaluable to this group!

I also want to thank **Jordy** for putting up with so many of my questions and doubts. Thank you for your mentoring on how to practically program and debug FPGAs. I have learnt a great deal from you! Thank you also for checking on my progress when you didn't have to and for your expert advice and help in addressing all the challenges the QCC faced.

Many thanks to **Wouter** for his work on the development of the CC hardware, which the QCC project relied on so deeply. This work would not have been possible without you. Thank you also for all the thought-provoking conversations and for everything you have taught me about system design. I hope I can contribute to the CC project at some point and honour all the help you have extended to me!

This brings me to **Gerco**; thank you for your help in all things software-related and for the interesting conversations on software development. Thank you also to **Martin Woudstra** for his help during the development of the drivers of the QCC, and to **Raymond V.** and **Raymond S.**, for their assistance and expert advice on the electronics that power QCC.

Outside the world of TU Delft, I would like to express my gratitude to **Professor Luis Miguel Silveira**. It was your support while I was still looking for where to pursue a MSc degree that made me choose coming to Delft, and how happy am I now with that choice! Thank you for your kindness, your constant availability and your thoughtful advice!

Thank you also to my uncles, Antonio and Luisa for their support, care and constant excitement, and to Primona, the best cousin a young kid could ask for; you were always like a second mother to me!

I extend a loving thank you to Leonor. Thank you for listening to my worries and for always putting challenges in perspective. Thank you for always lightning the mood with your laugh and for dreaming with me! Even though from a distance for long, you were always there.

Last but not least, I extend my deepest gratitude to my Mother. Thank you for always supporting me in pursuing my dreams, thank you for caring for me and for always being there when I needed you the most. I would not be the man I am today were it not for you.

Table of Contents

List of Figures

List of Acronyms and Symbols

AWG	Arbitrary Waveform Generator
DDR	Double Data Rate
FPGA	Field Programmable Gate Array
GPU	Graphics Processing Unit
ILP	Instruction Level Parallelism
I/O	Input and/or Output
ISA	Instruction Set Architecture
LSB	Least Significant Bit
MSB	Most Significant Bit
NISQ	Noisy Intermediate-Scale Quantum
NN	Nearest-Neighbour
QASM	Quantum Assembly
QEC	Quantum Error Correction
QECC	Quantum Error Correction Code
QED	Quantum Error Detection
QISA	Quantum Instruction Set Architecture
QuMA	Quantum MicroArchitecture
QCC	QuTech Central Controller
SERDES	Serializer/Deserializer
SIMD	Single Instruction Multiple Data
SoC	System-on-a-chip
UHFQC	Ultra-High Frequency Quantum Controller
VLIW	Very Long Instruction Word
VSM	Vector Switch Matrix

1 Introduction

To introduce the work developed during this thesis, this chapter starts by presenting an examination of the developments in the field of computer engineering over the past 50 years, as a way to motivate the need for developments in quantum computing and the implications that it could have in the way computation is performed in the future. Then, the objective of the work performed during this thesis is defined and an outline for this report is given.

1.1 Motivation

Over the past decades an incredible revolution in information technologies has been observed, a revolution driven mainly by ever increasing computing power. This increase in computing power has, itself, been motivated by advances in semiconductor manufacturing capabilities, which have allowed us to place more transistors (the most basic of computing elements) in a single chip and drive them at ever faster speeds. In the industry, the observational law governing the scaling of the number of transistors per processing chip was dubbed Moore's Law, after Gordon Moore. On the other hand, the law that explained the increase of transistor frequencies (the velocity at which we can drive these computing elements) was called Dennard Scaling, and stated that as transistors got smaller, their power density would stay constant, therefore allowing us to scale their speed, while maintaining their temperature within operating bounds.

However, several factors related to materials science lead to the breakdown of Dennard scaling, and the consequent need to fix the frequency of operation of conventional processors. This event lead to the radical end of a period of history where most performance in classical computers was obtained through Single Thread performance increase, where a single execution unit was respon-

sible for sequentially processing all information. Therefore, in the search for new ways to extract more performance from the growing number of transistors that could be placed on computing chips (since Moore's Law was still in effect), computer architects turned to Multi-Core architectures, where a single processor was made up of several processing units, each capable of executing their own stream of instructions and processing a subset of all the information required. This meant that, while these chips took up more resources (an abundant commodity, at the time) and their programming was more complex, better overall performance could still be achievable by dividing the problem in smaller pieces and computing them separately, in a manner analogous to divide-and-conquer strategies.

Be that as it may, this period was short lived as diminishing returns from parallelization were quickly observed, a prediction made in Amdahl's Law. This law stated that even if perfect parallelization was achieved, i.e. at no extra overhead, additional processors would only benefit the parallel section of a program and, therefore, performance increase would be limited. Diminishing returns from parallelization, predicted by this law, marked the end of an era where general purpose multi-core processors were the main way of driving performance forward in processing systems.

Instead, computer architects started to focus more attention on specializing different processors in their systems to excel at different tasks, combining their strengths to increase the overall performance of the system. This period could be called the Heterogenous Computing era and brought us accelerators like graphics processing units (GPUs), which are most often the reason behind great increases in performance in computing systems today. Notwithstanding, a new trend, called the Power Wall, has fundamentally threatened this strategy for the increase of computer's performance.

The Heterogeneous and Multi-Core eras were mainly driven by our ability to manufacture and operate ever increasing amounts of transistors per chip. However, it has been observed that the exponential increase in power consumed by these chips, has made it impossible to operate all the transistors in a single processor simultaneously, therefore restricting our ability to obtain more performance by simply increasing the number of execution units on a single processor. This has been the main factor in the end of the heterogeneous computing era.

All of these trends in processor architecture are visible in Figure 1.1, which represents a collection, over several years, of performance indicators of different microprocessors from different manufacturers. The logarithmic scale on

the vertical axis perfectly shows how the saturation of single core frequency (with the end of Dennard scaling) led to a significant decrease in single-thread performance in the following years.

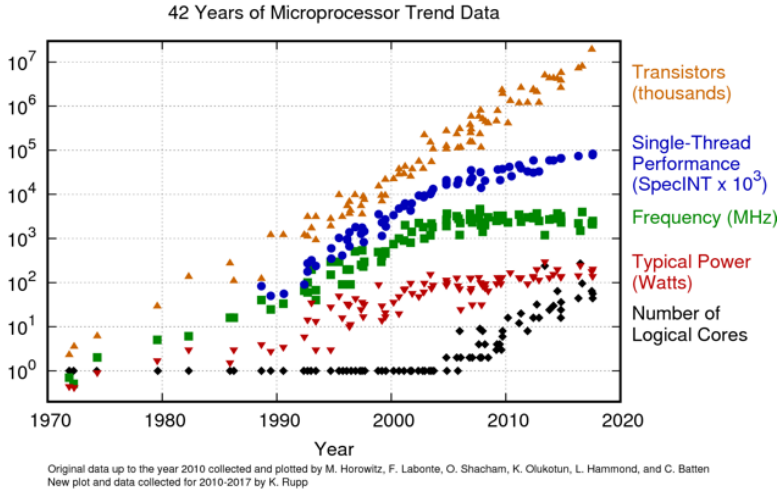


Figure 1.1: Trends in microprocessor specifications (collected and plotted by M. Horowitz, O. Shacham, K.Olukotun, L. Hammond and C. Batten, 2015)

Furthermore, the graph shows a clear saturation in the number of processors, between 2005 and 2010 (Amdahl's law), after which the increase has been mainly due to heterogeneous architectures, with GPUs holding massive numbers of execution units. Moreover, it highlights the tendency in the saturation of power per microprocessor, marking the maximum amount of power that is possible to deliver to a chip, otherwise known as the Power Wall. Finally, the chart clearly shows how, even after all of these years, Moore's law still holds true.

Most recently, a new era has emerged with application specific integrated circuits (ASICs) processors making up most of our ability to meaningfully increase performance in computing systems. First signs of the rise in popularity of these ASIC chips, used to accelerate only particular workloads, was seen in 2015. A good example of which is the Google TPU, used to accelerate neural network training and inference, while delivering superior energy efficiency results.

However, even ASIC chips, which some would argue are too expensive and narrow focused for the great majority of the computing community to rip benefits from, don't hold a good promise for the future of computing. This is

because Moore's law will, eventually, come to an end, as the size of the atom will stand as the absolute physical threshold to the reduction in feature size of transistors. Such a limit will lead to the subsequent stagnation of the number of transistors manufactured per chip and, therefore, the end of progress in our classical way of computing.

In the previous paragraphs, I have presented several arguments highlighting the shortcoming of the classical model of computation, also known as the Von Neumann architecture, or a machine whose data is moved to a processing system (from a storage medium) and back, in order to perform computation. It should be noted that this principle, in itself, already holds back classical machines in terms of energy efficiency, as nowadays the great majority of energy consumed by a processor is used to transport data back and forth from memory. Furthermore, it can only lead us to think that, to keep up with the demand for ever more powerful information processing systems, which have stimulated so many other fields of human knowledge, a radical revolution from the way in which computation is performed today will be required. One may even say it is essential for the continued development of humankind, through science and technology.

As will be discussed in the next chapters, Quantum Computers not only hold the potential to solve many of these challenges, but this new model of computing may also unlock NP-class problems, problems so hard to compute that they are beyond the grasp of classical machines.

1.2 Objective

The purpose of this thesis is to advance the state-of-the-art in control systems, enabling the control of larger numbers of qubits. It is my belief that developments in this area will be crucial for continued progress in the field of quantum computing.

In particular, the objective of this thesis is the design and development of a microarchitecture capable of controlling a Surface-17 quantum processor. Such a system will be based on the previously developed QuMA microarchitecture and expand its control capability to 17 qubits. Furthermore, the goal is to implement this microarchitecture, making use of a new hardware platform developed in-house, to create a device capable of running dynamic quantum information processing experiments with superconducting qubits. Such a device will be called the QuTech Central Controller (QCC).

1.3 Outline

This thesis is organized in six different chapters.

Chapter 2 (Background) provides a brief introduction to quantum information. Additionally, a description of the quantum model of computation is given, in order to motivate the need for the development of a central controller. The chapter will end with an overview into quantum technologies, and in particular superconducting qubits, so to give an understanding of how all system work to support quantum computation, from the digital level, to the electrical level and, finally, into quantum hardware.

Chapter 3 (Related work) presents several devices and architectures developed by different groups to address the challenge of quantum control, followed by an analysis into the merits and shortcomings of each of these approaches. These insights will be used to motivate new features in QCC and give a better understanding of the ongoing work on the field.

Chapter 4 (QuTech Central Controller) describes the work performed during this thesis to design, develop and implement the QuTech Central Controller. Initially, focus will be given to the change in hardware platforms performed during this project. Then, focus will be first shifted to the expansion of the QuMA core, responsible for the sequencing of quantum programs, and then to the changes in system architecture required to provide all the necessary I/O capability. The chapter will conclude with a description of the software systems developed to interface and control this hardware system.

Chapter 5 (Testing and verification) will provide a thorough description of all the tests performed at different stages of development of QCC to ensure the system met the desired design requirements and constraints. Furthermore, the chapter will present tests performed with the final system on the setting of experiments with superconducting quantum processors, in order to prove successful completion of the aforementioned objectives.

Chapter 6 (Conclusions and Outlook), as the name suggests, completes this report with an overview of the contributions made and draws on lessons learned while working on this thesis to give considerations on future work and provide an outlook into the broader field of quantum computer architectures.

2 Background

To fully appreciate the requirements and challenges presented by quantum computers, it is important to understand the basic principles of quantum information and quantum computation. It is the purpose of the following chapter to introduce these, starting with a description of the characteristics of quantum information and following that with a description of the quantum model of computation. Finally, a brief description of the physical systems that allow us to encode quantum information will be given, in an attempt to give a full understanding of how a quantum computer operates.

2.1 Quantum Information

Famously proposed by Richard Feynman in [1], quantum computers operate in a very different way from their classical counterparts. Their differences start immediately in the way that either machine encodes information. Computers available nowadays encode information using bits, which can be in either a 0 or a 1 state, as illustrated in Figure 2.1. However, in quantum computers information is encoded in quantum bits or qubits, which can be seen, in a way, as an extension of bits. In fact, they behave such that they can be 0, 1 or any combination (up to a normalizing factor) of 0 and 1, meaning that a quantum bit can be simultaneously in a 0 and a 1 state, a phenomenon called superposition. Indeed, a quantum state can be represented as in Equation 2.1, highlighting this phenomenon. However, upon measurement of a qubit, quantum mechanics dictates the collapse of this superposition state to the measurement result that is observed. Therefore, the measurement always has a definitive outcome. Indeed, upon measurement, the superposition of states is translated instead into the probability that a measurement outcome is observed.

To exemplify this lets return to Equation 2.1, where a qubit is partially (en-

coded by coefficient α_1) in the $|0\rangle$ state, and partially (encoded by coefficient β_1) in the $|1\rangle$ state. A measurement of this state would make the qubit collapse to state $|0\rangle$ with probability $|\alpha_1|^2$ or to state $|1\rangle$ with probability $|\beta_1|^2$, and would lead to a measurement result of '0' and '1', respectively.

$$|\psi\rangle = \alpha_1 |0\rangle + \beta_1 |1\rangle , \tag{2.1}$$

Furthermore, a visual representation of superposition can be had if one describes a quantum bit as a vector which can be in any position in a 3-dimensional space called a Bloch Sphere, as illustrated in Figure 2.1.

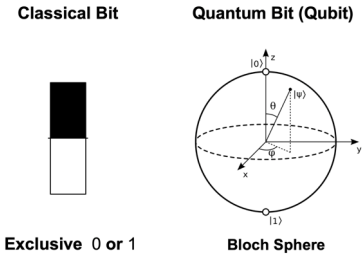


Figure 2.1: Fundamental units of information in classical and quantum computing

Even though strange at first, superposition is actually predicted by the laws of Quantum Mechanics and is, indeed, the way in which matter behaves at the nano-scale level. For the purposes of computation, however, this behavior is extremely interesting, as it allows the state space of our machine to grow exponentially with every added qubit. Think of it this way, if a qubit can be in a superposition, effectively being partially one and partially zero, as described by Equation 2.1, then two qubits in such a state would effectively represent, together, every possible combination of 0's and 1's, as described in Equation 2.2.

$$|\psi\rangle = \alpha_1\alpha_2 |00\rangle + \alpha_1\beta_2 |01\rangle + \beta_1\alpha_2 |10\rangle + \beta_1\beta_2 |11\rangle , \tag{2.2}$$

This then means that with every added qubit, the number of states that can be simultaneously represented grows exponentially as 2^n, whereas in a normal machine only one state can be described, as what instead grows with every added bit is the range of numbers that can be represented with that one state.

Now superposition is only one of the reasons why quantum computers are fundamentally more powerful than classical machines [2]. The second quantum phenomenon which gives quantum computers an edge over their classical counterparts is called entanglement. Even though harder to explain, this phenomenon means that qubits can be strongly correlated to one another (more so than classical bits) and, therefore, that operations performed on one, can strongly influence the other in ways that are not possible to replicate using classical bits.

It should be noted that, according to the quantum circuit model of computation, operations can then be performed on these qubits through the use of quantum gates. These can be thought of as somewhat of an analogue to classical logic gates in conventional digital circuits. They may act on one or more qubits at a time but, contrary to classical logic gates, the number of qubits in the input and output of a quantum gate must always be equal. A simple example of a single-qubit quantum gate is called the Pauli-X gate which maps the $|0\rangle$ state to the $|1\rangle$ state and the $|1\rangle$ state to the $|0\rangle$ state, in a way analogous with the NOT classical logic gate. The behavior of quantum gates can perhaps be better understood by observing Figure 2.2, where the effect of the Pauli-X gate on a qubit of state $|0\rangle$ (or $|1\rangle$) on the Bloch Sphere is represented.

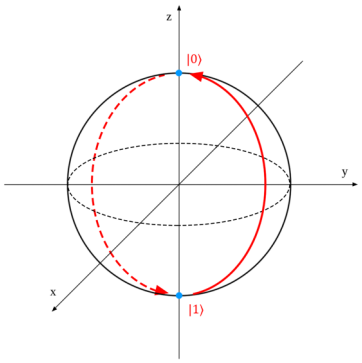


Figure 2.2: Effect of the Pauli-X gate on the qubit state

Through the use of quantum gates, in addition to the ability to initialize qubits and measure them, it is then possible to program a machine to perform quantum computation. In actuality, the DiVincenzo's criteria [3] set more rigorous conditions for this to be possible. However, this simple description highlights how a quantum computer can be programmed to make use of phenomena that are not available in classical computers, such as superposition and entanglement, to have an edge in processing information. Going back to my claim that

these machines may help us tackle NP-hard problems, it has been shown to be possible to factorize very large numbers (i.e. finding which prime numbers can be multiplied by one another to obtain a given number) very efficiently using quantum algorithms such as Shor's algorithm [4]. Such a problem is known to be very hard to compute classically, so much so that most of modern cryptography has been built on the premise that these problems can't be solved effectively. This is but one example of a problem domain only a quantum computer would allow us to tackle.

Other problems domains that we believe quantum computers could help us solve, due to other properties of these systems, include molecular simulation [5] and optimization [6]. Therefore, there is reason to believe that these machines could bring about true disruption to the field of computer engineering and support computing problems that could not be solved otherwise. Achieving such could truly be a turning point for the human species, allowing us to pursue ever greater scientific discoveries and substantially advance our technological capabilities.

2.2 Quantum Computation

An important realization into the way gate-based quantum computers are expected to work in the future, is that these machines will work as co-processors to general-purpose classical computers. There are several reasons why this would be the case.

Firstly, the no-cloning theorem, a physical principle stating that it is impossible to create an identical copy of an arbitrary unknown quantum state, limits the I/O capabilities of quantum computers. Therefore, these machines require careful initialization of their states. Furthermore, due to the destructive and probabilistic nature of measurements, quantum computers will either have to be run a statistically significant number of times per computation, to give the user an idea of the final state probability distribution, or its results will have to be checked by classical means.

Secondly, quantum algorithms are most often hybrid in their nature, requiring auxiliary classical computation to be performed before, during and/or after their execution. A good example of this is the aforementioned factorization algorithm. In fact, only the compute intensive part of the computation is performed on a quantum computer and used to find the period of a specific function. All remaining computation, required for integer factorization, is performed by a classical computer instead.

An analogue to this model of computation is found in Graphics Processing Units (GPUs) and Field Programmable Gate Arrays (FPGAs). In these devices an offload model of computation is used, where a general purpose processor prepares, starts and reads back the results of the computation carried out by these devices. Due to this intrinsic limitation in their way of operating, these devices are just used to accelerate computation heavy parts of programs, usually known as kernels. Due to constraints on the types of operations that quantum computers can perform, they are similarly expected to work as co-processors (or accelerators) of classical machines, therefore comprising a heterogeneous computer architecture.

Therefore, we should think of a quantum application as being composed of a host program, which should run on a central processing unit, and a quantum kernel, a program to be executed in a quantum computer to accelerate part of the total computation. An illustration of this model of computation in found in Figure 2.3.

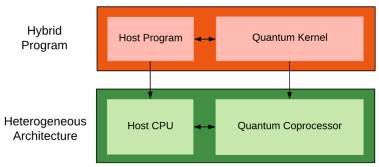


Figure 2.3: Heterogeneous quantum computation model

2.3 Quantum Technologies

In the previous sections, we have described some of the ways in which Quantum Information Theory will allow the acceleration of computations. However, to achieve a functional quantum computer, one needs to develop a physical implementation of qubits, so that we may operate them in a system. Furthermore, we need quantum mechanical effects, such as superposition and entanglement, to hold true for these systems to have a real quantum computer.

Different technologies have been developed to encode quantum information in a physical device. This can be done, for example, by encoding information in the polarization of photons [7]. Other examples of technological implementations of qubits exist, like ions trapped in electromagnetic fields [8], quantum dots [9] or nitrogen-vacancy centers in diamond [10], but in the next paragraphs we will focus on superconducting circuits [11], as a technology that

has matured more significantly than the rest in recent years.

Superconducting circuits allow us to implement qubits which are, on a fundamental level, simple non-linear RC oscillators, as represented in Figure 2.4. Just as we use voltage levels on electronic circuits to encode classical bits, the energy potential of these an-harmonic circuits is used to encode quantum information.

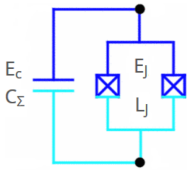


Figure 2.4: Circuit diagram of a superconducting qubit

Indeed, the energy potential diagram of a superconducting circuit, represented in Figure 2.5, shows how we can encode information on the discretized energy levels of these circuits, by encoding our 0 bit (state $|0\rangle$) on the lowest energy level of our system, and the 1 bit (state $|1\rangle$) on the level above it. Therefore, we can go from the $|0\rangle$ state to the $|1\rangle$ state by exciting our system, which is done by applying specifically tuned microwave pulses to it. The same can be done to drive transitions from the $|1\rangle$ state to the $|0\rangle$ state.

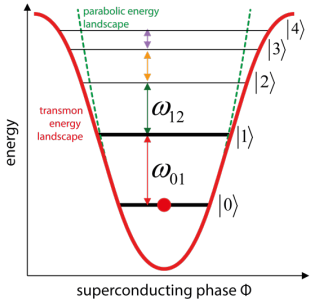


Figure 2.5: Energy potential diagram of a superconducting qubit (Christian Dickel, August 2017)

Just like logical gates on traditional computers rely on the switching behavior of bits, this brief example highlights how operations on quantum computers consist of specifically tuned microwave pulses, which can be used to implement single qubit gates, and flux pulses, which can be used to carry out two qubit gates. Additionally, these pulses can also be used to measure the state of qubits in somewhat of an analogue to traditional memory access but destruc-

tive, due to the properties of quantum mechanics. Therefore, a quantum chip is simply a superconducting circuit where operations are performed by applying electrical signals. A physical implementation of such a chip can be seen in Figure 2.6.

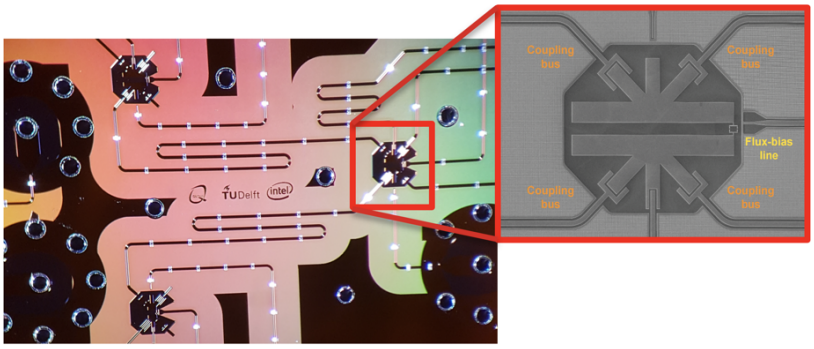


Figure 2.6: Superconducting Qubit chip

3
Related Work

The following chapter will present some of the work that has been carried out in the field of quantum control. It is the purpose of this chapter to highlight previous solutions to the challenge of programming and controlling quantum computing machines, solutions which will be built upon for accomplishing the objectives presented in this thesis.

3.1 Initial Approaches to Quantum Control

We have seen that electromagnetic pulses are used to control quantum bits. To understand the importance of accurately controlling which pulses are applied, two things should be noted. First, we should note that the technologies used today to implement quantum bits allow only very short lifetimes for these, in the order of a hundred microseconds, after which their quantum state decoheres. Secondly, that the correctness of the operations implemented by the microwave pulses is highly dependent on the timing at which they are applied. Therefore, it is extremely important to allow for the rapid and precise timing of quantum operations.

This is the reason why initial approaches to quantum control relied on the creation of long waveforms, made up of several smaller waveforms, each of which representing an operation, which would then be played all at once. We could then see these long waves as our quantum program, which would be played through Analog Signal Generators onto the qubits. Similarly, quantum measurement operations, which can be seen as somewhat of an analogue to classical memory accesses but that destroy the state of the measured qubits, relied on Digitizers to read back a waveform containing information of the state of a qubit.

However, there are several drawbacks to this approach. The first relates to the fact that programs are written as waveforms, limiting our ability to use high-level programming features (like parametric gates) to increase the abstraction available to developers in these platforms. Also, this extremely limits the scalability of our system, as adding qubits or executing longer programs would very quickly result on the overflow of the memory available on Arbitrary Waveform Generators (AWGs).

The second drawback relates to the fact that this control scheme requires the entire sequence of gates that composes our quantum program to be pre-programmed on the device and to be hard set, meaning that we could not change the order by which they will be applied. This is particularly limiting for quantum algorithms, which heavily rely on traditional control flow constructs, like conditional execution (i.e. conditional gates, gates that may or may not be applied depending on a measurement result) or repeat-until-success constructs, to implement the desired functionality.

In fact, most of the shortcoming of such an architecture are directly related to the four main challenges associated with a quantum control architecture, which we identify as being:

- scalability (in footprint and cost)
- flexible sequencing control
- signal synchronization
- real-time feedback

Therefore, a better scheme for quantum control was required, one that would allow the precise (and explicit) timing of quantum operations, while allowing for fast conditional execution and other control flow constructs. Furthermore, the scheme should be scalable to bigger quantum processors, and satisfy the high instruction issue rates constraints of quantum programs.

3.2 Previous Work

To satisfy the aforementioned requirements, new architectures started to be developed for quantum control based on a hierarchical system view. With these new approaches, researchers hopped to abstract complexity into different layers, much like in classical computing systems, and provide flexibility to the end programmer. The collection of these system, increasing in abstraction from bottom to top, is commonly called a stack, which creates a complete platform such that no additional software is needed to support applications. A

great example of such a control stack is the QuTech Stack, from which we will borrow for exemplification purposes in the following paragraphs.

In this new system architecture, an analog/digital interface, which was previously created using off-the-shelf Analog Signal Generators and Digitizers, first needed to be devised in such a way as to allow flexible digital control of all analog equipment. This was achieved by using Arbitrary Waveform Generators and Lock-in Amplifiers (with Analog to Digital Converters), both digitally controlled through codewords - i.e. sequences of binary bits. These devices are used to implement single qubit gates, two qubit gates and measurement operations, by generating the corresponding waveforms and applying them to the qubit chip.

Additionally, there was the need to develop a device responsible for controlling the aforementioned equipment with precise timing, through triggering of digital control codewords. Such a device was called a Central Controller (CC) and it establishes the hardware/software layer necessary to provide the required abstraction and flexibility to an end programmer. The architecture of such a system is represented by the diagram in Figure 3.1.

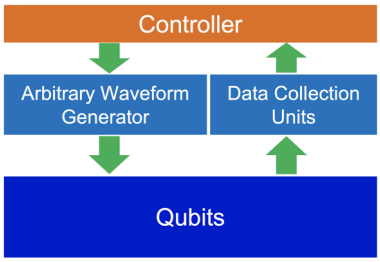


Figure 3.1: System architecture of the control hardware used at QuTech

The programming paradigm of our quantum control system also suffered a change, with the Central Controller acting as the interface between our program (software) and the underlying electrical systems (hardware) responsible for implementing quantum operations. The description in terms of functionality and organization of the hardware systems responsible for providing this interface is usually called a computer architecture, and an implementation of a particular computer architecture is usually called a microarchitecture.

In this sense, the CC presents an Instruction Set, where all the supported quantum operations are defined. Therefore, to program this machine, we should simply state the sequence of operations we want to run in a written description, commonly called QASM, or Quantum Assembly [12]. Our microarchitecture

will then be charged with triggering the AWGs and Digitizers, at the precise time, to implement these operations.

This sort of control scheme allows classical control flow structures to be implemented, since our microarchitecture deals with digital representations of the quantum operations up until the moment they are to be triggered, allowing repetition cycles to be dealt with very efficiently. Furthermore, having a centralized design allows the CC to very quickly decide on whether or not to trigger a gate, based on a measurement result it got back from the ADCs, allowing classical control flow structures, like conditional execution, to be implemented.

However, the level at which we can describe quantum operations to the CC is still very low. We can only describe the most basic of quantum gates and control constructs, almost directly implementable in a quantum chip using analog waveforms. To allow for the high-level description of quantum algorithms, we need to borrow from computer science and create the same abstractions that modern programming languages provide, to describe complex functionality from very basic operations.

To this end, and on top of the stack, reside high-level programming languages, in the QuTech Stack called OpenQL. Embedded in C++ or Python, well-known classical programming languages, OpenQL enables the description of quantum programs by making calls to other complex functions. This allows the creation of feature-rich libraries of quantum programs, in order to increase the efficiency of software development for these machines. Furthermore, it allows more effective and efficient manipulation of classical information and of the results of quantum computation.

After having described our quantum program in a high-level language, there is still the need to translate these high-level operations to the sets of instructions understood by our Central Controller, if we wish to run our program in a quantum machine. To achieve this, we make use of compilers, programs that known how high-level functionality can be built from low-level features, to automatically write code that is understandable by the microarchitecture. We can think of an analogue in classical computing, where a compiler can implement exponentiation through consecutive multiplications of a number by itself. In effect, the operations and optimizations a compiler is capable of making are much more complex. Nevertheless, this example illustrates how a compiler, having information related to the program we want to build and the infrastructure used to implement it, can intelligently translate and optimize our code. Furthermore, the translation of the high-level description of a quantum pro-

gram to a standardized, technology independent, low-level representation of it, like cQASM [12], would allow multiple compilation infrastructures, like ScaffCC [13], ProjectQ [14] or LIQUi|⟩ [15], to target the same quantum hardware or, even, allow the simulation of the program using a backend such as the QuTech QX Simulator [16], as represented in Figure 3.2.

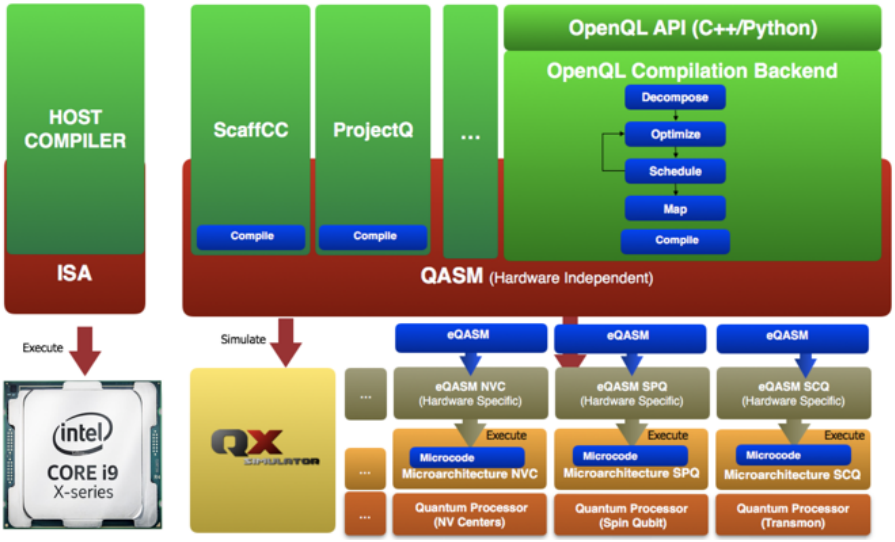


Figure 3.2: Quantum Compilation Infrastructure [16]

However, due to the characteristics of the aforementioned computing model for quantum computers, the compilation model for such a machine is slightly more complex. Since quantum programs are very often hybrid in nature, they may be composed of both quantum kernels and host programs, in addition to the auxiliary classical instructions required by the quantum kernel, such as conditional control flow structures. Therefore, a hybrid compilation framework is required, such that a host program written in a classical programming language like C++ can be compiled using classical compilation systems, while a quantum kernel, written in a quantum programming language like OpenQL, would be compiled by a quantum compilation system.

Such a compilation model is represented in Figure 3.3, with a hybrid compilation infrastructure generating a binary representation of a host program, to be run on a CPU, and a quantum compiler generating quantum code, made up of both quantum instruction (operations to be applied to the quantum chip) and classic auxiliary instructions (operations to be run on the central controller to support quantum computation).

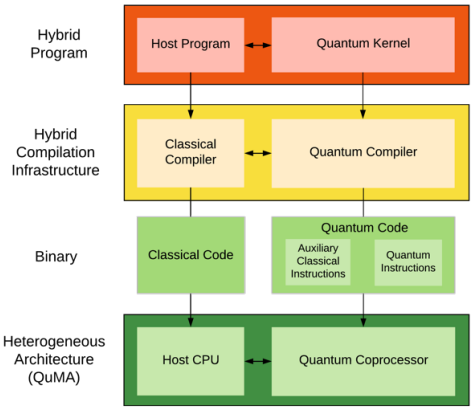


Figure 3.3: Quantum programming and compilation model [17]

All of the aforementioned technologies are combined in the full-stack to create a platform such that no additional software is needed to support the implementation of quantum applications, in an architecture capable of satisfying the initial requirements for scalability, flexible control, synchronization and real-time feedback. A visual representation of such a control stack is presented in Figure 3.4 and illustrates how algorithms are built on top of high-level programming languages, themselves making use of a compiler to be able to build on top of a Quantum Instruction Set Architecture, provided by our Central Controller, which then controls the hardware levels below to achieve the desired functionality. This represents the state-of-the-art in terms of quantum control systems, and the best way known to date to control a universal gate-based quantum computer.

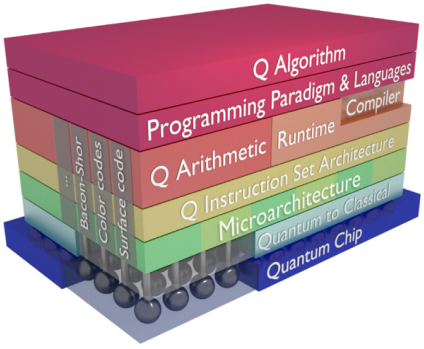


Figure 3.4: Full-stack quantum computing [17]

3.3 State-of-the-art

The focus of this thesis will be the microarchitecture that acts as an interface between quantum software and hardware. In this section, a description of CC-Light will be provided, given its role as the first implementation of the QuMA microarchitecture, a control microarchitecture developed at QuTech to target Surface-7 qubit chips within the framework of the aforementioned QuTech Stack.

The primary objective of this section will be to develop an understanding of the benefits and drawbacks associated with CC-Light's hardware, firmware and software implementations, in order to build from QuMA version-2 onto newer control microarchitectures.

Furthermore, other devices that have been developed to tackle the challenge of low-level quantum control, and that have recently been made commercially available, will be presented and evaluated. In particular, focus will be given to highlighting the advantages and shortcomings of different solutions to the challenge of programming and controlling quantum computers, solutions which may help address the objectives presented in this thesis.

3.3.1 CC-Light

The CC-Light is a controller developed at QuTech for dynamic quantum information processing experiments. This system can be seen in Figure 3.5 and implements a quantum microarchitecture known as QuMA, which represents a 32-bit instruction set instantiation of the eQASM executable quantum instruction set architecture [17] targeting a Surface-7 quantum processor.

Figure 3.5: Central Controller Light system (J.C. de Sterke, QuTech, 2018)

The full control system consists of a Central Controller, CC-Light, responsible for orchestrating slave devices, which comprise our analog-digital interface, for microwave control, flux control and measurement.

In particular, pulses are generated with Zurich Instruments High-Density Arbitrary Waveform Generators and are either directly applied to qubits, for flux operations, or modulated using a Rohde & Schwarz microwave source, for microwave operations. To implement the latter, an analog equipment called the Vector Switch Matrix is required for duplicating microwave pulses and routing these, while tailoring the waveforms to individual qubits using a frequency reuse scheme [18]. Furthermore, to carry-out measurements, a discrimination unit is implemented using two Zurich Instruments Ultra-High Frequency Quantum Controllers. These generate measurement pulses, sample received signals and determine measurement results. A frequency multiplexing scheme allows measurements of up to 9 qubits per unit.

All of these devices are controlled by the CC-Light using codeword triggers, sent through a 32-bit digital interface working at 50 MHz. That is besides the Vector Switch Matrix, which is controlled through digital signalling running at 400 MSa/s. The equipment comprising the analog/digital interface (ADI) of this system can be seen in yellow in the micro-architectural diagram shown in Figure 3.6.

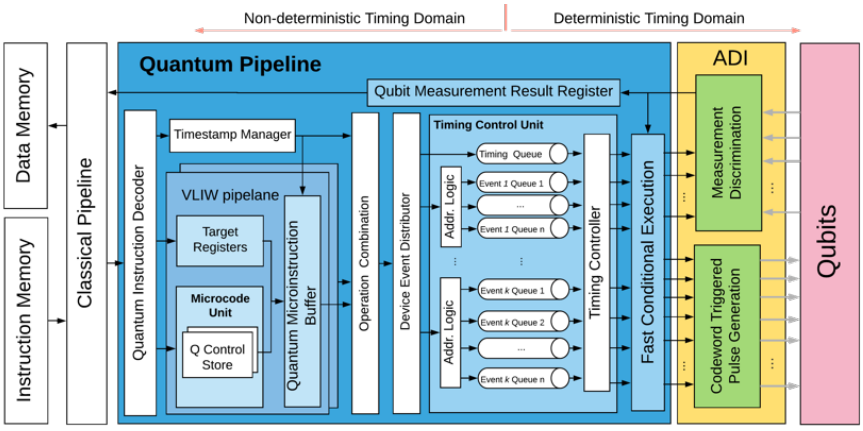


Figure 3.6: Quantum microarchitecture implementing an instantiation of eQASM for the control of a Surface-7 quantum processor [17]

The Central Controller, itself, can be seen to the left of the ADI in Figure 3.6, and is comprised of a Classical Pipeline and a Quantum Pipeline. The former is responsible for processing classical operations auxiliary to the execution

of quantum computation, such as branches and simple logical and arithmetic operations, and the latter is responsible for multi-level decoding and precise triggering of codeword-based events, being highlighted in blue on the aforementioned diagram.

Therefore, for the purposes of quantum control, the most important part of the microarchitecture is the Quantum Pipeline, the architecture of which will be described next.

Quantum Pipeline

Instructions destined for the quantum pipeline arrive through the classical pipeline, where the instruction memory is located. Upon reaching the quantum pipeline, the Quantum Instruction Decoder interprets the instruction into either a WAIT operation, a Register Setting operation or a quantum operation. Following is an explanation of the role of all functional units in the quantum pipeline, motivated by a description of how these three types of operations are processed.

A quantum operation represents an instruction which will be subsequently decoded into an operation to be performed on the quantum processor. Multi-level decoding in QuMA makes use of a microcode unit to translate every instruction into a micro-operation, allowing the underlying physical implementation of the quantum processor to be abstracted away from the programmer-visible ISA of our machine, therefore making QuMA quantum technology agnostic. Furthermore, QuMA allows the specification of quantum operations in a Very Long Instruction Word (VLIW) format, allowing the exploitation of instruction-level parallelism intrinsic to quantum programs. Therefore, quantum operations processed in VLIW lanes in parallel need to be combined, an operation performed by the Operation Combination unit. The last step of QuMA's multi-level decoding scheme consists of combining different micro-operations to be triggered simultaneously into device operations, the final representation of quantum operations in the quantum pipeline before they are sent as codeword's to the ADI. This is done by the Device Event Distributor.

WAIT instructions are the mechanism used in QuMA to efficiently and explicitly define the timing of quantum operations. In this way, a programmer is capable of explicitly defining when an operation is to be applied. This mechanism uses the Timestamp Manager to create a timing event every time a WAIT instruction is received, and stores it in a queue on the Timing Control Unit. Each timing event can them be associated with multiple device events on the

Timing Control Unit and, therefore, multiple events may be very efficiently triggered in parallel. Furthermore, this allows the Timing Control Unit to serve as the interface between the non-deterministic timing part of the pipeline and the time deterministic part, guaranteeing that our microarchitecture can work as fast as possible to ensure continuous processing of the quantum program while ensuring the triggering of quantum operations with very precise timing.

Register setting operations form the indirect qubit addressing mechanism used to increase the efficiency of qubit addressing in QuMA. Through this mechanism, an instruction first defines a set of qubit targets in the Target Registers, which may then be used by subsequent quantum operations as an operand. Therefore, a single-qubit operation may either be applied to a single qubit, by defining a single-qubit target, or to multiple qubits, by defining a multi-qubit target. The definition of both of these targets is done through the Set Mask Immediate for Single-qubit operations (SMIS) instruction. Finally, two-qubit operations can be performed by first defining a two-qubit target with the Set Mask Immediate for Two-qubit operations (SMIT) instruction, which again may hold a single qubit pair or multiple of these, allowing a way of operation analogous to Single Instruction Multiple Data (SIMD) in classical computers.

In the Quantum Pipeline, the Qubit Measurement Result Register is also worth noting, for its role in providing fast conditional execution. The fast conditional execution mechanism allows executing or canceling a single-qubit operation based on the execution flag register of the target qubit. This flag register is automatically derived by the microarchitecture from the last measurement result of the qubit using pre-defined combinatorial logic.

An overview of all quantum operations available in this instantiation of eQASM is presented in Table 3.1 but a reader interested in finding more details is encouraged to read [17].

Improvements to scalability

It should also be noted that the 2^{nd} version QuMA microarchitecture implemented in CC-Light, already makes a concerted effort to address many of the scaling problems encountered with previous centralized control microarchitectures. In particular, it addresses the instruction issue rate problem, which concerns the potential inability of the microarchitecture to issue and process all instructions required to control all qubits before such operations should be triggered.

This issue is alleviated by increasing the instruction information density

through the use of single operation multiple qubits execution, indirect qubit addressing mechanisms and a more efficient method for explicit timing specification (through the use of pre-intervals built into quantum instructions). Furthermore, a VLIW architecture is used to take advantage of the instruction-level parallelism, combining two parallel and different operations in a single instruction. However, I leave the specifics of each to [17], where these mechanisms are detailed.

Critical appraisal

CC-Light allows codeword-based instrument control and measurement automation through a unified interface accessible for high-level quantum programming languages through a compilation infrastructure. As the main advantages of this architecture, five should be highlighted:

- capability to explicitly specify timing of operations, achieved through the use of queue-based timing control scheme;
- power to implement program flow control, including (fast or comprehensive) run-time feedback, through either auxiliary classical instructions or automatically derived measurement flags;
- flexibility provided by the expressive definition of eQASM assembly, where quantum operations are defined at compile time instead of QISA design time
- agnosticism to quantum hardware implementation, through the use of multi-level decoding;
- scalability potential through the increase of the instruction information density, to alleviate the instruction issue rate problem.

However, CC-Light still has some shortcomings, particularly related to its implementation as a centralized architecture, i.e. a single entity through which all control and data paths flow, which may soon become a bottleneck for the processing and distribution of instructions in such a system. As the main disadvantages of this microarchitecture, the following should be noted:

- centralized design presents a grave scalability concern both by constraining the operation of all interface devices to the instruction issue rate capability of the QuMA pipeline and by the subsequent challenge in satisfying the input/output requirements of such a design in any individual platform;
- lack of full integration presents both a scalability challenge, in terms of cost and footprint, and also a control challenge, limiting the performance

Table 3.1: Overview of eQASM Instructions [17]

Type	Syntax	Description
Control	`CMP Rs, Rt`	Compare GPR `Rs` and `Rt` and store the result into the comparison flags.
	`BR <Comp. Flag>, Offset`	Jump to `PC + Offset` if the specified comparison flag is '`1`'.
Data Transfer	`FBR <Comp. Flag>, Rd`	Fetch the specified comparison flag into GPR Rd.
	`LDI Rd, Imm`	Rd = sign_ext(Imm[19..0], 32).
	`LDUI Rd, Imm, Rs`	Rd = Imm[14..0]::Rs[16..0].
	`LD Rd, Rt(Imm)`	Load data from memory address `Rt + Imm` into GPR `Rd`.
	`ST Rs, Rt(Imm)`	Store the value of GPR `Rs` in memory address `Rt + Imm`.
	`FMR Rd, Qi`	Fetch the result of the last measurement instruction on qubit `i` into GPR `Rd`.
Logical	`AND/OR/XOR Rd, Rs, Rt` `NOT Rd, Rt`	Logical and, or, exclusive or, not.
Arithmetic	`ADD/SUB Rd, Rs, Rt`	Addition and subtraction.
Waiting	`QWAIT Imm` `QWAITR Rs`	Specify a timing point by waiting for the number of cycles indicated by the immediate value `Imm` or the value of GPR `Rs`.
Target Specify	`SMIS Sd, <Qubit List>` `SMIT Td, <Qubit Pair List>`	Update the single- (two-)qubit operation target register `Sd` (`Td`).
Q. Bundle	`[PI,] Q_Op [| Q_Op]*`	Applying operations on qubits after waiting for a small number of cycles indicated by `PI`.

of the control system to that of its interface devices;
- rigid control scheme, due to the translation and sequencing of quantum gates at the firmware level, leaves QuMA inflexible to changes in chip size, layout or instrument configuration.

Addressing these will be both of particular importance for successfully scaling the QuMA microarchitecture to control greater numbers of qubits, a topic which will be covered in the next chapters, and for developing next-generation systems, which may bolster better scaling properties, provide full-integration and a more flexible definition of sequenced codewords for any quantum program.

3.3.2 Raytheon BBN APS2

Outside the scope of the QuTech stack, introduced in Section 3.2, other devices have been developed to tackle the challenge of low-level quantum control, and have been made commercially available. An example of such a system, the Arbitrary Pulse Sequencer 2 (APS2) was developed by Raytheon BBN Technologies for the control and readout of dynamic quantum information processing experiments on superconducting qubits [19].

A fully-populated APS2 system can be seen in Figure 3.7 and is composed of 9 APS2 modules and a trigger distribution module (TDM), which provide full control capability for a maximum of 8 qubits. Given its distributed architecture, to program the APS2 a quantum kernel needs to be translated into multiple separate binary executables which will run concurrently on each of the APS2's modules.

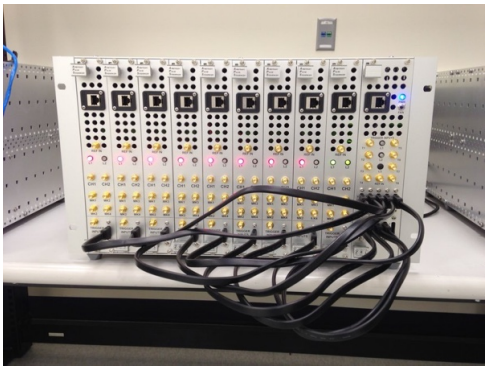


Figure 3.7: Raytheon BBN Technologies APS2 system [19]

In this scheme, precise timing is achieved via WAIT periods with the trigger distribution module generating an external signal that provides simultaneity and a method for synchronization of multiple modules. Additionally, a barrier-type instruction dubbed SYNC can be used to allow synchronization after non-deterministic waiting periods by stalling the processing of instructions until all execution queues are empty.

Critical appraisal

The main advantage of the APS2 is its distributed architecture, which leads to more natural scalability of the system. In particular, challenges like the instruction issue rate problem are more easily addressable on such a system, since

separate instruction streams are used to generate concurrent operation on individual modules. Furthermore, extending control capability to more qubits can more naturally be achieved by connecting APS2 systems together. Nonetheless, the system is currently only capable of fully controlling up to 8 qubits in a single enclosure. Therefore, and since no updates have been announced to add support for configurations of multiple APS2 systems together, the real advantage of implementing a distributed architecture is questionable at this point.

However, the distributed nature of the APS2 architecture does lead to some of the most significant shortcomings of this system. Furthermore, the loosely-coupled architecture of the system, in addition to its lack of native support for auxiliary classical instructions, hinders its ability to implement hybrid quantum-classical algorithms. The main disadvantages of the APS2 can be summarized as:

- Lack of support for comprehensive feedback (flexible and arbitrary decision logic) or for auxiliary classical instructions (real-time computation). This hinders the system's ability to implement hybrid quantum-classical algorithms, requiring host processor intervention for complex feedback, leading to likely catastrophic control latency;
- Complex compilation process requiring generation of multiple binaries from a single quantum application and conversion of quantum semantics into low-level operations (waveform output and program flow control). This is significantly more complex than what is required for CC-Light, which uses a single binary executable and explicit definition of timing and quantum semantics at the instruction level. However, it does allow more flexibility by decoupling the translation of quantum instructions into their respective waveforms (done in software) from the firmware implementing the pulse sequencer, therefore removing dependencies on chip layout and qubit count;
- Requirement for precise inter-module synchronization, due to distributed nature of the architecture, implemented through fragile interconnect network. This limits the immediate scalability of the system to multiple APS2 enclosures, required for controlling greater number of qubits;
- No support for output instructions during inter-module synchronization.

These shortcomings make the Raytheon APS2 system incapable of satisfying the control requirements of a heterogeneous, loosely-coupled computing system capable of supporting a hybrid quantum-classic quantum model of com-

putation.

3.3.3 Keysight HVI Engine

The Quantum Engineering Toolkit (QET) is a system developed by Keysight Technologies for the control of single and multi-qubit architectures. It consists of several hardware systems, which can be seen in Figure 3.8, and their respective control and calibration software packages. In particular, this toolkit is composed of:

- Chassis supporting a modular system composed of PXIe arbitrary wave generators (AWG), digitizers (DIG), embedded controllers and microwave-frequency local oscillators;
- I/Q Modulators/Demodulators systems for up-conversion of pulses to microwave frequency and down-conversion to DC, baseband or intermediate frequencies;
- Software environment for development of the necessary control sequences and for signal path manipulation;
- Labber software suite for high-level experimental control, visualization and result data management

Figure 3.8: Hardware systems composing Keysight Technologies' Quantum Engineering Toolkit [20]

For the purposes of this thesis, focus is given to Keysight's Hard Virtual Instrument (HVI) Engine, a sort of High Level Sequencer which provides the capability to create time-deterministic control sequences to be executed by the hardware modules in parallel. Therefore, and similarly to the CC-Light and APS2, the HVI Engine is responsible for the orchestration of a quantum com-

puting experiment, enabling synchronous control of the signal generation and acquisition modules, and feedback by implementing decision making between AWG and digitizer modules.

Critical appraisal

Notwithstanding the lack of details on its implementation and characteristics, such as latency, the claimed ability to synchronously control signal generation across several ports, modules or even multiple chassis, is one of the greatest advantages of this system, in addition to its modularity. Furthermore, classic auxiliary operations like arithmetic operations are supported, allowing (at least theoretical) hybrid quantum-classical algorithm support. Additionally, custom real-time FPGA-based processing can be added to the data path, between the acquisition and the transmission of data to the computer, allowing for more comprehensive feedback support.

However, similarly to the APS2 system, the programming of this device requires generation of multiple binaries from a single quantum application and conversion of quantum semantics into low-level operations. Additionally, the requirement on writing these as time-driven flowcharts exacerbates this issue by leading to slower programming and difficult code reuse.

Therefore, the shortcomings of this system, together with the lack of detailed information on its implementation and operation characteristic, make the Keysight QET system inapt at tackling the challenge presented by the need for scalable, high-level control of a quantum processor.

4

QuTech Central Controller

The following chapter will describe the work performed to implement the QuTech Central Controller (QCC), a device created to support dynamic quantum information processing experiments on superconducting qubits. This device will be based on an expanded QuMA core, targeting a Surface-17 quantum processor, and make use of a new hardware architecture developed at QuTech for next-generation quantum controllers. In particular, focus will be given to how the main limitations of a centralized architecture were overcome, and how these culminated in the new architecture for quantum control called the QuTech Central Controller.

The first section details how the QuMA architecture was ported to a new hardware platform, developed in-house to accommodate next-generation central controllers. Then, a description of how the 2^{nd} version QuMA microarchitecture was expanded to provide support for the control of 17 qubits will be given. Following this, focus will be shifted to how the interface scaling challenge was addressed, through the redesign of the system to a semi-distributed architecture. Finally, the software infrastructure developed to interface and control this microarchitecture will be presented. This chapter will conclude with a summary of all aforementioned modifications, as a way to review the work that was performed during this project.

4.1 Platform Change

The initial instantiation of eQASM to target a Surface-7 quantum processor was implemented in CC-Light through the use of an Intel Altera Cyclone V System-on-a-chip, incorporating a Field Programmable Gate Array and a ARM-based Cortex-A9 MPCore as a hard processor system, in addition to

other interface and peripheral devices.

However, for the next generation central controllers, there was a concerted effort to switch platforms to a Xilinx Zynq-7000 SoC, a device incorporating a Kyntex-7 FPGA and a similar dual-Core ARM-based Cortex-A9 MPCore processing system, shown in Figure 4.1 in its board package. The reasons for this change are beyond the scope of this report. To implement QuMA in this new platform several changes were required, which will be detailed next.

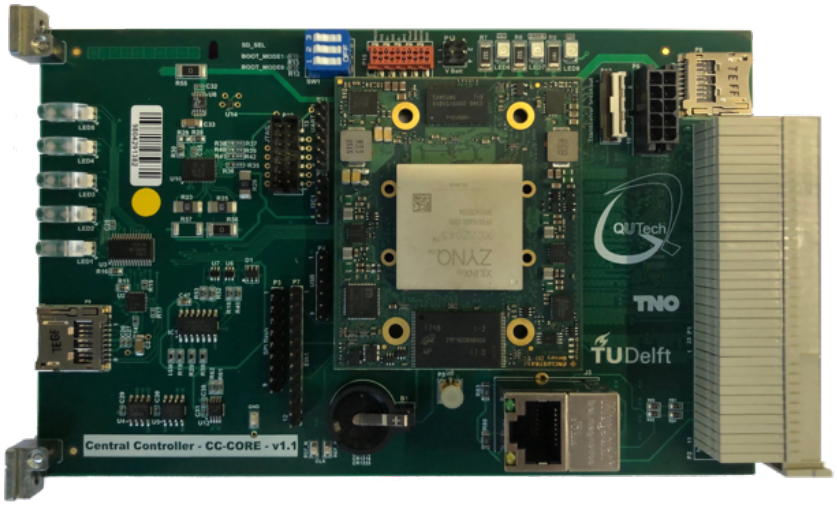


Figure 4.1: FPGA and processing system used at QuTech for the control of quantum experiments (Vlothuizen W.J., QuTech)

Firstly, there was a need to migrate all project files to Xilinx's design suite, to allow the use of its synthesis and integration toolchains. At this stage, there was also the need to regenerate the Hard Processing System previously available in the Altera platform and used to interface with the QuMA microarchitecture through AXI-based transactions. The generation of this Processing System in Xilinx's toolchain allowed maintaining this form of interfacing unchanged.

Then, there was the need to remove vendor specific pragmas and precompilation commands from all VHDL files describing QuMA's implementation. These pragmas, otherwise known as compiler directives, are additional commands given to the compiler to influence how VHDL code will be synthesized into an equivalent hardware representation. Since these define constructs with no predefined language semantics, new directives had to be specified for

Xilinx's compilation infrastructure, to ensure proper synthesis of the design.

Next, all vendor specific Intellectual Property (IP) had to be removed and regenerated for the new platform. In particular, all First In First Out (FIFO) data elements generated using the Quartus tooling and used to implement the timing control and measurement result analysis units, among others, had to be regenerated using Xilinx's FIFO LogiCORE v13.2 and simulated for correct operation. The reset behavior of these memory elements was of particular concern and their timings presented some issues in intermediate designs, problems which we will cover in Chapter 5.1.

Finally, first-stage and universal boot-loaders had to be created for the new platform. The first-stage boot-loader is responsible for loading the bitstream of our synthesized design into the reconfigurable fabric of the System-on-a-Chip (SoC). Furthermore, it is responsible for configuring the Processing System of the SoC at boot time. On the other hand, the universal boot loader is responsible for loading the device's operating system kernel, allowing the booting of a custom embedded Linux distribution from the SoC's Hard Processing system. This software system will allow the creation of the interface to the QuTech Central Controller, allowing its configuration, control and the transfer of information through an internet protocol running over an ethernet link.

4.2 QuMA Core Expansion

The necessity for the QuTech Central Controller came from the need to experimentally control a Surface-17 quantum chip, a schematic of which can be found in Figure 4.2. Such a quantum processor provides the necessary fabric of fast-flux-tunable transmon qubits interacting with nearest neighbors, as well as the necessary quantum and classic interconnect [18] to implement quantum error correction cycles required for the surface code quantum error correcting code [21]. It should be noted that the purple lines represented in Figure 4.2 are the feedlines and resonators required to perform measurements of the qubits, while the yellow (red) lines represent the I/O used to apply flux (microwave) operations to qubits and the orange lines represent the coupling buses used to mediate two-qubit interactions.

Such a surface code array could implement a distance-3 logical qubit, the basic unit of information in a fault-tolerant quantum computer, allowing great progress in research into error correction in quantum computers. Therefore, there was the need to expand the QuMA version-2 microarchitecture to allow the individual control of 17 qubits, to implement the control scheme necessary

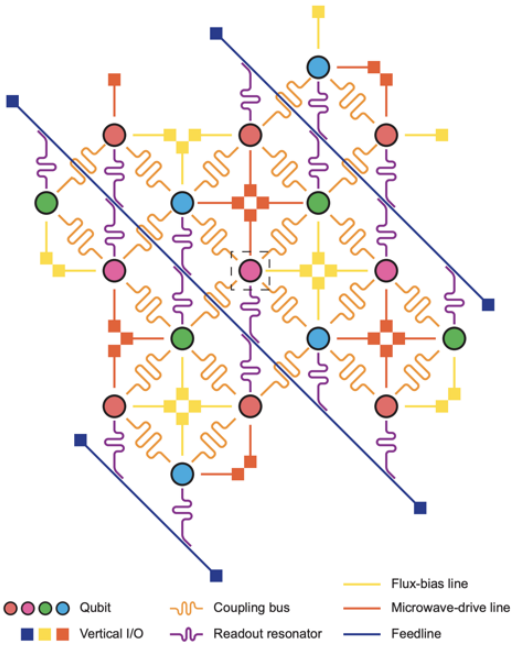


Figure 4.2: Schematic of a Surface-17 quantum processor, highlighting quantum and classical interconnect [18]

for such an application.

To expand QuMA v2 to satisfy the control requirements of QCC, several modifications to the quantum pipeline were necessary. An initial project into the expansion of QuMA v2 had already identified and implemented some of these. However, the need to migrate hardware platforms lead to the need to rebuild an expanded version of QuMA v2. The following sections will be dedicated to describing the changes that were performed and to explain how the scaling challenges associated with a centralized architecture were addressed in QCC.

Instruction Reformatting

The increase of the amount of individually addressable qubits has an immediate effect on the instruction word used in QuMA. However, this effect is somewhat reduced due to QuMA's use of an indirect qubit addressing mechanism. Indeed, quantum instructions only specific a target register (either for single or two qubit operations), which is addressed with 5 bits in this specific implementation. Therefore, by maintaining this addressing scheme in QCC,

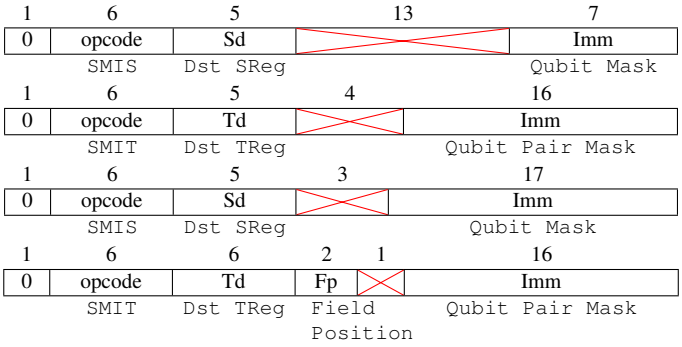


Figure 4.3: Format of the `SMIS` and `SMIT` instructions used in CC-Light (top two), and `SMIS` and `SMIT` instructions used in QCC (bottom two)

only the register setting instructions (SMIS and SMIT) will need to be modified to support addressing up to 17 qubits. The format of SMIS and SMIT instructions used in CC-Light can be observed in the top two images presented in Figure 4.3.

Given the 32-bit instruction word instantiation of eQASM in CC-Light, there are enough bits available in the instruction word to encode the additional 10 qubits addressable in QCC for single-qubit operations. Therefore, the qubit mask field of the SMIS instruction was simple enlarged to 17 bits for QCC, as represented in the 3^{rd} image presented in Figure 4.3.

However, for two-qubit operations, this is no longer the case. Indeed, the topology of a Surface-17 quantum chip allows 48 different two-qubit pairs, as illustrated in Figure 4.2. It should be noted that each coupling bus in Figure 4.2 represents a potential two-qubit interaction and that each edge representing such an interaction should be directed since a CNOT gate, for example, has a different effect on the control and the target qubits, leading to 48 different potential two-qubit interactions that need to be individually addressed in a Surface-17 chip.

Therefore, to introduce minimal change to the way the SMIT instructions operate in QCC, it was decided to establish a new scheme for the setting of two-qubit target registers, where the *Qubit Pair Mask* would be split between three different instructions, each identified with a unique *Field Position*, so that they could then be re-combined for proper storing in the *Mask Register File*. Furthermore, the number of two-qubit target registers was expanded from 32 to 64, to account for the additional number of potential two-qubit pairs. This was done in an attempt to reduce the potential overhead incurred by requiring a

change of the Qubit Target Registers during the execution of a quantum program. This scheme for the setting of two-qubit target registers in QCC lead to the changes in format of the SMIT instruction represented in the bottom image of Figure 4.3.

Instrumentation and Qubit Topology

In addition to the changes in addressing mechanisms introduced in QCC, the increase of the number of individually addressable qubits also had an effect in the requirements on the Analog/Digital interface, implemented through the use of Arbitrary Waveform Generators (AWG) and Ultra High-Frequency Quantum Controllers (UHFQC), as previously described. Indeed, an AWG or UHFQC device has a limited number of input/output channels and, therefore, a change in the number of addressable qubits requires the control of additional such devices.

For Surface-17, a microwave frequency reuse scheme is used, as described in [18] and, therefore, only two AWGs are required for microwave operations, in addition to a Vector Switch Matrix (VSM) device, necessary to implement the reuse scheme. Also, the need to individually control 17 flux-tunable qubits adds the need for an additional three AWGs, to apply flux operations. Finally, three UHFQC devices will be used for measurement of all 17 qubits through the three available feedlines.

These changes required modifications to the Device Event Distributor, the unit responsible for combining micro-operations into device events, used for triggering the aforementioned analog devices. In addition to expanding the Device Event Distributor to the extra devices used for controlling Surface-17, the decoding of target qubits to instruments assigned to implement operations on them had to be changed, to satisfy the new chip topology.

For microwave drives, qubits were assigned to AWGs according to the qubit frequency group where they belonged. Therefore, all qubits highlighted blue in Figure 4.4 were assigned to the same device, as were those highlighted in orange, green and yellow. Moreover, for flux operations, qubits were assigned to AWGs according to their qubit number, represented in red in Figure 4.4. Thus qubits 0 to 7 were assigned to the first AWG and similarly for all other qubits. Finally, for measurement operations, qubits were assigned to UHFQC devices according to the feedline assigned to them. Hence, qubits 0 and 3 were assigned to the first device and similarly for the rest.

However, changes in the qubit plane topology also influenced the way in which

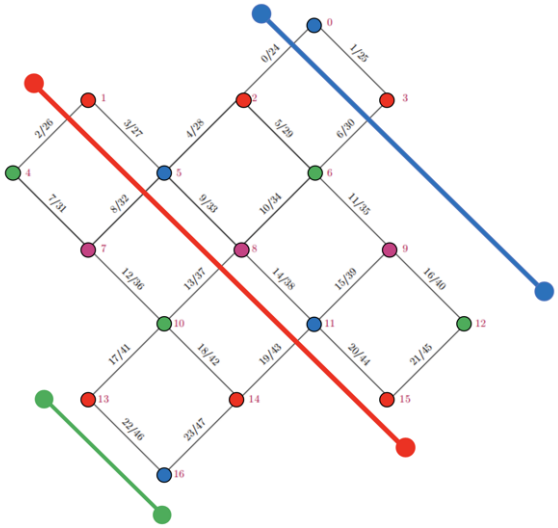


Figure 4.4: Feedline and microwave frequency scheme for a Surface-17 quantum processor

quantum instructions are decoded throughout the Quantum Pipeline. In particular, changes to the physical addresses of individual qubits, and to the addresses of two-qubit pairs, required a new decoder scheme for selecting micro-operations from the Microcode Unit. Therefore, the Microcode Address Decoder was changed to reflect the new physical addresses of qubits, specified by the numbers visible in the topology diagram represented in Figure 4.4. Furthermore, the *Measurement Result Analysis* unit, responsible for interfacing with the UHFQC devices to read-back measurement results, was modified to account for the new feedline topology, and so was the *Measurement Issue Generation* unit, responsible for dispatching measurement triggers. It should be noted that the way in which the UHFQC devices work allows them to take as input from the QCC a measurement operation and to output a discriminated measurement result after concluding the measurement operation.

Finally, the *Timing Control Unit*, responsible for maintaining the deterministic timing domain in our microarchitecture, was expanded to include device event queues for all additional instruments, concluding the necessary modifications to interface with the Surface-17 quantum processor.

Before concluding the expansion of the QuMA Core, the control and data paths of the microarchitecture add to be expanded to operate 17 qubits and the Classical Pipeline, where the *Measurement Register File* (used to store measurement

results) is located, similarly expanded.

The AXI driver, responsible for mediating communications between the Cortex-based Processing System and the registers and memory located within QuMA, was also expanded to allow setting new control signals for QuMA configuration.

Scaling Challenges

Having completed the expansion of the QuMA core to target a Surface-17 quantum processor, the proposed design was tested with a synthetic benchmark, to determine whether the instruction issue rate was sufficient to successfully drive all devices in parallel. Such a benchmark had to be made of consecutive parallel operations on all qubits, to ensure maximal pressure on the instruction fetch and decode units. The successful fulfillment of this experiment indicated that the mitigation schemes implemented in QuMA version 2 to reduce the instructions issue rate problem, namely VLIW architecture, indirect qubit addressing and efficient timing specification, were sufficient for an architecture controlling up to 17 qubits. A thorough description of all the experiments that were ran to test the expanded QuMA core are presented in Chapter 5.1.

However, not all challenges associated with the limitations of a centralized architecture were surpassed with the expansion of QuMA. Again, these challenges were determined to be:

- instruction issue rate problem
- input/output from FPGA
- scalability

Having addressed the scalability challenge by manually enlarging all the control and data paths in the microarchitecture to support the control of 17 qubtis individually, and the instruction issue rate problem through the use of 2-way VLIW, indirect qubit addressing and efficient timing specification, we have yet to address the input/output challenge. Indeed, even though QCC should now be capable to implement all of the desired functionality, the FPGA platform it is implemented in only has 48 additional pins available for I/O, whereas the interfaces used to communicate with all of the analog instrumentation require an aggregated 288 pins for communication. Addressing this challenge will require us to soften the limits of what we understand as a centralized architecture, and will be the focus of the next chapter.

4.3 System Redesign

Having completed the design for a QuMA core targeting the control of 17 qubit chips, there was the need to build the interface capability to connect the control signals generated in that system to the device responsible for generating the corresponding analog waveforms for qubit control.

Given the high-demand for parallel, low-speed signalling from the control unit to these devices, there are not enough extra pins in the FPGA package to be used to implement such interfaces. To solve this problem, the system architecture of QCC was redesigned to allow a semi-distributed control approach, where serialization of signals to an intermediate platform is used to increase the output throughput achievable from the limited number of I/O pins available in the FPGA package where the QuMA core is implemented.

System Architecture

A diagram of the full system architecture of QCC is represented in Figure 4.5 and was designed to combine a CORE platform, implementing the QuMA core, with up to 12 IO platforms. These IO platforms are made up of similar FPGA boards, combining a hard processing system with reconfigurable logic fabric, and are connected to the CORE platform through a custom designed backplane, allowing high-speed serial communication between the CORE and IO boards. Furthermore, the system incorporates a power supply unit (PSU), a clock distribution network and an Ethernet switch, connecting all boards together in a star configuration.

Such an architecture allows the scaling of the interface of the QuMA core through the serialization of all IO communication. By increasing the speed at which information is sent from the CORE to the IO boards, it is possible to achieve sufficient throughput with reduced requirements on the number of IO pins. This scheme calls for high-speed serial communication between the CORE and IO boards, and the subsequent de-serialization of these signals in the IO board, therefore allowing the combined system to drive all instruments through the initial parallel protocol.

In particular, the parallel protocol requires digital communication through 32 pins at 50 MHz. Since 48 pins are available in the CORE FPGA package, it was decided to dedicate 4 pins for communication with each of the 12 IO boards, of which 2 pins will be required for implementing a serial synchronization protocol, as will be detailed in the section dedicated to the I/O Board Design.

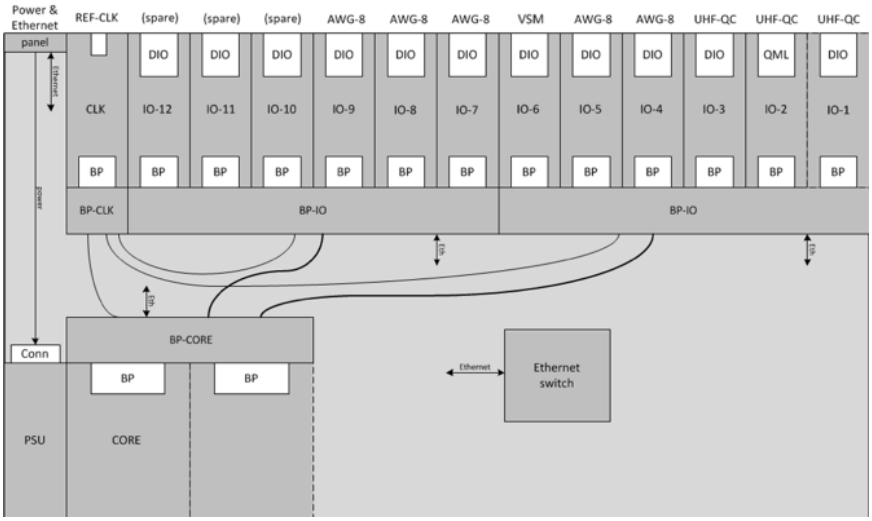


Figure 4.5: Next-generation quantum controller hardware diagram (Vlothuizen W.J., 2017, adapted)

Therefore, serialization of data to 800 MHz was required to achieve similar throughput using 2 pins as would be achieved through a parallel interface of 32 pins running at 50 MHz.

Feedback Latency Analysis

A very important part of a quantum program resides in the ability to perform feedback based on measurement results. What's more, most know quantum algorithms (and other control schemes, like active qubit reset) require the use of classically controlled gates, which make use of this feedback mechanism.

The requirements on this form of feedback are especially stringent for a system which aims to achieve quantum error correction through the use of stabilizer-based quantum error correction codes (like is the case with Surface-17). This is because, in order to implement such error correction schemes, one needs to constantly perform measurements of certain qubits (known as ancilla qubits) and apply operations on others (known as data qubits), depending on the measurement results obtained from the first. The duration of the cycles consisting of these operations is vital for the success of the correction scheme. Therefore, the design of QCC was done in such a way as to achieve the smallest possible feedback latency loop. To understand what went into this decision, it is first important to understand the mechanism for feedback used in the QuMA core.

Due to the latency associated with operating digital equipment, such as the ZI UHFQC and AWG, it was decided that fast-feedback, the mechanism in QuMA that allows feedback with latencies of the order of a hundred nanoseconds (for conditional execution of gates), should instead be implemented through the VSM, a completely analog device used to duplicate, route and tailor waveforms to individual qubits. It should be noted that the VSM is driven by purely analog signals and, therefore, is not contingent on the 50 MHz protocol implemented for communication with all other analog-interface devices.

Therefore, the feedback scheme should start with a measurement operation being sent to the UHFQC, quickly followed by sending (to the AWG) the microwave operation we want to condition on the measurement result, even when no information is available in the QCC to determine whether or not this operation is to be performed. Instead, when the measurement result is returned to the QCC, a mask operation is quickly sent to the VSM to either route the signal sent by the AWG to the qubit (in case the measurement result supports performing this operation) or to block it. Such a scheme allows avoiding the latency associated with executing conditional execution on the AWG, reducing the feedback loop cycle to its minimum by using the fastest equipment available (a completely analog-controlled set of switches in the VSM) to perform Go/No-go decisions on measurement results. A visual representation of this feedback scheme is presented in Figure 4.6.

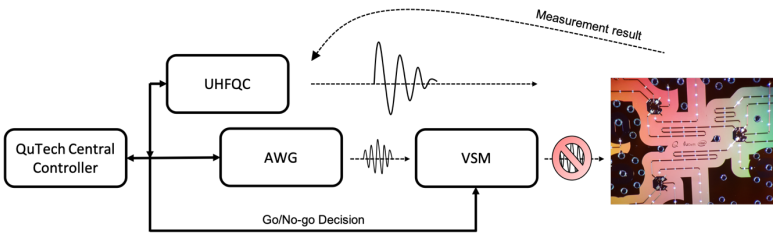


Figure 4.6: Illustration of feedback mechanism implemented in QuMA

It should be noted that all qubit operations have a predefined duration and that, therefore, by tuning the delay added to the lines used to operate each device, it is possible to calibrate all the equipment so that a go/no-go decision on the VSM coincides with the receiving of the corresponding microwave coming from the AWG. Additionally, it should also be noted that feedback latency for QuMA is defined as the time between a measurement result being received and a signal being sent to the VSM for triggering the correct routing decision.

I/O Board Design

Having a monolithic architecture, CC-Light is particularly suited for this scheme because the delay, required to calibrate the devices, can be added to the signalling cables (in the case of the VSM) before the final signal is sent to the device, therefore preventing the calibration delay from counting towards the feedback latency loop. This is because the delay is added before a decision based on the measurement result has to be made, therefore attenuating the impact of delay added for calibration of device communications in the feedback latency of the system. Another way to think of this is to consider that delay is being added while we are waiting for the measurement result to be received from the UHFQC for a conditional decision to be possible. In CC-Light, since all signals (including the 400 MHz signal used to communicate with the VSM) are generated and received in the same FPGA package, such a scheme is easily implemented.

However, the semi-distributed nature of QCC, and the fact that communication between the boards is done at 50 MHz (even for the IO board controlling the VSM device) means that the VSM control signal has to be up-converted from 50 MHz to 400 MHz on the respective IO board, to implement the 2.5 ns delay increments that were found to be necessary to properly configure communication with this device. Therefore, all the delay that can be added before conditioning the execution of a signal on a measurement result, something that can only be done on the CORE board (since this is the one where measurement results are available) is minimally limited to 20 ns, due to the interface of the CORE board being limited to 50 MHz. Thus smaller delay increments (of 2.5 ns) have to be applied to the VSM signal in the respective IO board, after conditional execution has been performed.

This means that, in the worst case scenario where 17.5 ns of delay are required to configure communication with the VSM, the feedback latency of the system could be worsen by that amount. With these considerations in mind, the IO board was designed to include the following three main components.

First, a set of configurable delay modules capable of implementing 2.5 ns delay increments up to 17.5 ns of total delay. It should be noted that only a total of 17.5 ns should ever be implemented in the IO side, since 20 ns delay increments can be added in the CORE side, resulting in a reduced feedback latency loop, while still satisfying communication requirements.

Secondly, a de-serialization module to allow the IO board to receive (or send) data from (or to) the CORE at 800 MHz through the 4 available serial lanes.

The scheme for synchronization of the serializers has evolved over time to address some of the problems encountered with data loss and corruption, which will be covered in Chapter 5.1. However, as a very basic insight into the operation of SERDES, these devices make use of a string of pulses, output from a serializer and received by the de-serializer, to determine if there is alignment between lanes and to correct for the miss-reception of signals. This is the reason why, from the 4 serial lanes running at 800 MHz, we only obtain an effective throughput similar to that achieved by 2 such lanes, as the rest are used for implementing this synchronization protocol. Furthermore, bitslip mechanisms where developed to allow data alignment, i.e., to determine where the MSB and LSB are in the serial data stream, in addition to the inclusion of delay mechanisms used to overcome setup/hold violations in the registering of data received from the serial lines. All of these mechanisms require proper configuration, whether to determine how much delay should be added for synchronization or to configure the SERDES direction, since communication can be bi-direction.

Thirdly, an AXI interface driver to make use of the Hard-Processing System available in the SoC for configuration of all of the aforementioned parameters from the software layer, through AXI-mediated transactions. The need to be able to configure all of these parameters from a higher abstraction level is justified by the necessity to run more complex algorithms in order to automatically determine the appropriate parameters for each. This requirement will be satisfied by the implementation of a software system composed of system drivers and control software, which will be introduced in the next chapter.

4.4 Control Software Architecture

The main goal of the control software package of QCC is to allow the configuration, calibration and control of the QuMA core and all its supporting hardware. One of the main challenges of its design was to maintain the same high-level interface to QCC as was previously available in CC-Light, in order to allow the effortless transition between the two systems, even after a major change in hardware design, which led to the implementation of QCC in a semi-distributed architecture. To understand the control software architecture of QCC it is helpful to recognize its three main components:

1. Linux Kernel Modules, used to allow interfacing with QuMA's memory and register files from the Processing System embedded in the same SoC, through AXI-based transactions. This will be referred to as the

Device Kernel.

2. Control software running on top of the embedded Linux distribution, available in the Processing System. This software package allows the automatic configuration of all the hardware supporting the QuMA core and creates the server which established the interface to QCC. It will be referred to as the Device Manager.
3. A Python Driver running on the host computer, which is responsible for establishing communication with the QCC device and supporting high-level interfacing with it, for configuration, calibration and control.

To better understand how all the systems work together to achieve the desired functionality, the control flow necessary to perform a quantum computing experiment is presented as a flowchart in Figure 4.7 for the QuTech stack. This chart will serve as a reference to guide our description of the functionality of each of the three main components of the control software system in the following sections.

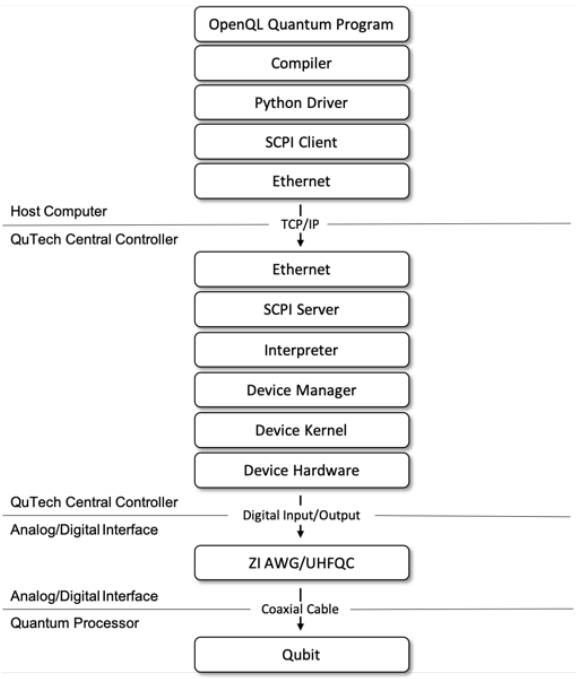


Figure 4.7: Full-stack control flow for a quantum computing experiment

Linux Kernel Modules

In order to allow interfacing with the QuMA core, implemented in the FPGA available in the Xilinx SoC, a custom embedded Linux distribution is run from the Hard Processing System available in the same package. Both systems are setup in a slave/master AXI-based configuration, allowing access to the address space of QuMA from the Processing System.

To allow the Linux Kernel to access this hardware device, a Linux Kernel Module had to be developed to support QCC. Kernel Modules are pieces of code that can be loaded and unloaded into the kernel upon demand. They extend the functionality of the kernel without the need to reboot the system, providing translation between user-level software and the hardware devices implementing the low-level functionality. A diagram of this hierarchy is presented in Figure 4.8.

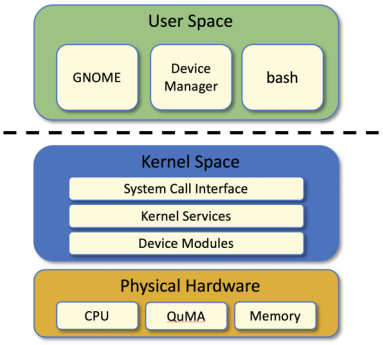


Figure 4.8: Kernel modules are translators between user-level software and hardware devices

Therefore, the Kernel Module developed to support the control of QuMA had to satisfy several requirements.

First, it was necessary to perform the memory mapping of the device memory (instruction memory and control store) to a user's address space. This was done due to a decision to use direct memory mapping to handle memory transfers, whereas the setting and reading of registers was chosen to be done through explicit copying of data from user space to kernel space. This step also required handling of the physical address translation of QuMA.

Then, to handle the setting and reading of registers in QuMA, methods were defined to perform copying of data from user space to kernel space and back,

in addition to validating all variables according to their allowed values.

Finally, the Kernel Modules needed to define all IOCTL system calls for device-specific input/ouput operations. These instruct the Linux Kernel on how to deal with device-specific system calls for all functionality related to QuMA, from the uploading of the instruction program and control store, to the setting of delays for inter-device communications, to starting and stopping the device and providing version information on the firmware.

Control Software

The Control Software package running on the SoC's processing system creates the outside interface of the QCC, in addition to configuring all of the internal systems to perform the required functionality. Therefore, the Control Software is made-up of several individual packages, as shown in Figure 4.7. In particular, it is composed of the Device Manager, the Interpreter and the SCPI server. In the next paragraphs, each of these sub-systems will be analyzed, to describe their functionality and the design decisions that went into each.

The SCPI server provides the external interface of QCC. SCPI stands for Standard Commands for Programmable Instruments, and it defines a standard for syntax and commands to use in controlling programmable test and measurement devices. Therefore, it provides a *Transmission Control Protocol - Internet Protocol* (TCP/IP) network socket where a host computer can connect to send and receive commands using a SCPI client.

The Interpreter works in parallel with the SCPI server, to interpret the SCPI commands received by the server and trigger the appropriate replies. Therefore, it defines all commands that QCC responds to, including all commands necessary to upload and retrieve the instruction memory and control store, and to set all QuMA registers. To implement these commands, it makes use of the functionality provided by the Device Manager to trigger IOCTL system calls, which are handled by the Linux Kernel Module to interface with the QuMA core implemented in hardware.

The Device Manager is the main software engine of QCC. It is responsible for defining all the methods that implement the functionality required to handle SCPI commands received by the QCC. To do this, it makes use of the system calls defined by the Linux Kernel Module to access functionality provided by the QuMA core. In addition to this, the Device Manager is responsible for configuring the QuMA core and all supporting infrastructure when the QCC device is initialized. In particular, the Device Manager launches three pro-

cesses on start to achieve this.

First, a process is launched to lock and monitor the PLL, a hardware unit available in the Xilinx SoC and responsible for generating the internally used clocks, from the external reference clock distributed to all hardware devices.

Then, a ZeroMQ-based high-performance asynchronous communication system is launched to provide communication capability between all CORE and IO boards available in QCC. Such a scheme allows maintaining a single interface point to QCC, while providing the ability to interface to all of the boards that make up the device individually. This is done by passing SCPI commands received by the CORE SoC through to the IO SoC, where they are handled by a similar command interpreter. Furthermore, this intra-QCC communication system will be critical to QCCs ability to automatically configure itself on initialization. The internal structure of QCC device modules is represented in Figure 4.9, highlighting all inter-board communication.

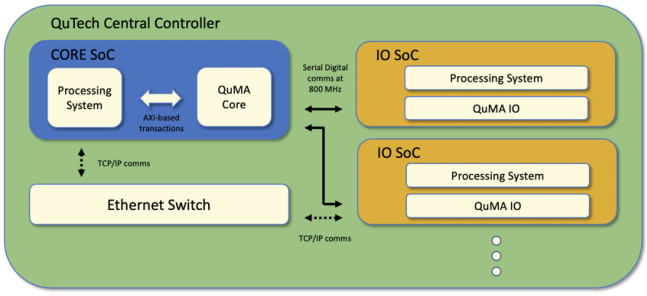


Figure 4.9: Internal structure of QCC device modules

Finally, a configuration process is launched. This process will take care to automatically configure the SERDES infrastructure according to the functionality of each IO board. For example, an IO board responsible for interfacing with a UHFQC should have bi-directional serial communication with the CORE, whereas for a IO board interfacing with a AWG, only a uni-directional serial signal is required. Furthermore, a algorithm is launched to configure the SERDES delay steps, used to avoid HOLD/SETUP violations in the input registers to the serial lanes. The algorithm starts after all boards communicate to indicate that the initial configuration of the PLL and SERDES directions as been successfully concluded, since only then can the delay configuration start. It works by sequentially going through the 32 delay steps of 39 pico-seconds available, each time performing a count of pulses successfully received from the other side, to determine if synchronization was successful. Due to the

aforementioned potential timing violations on the input registers, a set of delay values will lead to improper synchronization. The algorithm will obtain these through access to QuMA's registers and determine the best delay value to use.

Python Driver

Having completed the development of all control software running inside of QCC, attention was focused on the software required to communicate and control QCC from a host computer. This was done by creating a Python driver class responsible for implementing a SCPI client, capable of interfacing with the SCPI server available in the QCC.

In addition to completing the high-level interface to the QCC device, the Python driver is also responsible for obtaining the QISA program and the contents of the control store from the OpenQL compiler, required to properly configure QCC for a specific experiment. Furthermore, this driver completes the assembling of the QISA program to be uploaded. It should again be noted that the Control Store holds the microcode, which determines how each operation is translated into device events, and that the QISA program contains the compiled program, expressed in instructions understood by QuMA.

Having concluded the description of the Control Software Architecture of QCC, it is now possible to understand the full flow of control operations required to run an experiment through the QuTech stack. This flow is schematically represented in Figure 4.10.

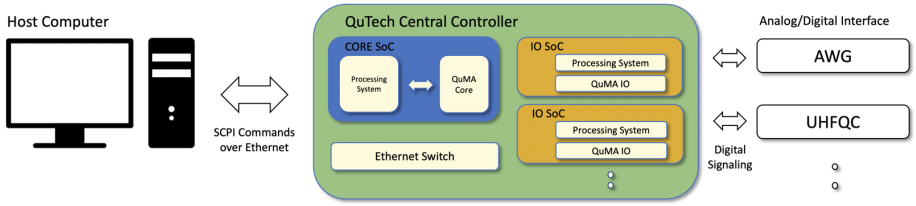


Figure 4.10: Internal structure of QCC device modules

4.5 Summary

The following is a detailed description of the main modifications performed to the QuMA microarchitecture, the Central Controller device and the control software stack in order to support the control of 17 qubits. It is the combination of these three systems that make the QuTech Central Controller.

- Development of the Linux Kernel Drivers used to create the interface between the QuMA core and the Hard Processing System;
- Development of the Control Software responsible for implementing the external interface of QCC and for managing the low-level configuration of the device parameters.
- Added serialization infrastructure to QuMA core for the implementation of an inter-board serial communication protocol;
- Design of IO Core responsible for the de-serialization and tailoring of serial signals received from the QuMA Core;
- Expansion of the control and datapaths on the QuMA Core for operation of up to 17 qubits;
- Reformatting of SMIT instruction for expansion of indirect addressing mechanism to 17 qubits, including modifications to the assembler, the quantum instruction decode unit and the qubit mask management units;
- Expansion of Timing Control Unit to additional UHFQC and AWG devices used, respectively, for measurement and flux control;
- Removal of vendor specific IP and constructs and migration of project design files;
- Rebuild of Hard Processing System and AXI bus interconnect on new hardware platform;

5 Testing and verification

This chapter presents a compilation of all the verification and testing routines performed for the QuTech Central Controller.

In the Verification section, focus will be given to the tests performed at several stages of the development of QCC, either to certify viability of a particular design solution, or to verify correct functionality of the implemented sub-system. Therefore, the chapter will cover tests performed after the migration to a different platform, to ensure that functionality was maintained, and then follow each of the design steps of the development of QCC, to ensure that the behavior expected from the system was observed. In particular, the correct operation of the expanded QuMA core will be tested, followed by testing of the SERDES infrastructure and of the drivers and control software developed for the system.

The Testing section will instead describe the tests performed on the system after development was completed, so to determine whether or not functionality was developed as per requirement. Therefore, tests will be, for the first time, performed on the final hardware platform of QCC, and will see this system integrated in the quantum control rack currently used in the DiCarlo laboratory at QuTech to perform quantum computing experiments.

5.1 Verification

Platform Migration

Having completed the migration of all design files to the Xilinx Design Environment, as well as the re-generation of the Processing System and all vendor-specific IP, attention was focused on the verification of these changes. Due to the unavailability of behavioral models for the Processing System, it was impossible to simulate this modification. However, attention had been previously

given to ensure the compatibility of the design, which was possible given the use of a standardized, AXI-based communication scheme between the Hard Processing System and QuMA.

However, it was possible to test the regenerated vendor-specific IP, consisting mainly of First-in first-out memory elements. A behavioral model of these memory elements was used to integrate them into the VHDL simulation environment used to verify correct operation of the QuMA core. This simulation environment consists of the behavioral models describing all the units that QuMA is composed of, in addition to the set of scripts and signal drivers required to perform the simulation, and is powered by a QuestaSim simulation engine [22].

In particular, this environment allows the simulation of all aspects related to the operation of QuMA. From the system reset operation representing its start, to the uploading of the control store and instruction memory, to the generation of a sequenced set of codewords to send to the devices making up the analog/digital interface. Furthermore, it is possible to simulate feedback instructions through the inclusion of behavioral models emulating the operation of the measurement equipment, therefore allowing the simulation of complex functionality like fast-conditional execution and memory writing and reading.

To verify correct operation of the system, a synthetic quantum program was written to include the triggering of operations on all quits at consecutive times. Such a quantum program would guarantee the need to use all memory elements for correct operation, therefore allowing the test of all these through the comparison of the output from the timing control unit, to the expected sequence of gates.

Upon initial testing, a problem was detected with the system since the execution of such a program led to the loss of the initially triggered codewords. After careful examination, the problem was individuated to the FIFO memory elements composing the timing control unit. Indeed, a difference in reset behavior from these memory elements to the ones previously used in the Altera platform, lead to the loss of data first transmitted to them. This erroneous behavior is observed in the simulation result presented in Figure 5.1, where all the internal signals of the FIFO used for time-stamps are displayed.

In particular, attention should be focused on the *din*, *wr_en* and *wr_data_count* buses, used respectively for supplying the data to be stored, providing the valid signal for the storing of data present on the *din* bus and counting the number of time-stamps stored. It should be observed that, between 11.4 us and 11.5 ns of simulation, six time-stamps are supplied (in bus *din*) for storage while the

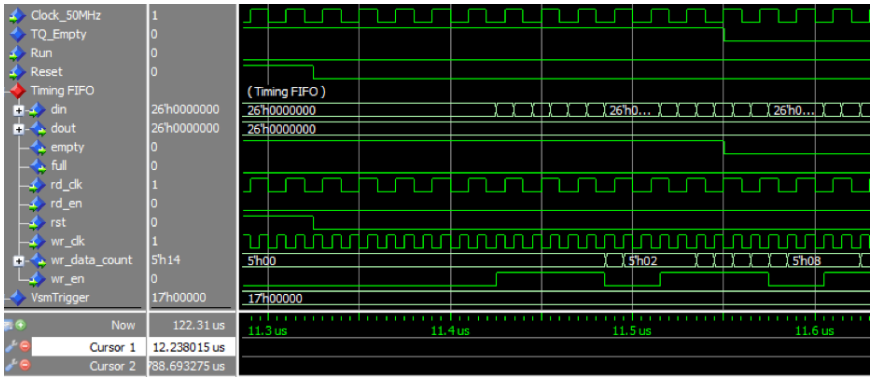


Figure 5.1: Evidence of erroneous reset behaviour observed for FIFOs regenerated for Xilinx platform

wr_en signal is asserted high. However, only two of those six time-stamps are actually stored in the FIFO, as indicated in the *wr_data_count* bus at the 11.5 ns mark. This lead to the loss of the initial 4 time-stamps and explains the loss of several of the initially triggered codewords, since time-stamps are used to trigger operations at a precise time.

After close analysis of the operation of the FIFO memory elements, it was determined that this loss of information was explained by the need to wait for a set number of cycles after reset, before proper operation could be ensured. Indeed, the memory elements generated for the Xilinx platform assumed a period after reset where its use was not allowed, and had done so without assertion of the full flag, commonly used to signal incapability to store more information. Therefore, the unit responsible for managing the timing control memory elements was unable to stall the pipeline until storing on the FIFO was allowed, and data was lost.

To correct this error, the memory elements were regenerated with an updated reset behavior. After compilation of the new design, simulation with the same synthetic program now led to correct operation, as can be observed in Figure 5.2. Indeed, the storing of the first 6 time-stamps is now successful, with the *wr_data_count* bus now reading 6 fully written values. Furthermore, all the outputs of the timing control unit displayed the expected set of sequenced codewords, guaranteeing correct operation of the entire pipeline.

This set of tests allowed verification of the correct operation of the QuMA core after migration of platforms and led to the task of expanding the QuMA core to support control of 17 quits, work which was detailed in Section 4.2.

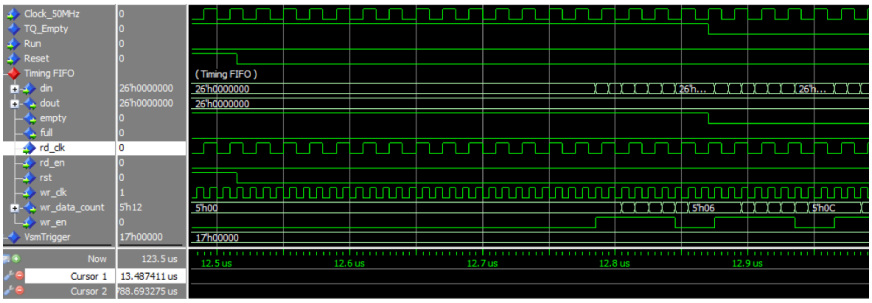


Figure 5.2: Reset behaviour obtained for FIFOs after correction

QuMA Core Expansion

To verify correct operation of the system after expansion of the QuMA core to control 17 qubits, a synthetic quantum program was written to include the triggering of operations on all quits at consecutive times. This was done to guarantee that maximum stress was placed on the instruction issue unit, consisting of the instruction fetch and decodes units. In this way, it would be possible to determine if the measures taken to address the instruction issue rate problem (concretely VLIW architecture, efficient timing specification and indirect qubit addressing) were sufficient for the control of 17 qubits by simply comparing the outputs of the timing control unit to the expected sequenced set of gates.

In particular, attention will be given to the VSM control signals in Figure 5.3, for simplicity in describing the observed behaviour, even though all output signals were individually checked.

The first two signals in Figure 5.3 represent the clock driving the output of the timing control unit and the instruction currently in the Instruction Decode Unit of QuMA, respectively. For reference, it should be noted that the output of the Timing Control Unit is driven on cycles of 20 ns due to the requirements of the digital protocol used to communicate with the analog/digital interface. The three signals after those represent the status of the pipeline (valid instruction, invalid instruction or stall of the pipeline), the digital representation of the current instruction being decoded and the disassembled instruction, respectively.

Therefore, the program consists of the triggering of both microwave AWGs and their respective channels on the VSM, as can be seen in signal *CurrentDisas*, which represents the instructions being processed, in a human-readable manner. In particular, *CW_01* represents a microwave operation, and *s0* is the

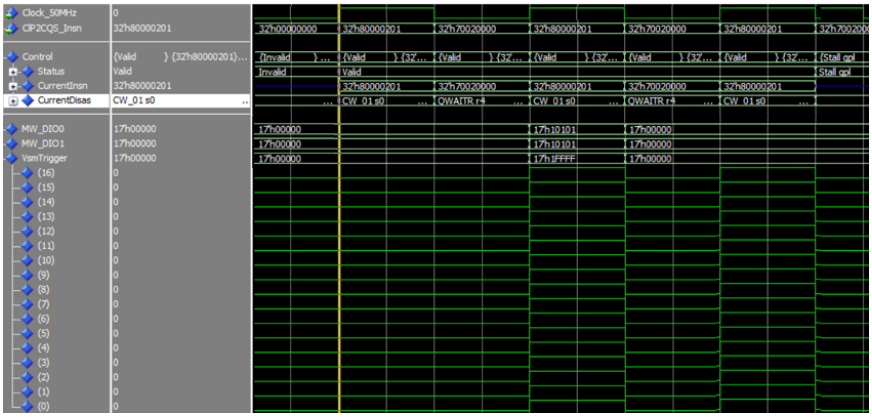


Figure 5.3: Sequenced gates for VSM control program simulation on the expanded QuMA core

target register holding the qubit addresses to which the operations will be applied.

Since operations are to be applied on all qubits, it is expected that all 16 channels of the VSM should switch on when an operation is triggered. Indeed, that is what we observe with the *VsmTrigger* signal switching on a cycle after the first instruction is received (time marked in yellow in Figure 5.3). It should be noted that the latency between the reception of an instruction and the switching of the output signal is due to these being governed by different stages of the pipeline.

In order to correctly observe the disassembled instructions for the sequencing of quantum operations, only a small time windows from the simulation was taken. However, simulation over several milliseconds proved the correct operation of the entire pipeline, by displaying a digital set of pulses consistent with the consecutive set of operations triggered by our quantum program.

Having verified the modification done to the QuMA core to extend its control to 17 qubits, attention was shifted to the verification of the SERDES infrastructure, used to bridge the CORE and IO boards together, to allow the implementation of the full QCC system architecture.

Serial Communication

In order to test the serializer/de-serializer infrastructure, a simulation of the communication between two of these modules was first simulated. In Figure

5.4 we can see the result of this simulation.

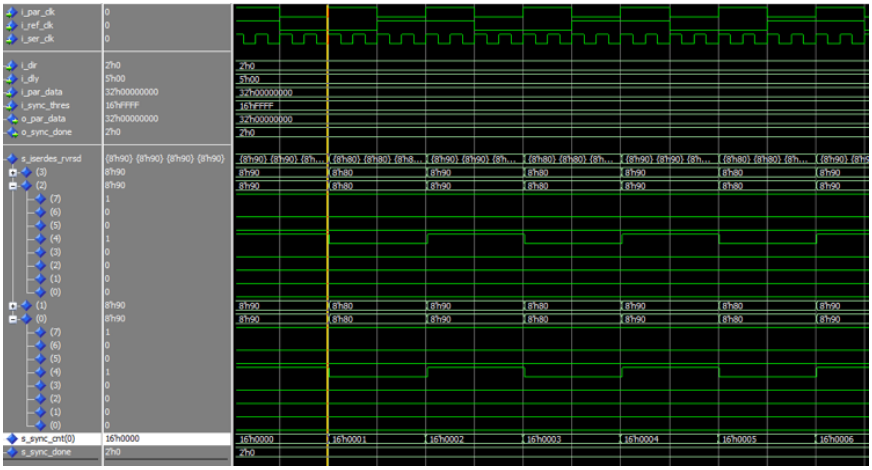


Figure 5.4: Simulation of SERDES synchronization protocol.

The first set of signals represent the three clocks used by the SERDES. Signal *i_ref_clk* is the 50 MHz clock signal used to power the entire timing control unit; *i_par_clk* is the 100 MHz signal used to sample the data signals and *i_ser_clk* is a 400 MHz signal used to drive the Double Data Rate units, responsible for creating the 800 MHz signal used to connect the CORE and IO boards.

The second set of signals corresponds to the SERDES configuration signals. These include *i_dir* which encodes the direction (input or output) of the SERDES, *i_dly* which represents the added delay (for calibration of the serial signals) and *i_sync_threshold*, the threshold value used to assert proper synchronization of serial delays. These are the signals affected upon by the control software to configure the SERDES infrastructure. Additionally, the second set of signals include the parallel data (*_par_data*), i.e. the information to be serialized, the de-serialized information recovered on the output side (*o_par_data*) and a bit to assert when correct synchronization was found (*o_sync_done*).

Finally, the third set of signals represents the 4 ((0) to (3)) serial lanes used for communication between serializers and de-serializers. The most important for the purpose of this test are signals in bus (0) and (2), used for sending a train of pulses from the serializers to the de-serializers, in order to allow the configuration of the calibration delays.

Indeed, signal *s_sync_cnt* represents the count of pulses properly received by the de-serializer, which is compared against the threshold count and used to

indicate when correct synchronization between the SERDES was achieved.

Therefore, a simple test were no input is supplied to the SERDES, allows the visualization of this synchronization protocol, with an increasing count of pulses received in signal *s_sync_count* proving the correct functioning of the serializers synchronization algorithm. In Figure 5.4, the assertion of signal *s_sync_count* is not seen as the threshold used was too high to display the entire count on a single image.

Making use of Chipscope, a tool which allows us to probe the signals in an FPGA, we can now see in actual synthesized hardware the mechanism for the serialization of information working. This is shown in Figure 5.5.

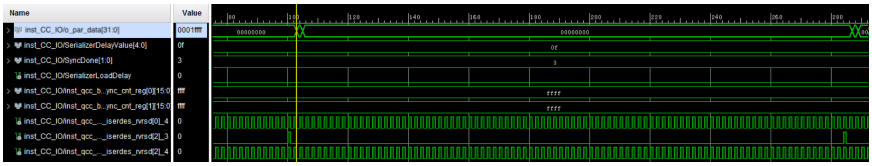


Figure 5.5: Chipscope image represents signals received from IO in experiment and additional configuration signals

It should be noted that the algorithm used by the driver to synchronize the serial links asserted 15 (ox0f) as the best value for the serial line delay, as can be seen in signal *inst_CC_IO/SerializerDelayValue*. This lead to the count of pulse trains represented in signals *inst_CC_IO/inst_qcc_..._iserdes_rvrsd[0]_4* and *inst_CC_IO/inst_qcc_..._iserdes_rvrsd[2]_4*. Since synchronization was correct, this count surpassed the threshold value and lead to both *SyncDone* bits being asserted high. Signal *inst_CC_IO/inst_qcc_..._iserdes_rvrsd[2]_3* should be ignored, as it was only used for proper triggering of Chipscope.

Running the synthetic program described in the previous section, which should consecutively trigger all 16 bits of the VSM, we can use Chipscope again to observe that all 16 of these signals are properly received at the IO board, as shown in Figure 5.6. This demonstrates correct operation of SERDES infrastructure.

Control Software and Drivers

To conclude this section on the verification of QCC, focus will be shifted to the test of the control software and drivers used to interface with the device. To achieve this, experiments will be run from a computer, connecting to QCC and executing on the analog/digital interface. To ensure that correct results are

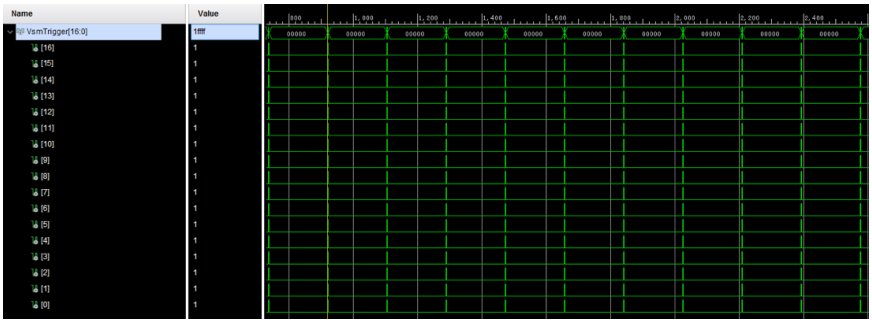


Figure 5.6: Chipscope image from VSM IO board showing proper reception and alignment of all signals generated from All-VSM program

obtained, the analog lines will be monitored using an oscilloscope.

In particular, staircase experiments will be used since they provide a simple way to visually verify the results. A staircase experiment consists of repeatedly triggering consecutive codewords, each associated to a pulse of decreasing (or increasing) amplitude. The results should therefore look similar to a staircase.

A Python script was written to connect to QCC from a measurement computer and assemble such a QISA program to run. First, a program triggering consecutive codewords on the AWG system was written. Having connected an oscilloscope to the analog channels of this device, the waves shown in Figure 5.7 were observed.

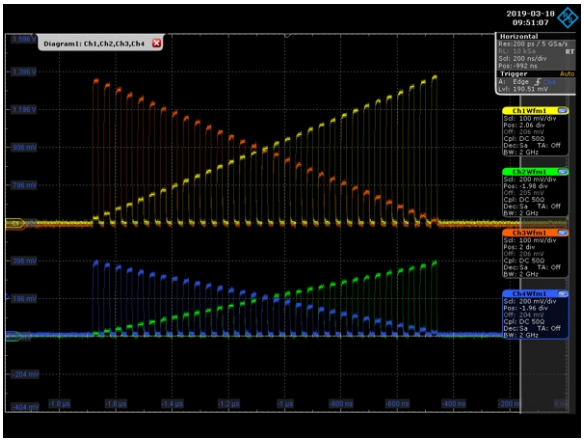


Figure 5.7: Microwave drive from a staircase experiment running on QCC and a ZI AWG-8

This proved for the first time that the drivers were capable of interfacing with and controlling the QCC and that the QCC itself was capable of triggering the AWGs at precise times.

To achieve a similar result in controlling the UHFQCs, used to measure qubit, a program was written to trigger consecutive measurements of different qubits. Since a UHFQC works by playing a measurement wave through a feedline and reading back the result, associating different qubits to measurement waves of increasing amplitudes should lead to a similar staircase being observed on the feedlines. Indeed, upon connecting the scope to these feedlines, the staircase shown in Figure 5.8 was observed.

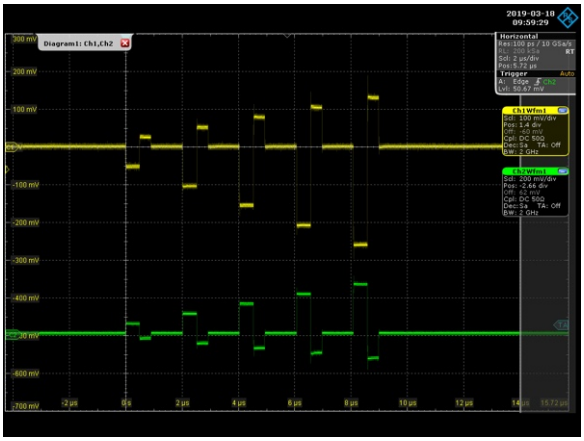


Figure 5.8: Readout pulses from a staircase experiment running on QCC and a ZI UHFQC

5.2 Testing

After verifying basic functionality of the system, the next step was to build the final hardware platform implementing the QuTech Central Controller.

This consisted of 9 Xilinx Zynq-7000 SoC platforms to implement the IO boards, a single similar SoC board implementing the CORE, and a backplane implementing the serial communication system, in addition to an Ethernet switch, which guaranteed TCP/IP communication between all boards.

An image of the final system can be seen in Figures 5.9 and 5.10, with the IO boards occupying the top half of the second image and the single CORE occupying the bottom. Additionally, it is possible to see the (detached) cooling strip

and the power supply unit (bottom left), responsible for maintaining thermals and supplying power to all systems.

Figure 5.9: Front view of the enclosure of QCC (Vlothuizen W.J., QuTech)

The disposition of all IO boards side-by-side on the back of the enclosure was done so to achieve high density of QCC interfaces to the analog/digital instrumentation. Each of the large vertical connectors presented in Figure 5.11 provide the digital interface running at 50 MHz required to interface with each of the UHFQCs and AWGs.

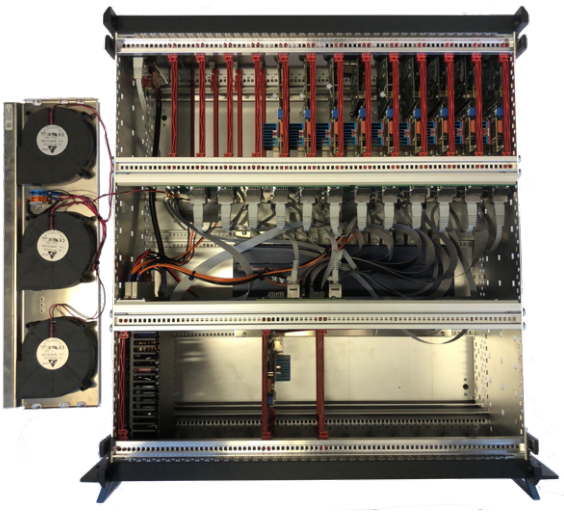


Figure 5.10: Top view of the enclosure of QCC, complete with all cooling, power, ethernet, core and io boards (Vlothuizen W.J., QuTech)

Finally, to allow running quantum information processing experiments on superconducting qubits, the QCC system was integrated in the full control rack used within QuTech, which is depicted in Figure 5.12. This rack is composed (top to bottom) of the host computer, the radio-frequency sources used to generate clocks and signals for modulation, the AWGs, a set of mixers used to

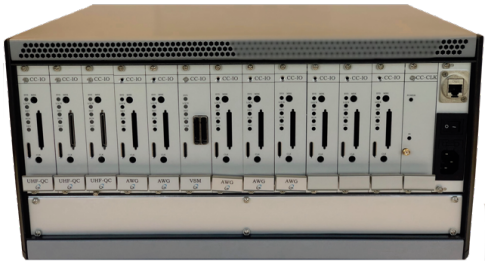


Figure 5.11: Backside view of the enclosure of QCC, highlighting all of the I/O available for interfacing with the analog instrumentation (Vlothuizen W.J., QuTech)

generate control signals, the Central Controller itself, the Vector Switch Matrix used to route signals, the UHFQCs and, finally, a set of amplifiers used to drive final signals to the qubits.

Having assembled the full system required for quantum control, the final step was to perform experiments controlling actual qubits, to test the QCC under normal working conditions. Attention was focused on four experiments, aimed at highlighting all possible control scenarios, in an attempt to show any potential bugs or shortcoming of the architecture under test.

To test calibration routines, Qubit Spectroscopy experiments were first run. These were followed by Randomized Benchmarking experiments, aimed at showing single-qubit control capability. Then, Simultaneous AllXY experiments were used to prove multi-qubit control. Finally, Chevron experiments allowed proof of two-qubit control capability. A description of these experiments and their results will be the subject of the next sections.

It should be noted that all experiments were performed on a Surface-7 quantum processor since no Surface-17 quantum chips were available for testing at the time of writing. However, this allowed the use of CC-Light, the control architecture developed for Surface-7, to create a baseline of performance expected from the quantum processor, allowing confirmation of the results obtained with QCC.

Indeed, all results presented in the following sections were validated, proving correct implementation of the extended control microarchitecture and the capacity of QCC to control quantum computing experiments.

Figure 5.12: Complete quantum control system used to perform quantum computing experiments (J.C. de Sterke, QuTech, 2018)

Calibration Experiments

Before any experiment with qubits is possible, it is necessary to determine the frequencies at which each of the qubits in the quantum processor is operating. This is part of the calibration routines that need to be performed before a quantum computer is ready for use.

The aim of this first experiment was to prove that QCC could be used to perform these routines. In particular, for the calibration of qubit frequencies, spectroscopy experiments were used, in which the QCC generates a set of microwave pulses followed by a measurement. By changing the frequency with which each of these pulses is modulated, it is possible to determine at which frequency the qubit sits by observing the measurement results.

The QASM file used to program the QCC to perform this experiment is presented in Figure 5.13.

```
1  SMIS s0, {0}
2
3  k_main:
4      1     cw_08 s0
5      1     cw_08 s0
6      1     cw_08 s0
7      1     cw_08 s0
8      1     cw_08 s0
9      1     MEASZ s0
10     QWAIT 300
11
12     BR ALWAYS, k_main
13     NOP
14     NOP
```

Figure 5.13: QASM program used to perform spectroscopy experiments.

After running this experiment on a particular qubit, a graph such as the one presented in Figure 5.14 was plotted. It can clearly be seen that the qubit was found at 5.882 GHz, a value consistent with what was obtained with the CC-Light for the same qubit.

Single-qubit Control

Having concluded the calibration experiments, attention was focused on Randomized Benchmarking experiments. Randomized Benchmarking was chosen as an experiment that is capable of demonstration single-qubit control. In particular, this experiment is made up of 3000 rounds of different arrangements of operations, each composed of sequences of different gates. The number of

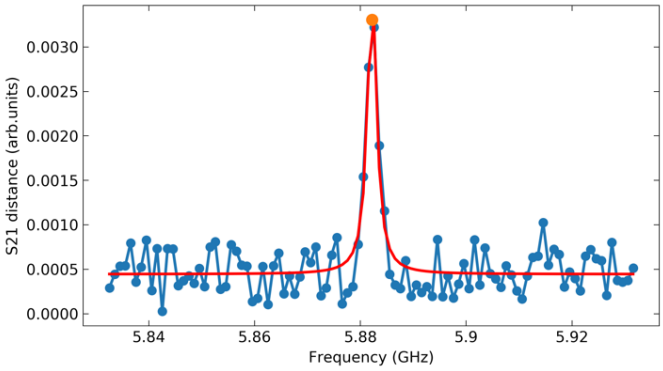


Figure 5.14: Plotted results from the qubit spectroscopy experiment.

operations in these sequences varies from 2 to 2000. Furthermore, each of the groups of operations is especially crafted to add to identity and should have no effect on the resulting state of the qubit. Therefore, randomized benchmarking is commonly used to assess the fidelity of single-qubit operations.

The QASM file used to program the QCC to perform this experiment is presented in Figure 5.15.

```
1  SMIS s0, {0}
2
3  round_0:
4      1    prepz s0
5      QWAIT 4999
6      1    cw_01 s0
7      1    cw_09 s0
8      1    MEASZ s0
9      QWAIT 300
10
11 round_1:
12     1    prepz s0
13     QWAIT 4999
14     1    cw_03 s0
15     1    cw_04 s0
16     1    cw_03 s0
17     1    MEASZ s0
18     QWAIT 300
19 ...
```

Figure 5.15: QASM program used to perform Randomized Benchmarking experiment.

The results from running this experiment are presented in Figure 5.16 and

show a single qubit fidelity of 99.5%, a result that is not representative of the capability of the quantum processor but that is consistent with the value obtained by CC-Light. Successful completion of this experiment demonstrated QCC's capability to implement single-qubit control and readout.

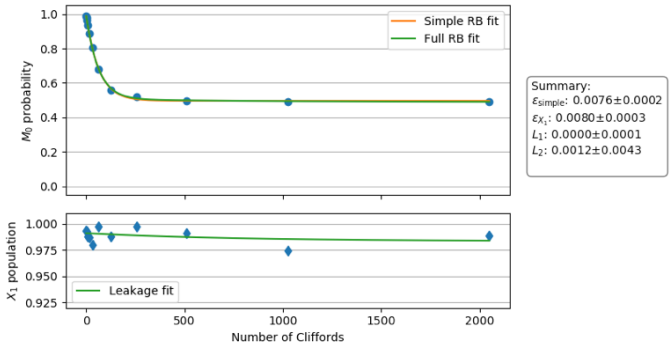


Figure 5.16: Plotted results from the randomized benchmarking experiment.

Multi-qubit Control

Having demonstrated single-qubit control, an AllXY experiment was used to prove the capability of QCC to implement multiple qubit control. In particular, the AllXY experiment consists of applying pairs of gates to qubits in such a way that the expected measurement outcomes produce a characteristic staircase pattern. This is done using combinations of gates from the Pauli set, in addition to the identity gate.

The QASM file used to program the QCC to perform this experiment is presented in Figure 5.17.

The results of the experiment are plotted in Figure 5.18, where the top image represents the results from the first qubit, and the bottom image the results from the second qubit being controlled in parallel. In the first image, a line is represented in orange where results were expected to fall, showing a deviation from the expected results. However, since similar results were obtained for CC-Light, the observed deviation was attributed to problems unrelated to the QCC and likely associated to the calibration of the analog equipment used to interface with qubits. Therefore, this experiment successfully demonstrated the capability of QCC to drive multiple qubits.

```
1  SMIS s0, {1, 7}
2  SMIS s1, {1}
3  SMIS s2, {7}
4
5  k_AllXY_1:
6      1    prepz s0
7      QWAIT 4999
8      1    cw_00 s0
9      1    cw_00 s0
10     1    MEASZ s0
11     QWAIT 300
12
13 k_AllXY_2:
14     1    prepz s0
15     QWAIT 4999
16     1    cw_00 s0
17     1    cw_01 s1 | cw_00 s2
18     1    MEASZ s0
19     QWAIT 300
20
21 k_AllXY_3:
22     1    prepz s0
23     QWAIT 4999
24     1    cw_01 s1 | cw_00 s2
25     1    cw_02 s1 | cw_00 s2
26     1    MEASZ s0
27     QWAIT 300
28 ...
```

Figure 5.17: QASM program used to perform AllXY experiment.

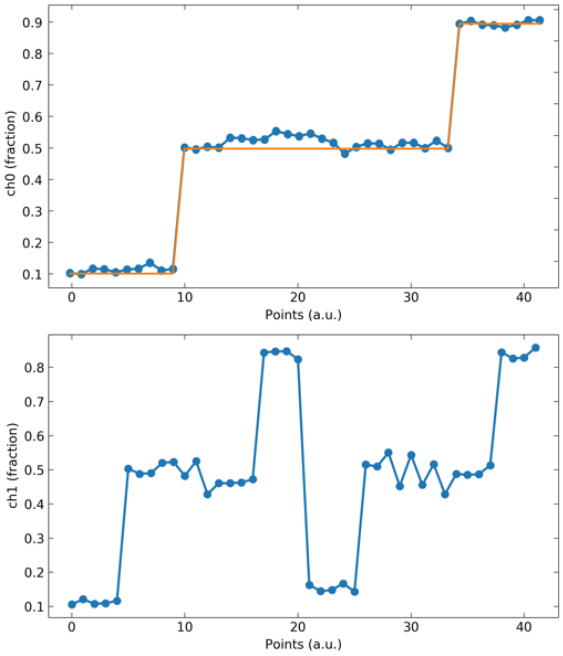


Figure 5.18: Plotted results from the AllXY experiment.

Two-qubit Control

Finally, a Chevron experiment was used to prove the capability of QCC to implement two-qubit control. The specifics of this experiment are beyond the scope of this thesis but its purpose is to perform flux-controlled excitation swapping. Therefore, it makes use of two-qubit operations, allowing the demonstration of two-qubit control using QCC. Furthermore, it should be noted that a chevron pattern is expected from each of the qubits targeted in this experiment and that the excited state probabilities of the two should be inverted in these diagrams.

The QASM file used to program the QCC to perform this experiment is presented in Figure 5.17.

```
1  SMIS s0, {1}
2  SMIS s1, {7}
3  SMIS s2, {1,7}
4  SMIT t0, {(10, 8)}
5
6  k_chevron:
7      1    prepz s0
8      2    prepz s1
9      QWAIT 4997
10     1    cw_01 s0
11     2    cw_01 s1
12     1    fl_cw_02 t0
13     QWAIT 9
14     1    cw_01 s0
15     1    MEASZ s2
16     QWAIT 300
17
18     BR ALWAYS, start
19     NOP
20     NOP
```

Figure 5.19: QASM program used to perform Chevron experiment.

The results of this experiment are plotted in Figures 5.20 and 5.21, and show the expected patterns out of both qubits used in this experiment. Therefore, successful completion of this experiment demonstrated QCC's capability to implement two-qubit control, leading to the successful completion of all tests performed to QCC.

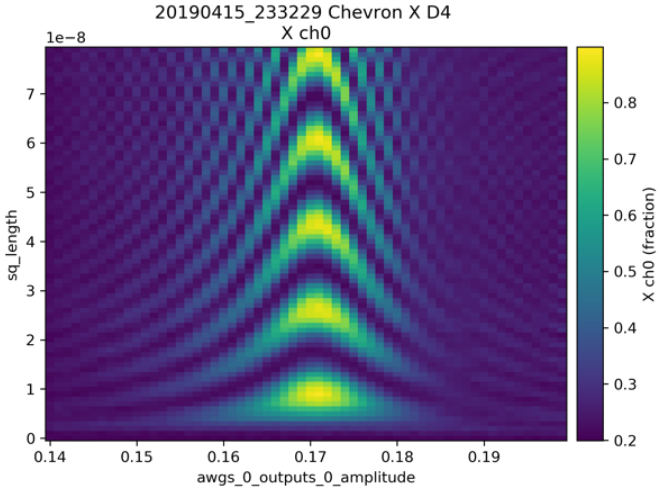


Figure 5.20: Plotted results from the Chevron experiment on the first qubit.

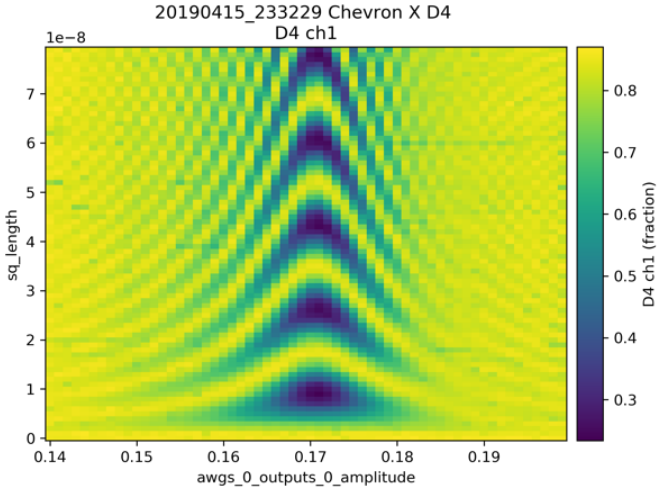


Figure 5.21: Plotted results from the Chevron experiment on the second qubit.

6 Conclusion and Outlook

This chapter presents a summary of the work developed during this thesis, in addition to an analysis of the results and contributions made with this work. Furthermore, close examination of the limitations of the system developed during this thesis and its implications on the future of quantum control will be provided. The chapter will conclude with a set of predictions and recommendations in an outlook onto the future of the field of quantum control.

6.1 Conclusion

The objective of this thesis was the design and development of a control architecture based on the QuMA model and capable of controlling a Surface-17 quantum processor.

To achieve this, the QuMA control microarchitecture was first expanded to allow the control of 17 qubits. This lead to the redesign of the mechanisms used to implement indirect qubit addressing, the extension of the *Timing Control Unit* as well as the expansion of the datapath and control.

Next, the design was migrated to a new hardware platform, capable of supporting the additional interfaces required for the Surface-17 control scheme. Furthermore, the system architecture of a central controller was re-imagined to include several FPGA packages implementing a semi-distributed architecture. In this platform, high-speed serial communication links between these packages were used to overcome the limitation in I/O capability associated with the initial monolithic design.

Then, a control software architecture was created to allow automatic low-level configuration of the developed design and to support external interfacing and control, allowing the use of this device as part of the existing control systems

employed for quantum computing experiments.

Finally, the aforementioned hardware and software designs were implemented in a device called the QuTech Central Controller and extensively tested in quantum information processing experiments with superconducting qubits. Successful completion of calibration routines, in addition to demonstrations of single-qubit, two-qubit and multi-qubit control in these experiments constituted sufficient proof of correct operation of the device.

Overall, this thesis successfully demonstrated that the QuMA control architecture could be expanded to control a Surface-17 quantum processor and provided a first step in incorporating the new control architecture into the systems used to conduct superconducting quantum computing experiments.

6.2 Outlook

The research presented in this thesis opens a number of research lines that should be explored in the future. These are motivated both by the challenges presented by the design developed in this thesis and by the requirements envisioned for a next-generation quantum control system, which should foment new developments in the field of quantum computing. In this section, I will draw on the lessons learned while developing this thesis to provide a set of recommendations on the research lines that should be addressed in the future.

A big part of the design of QCC has concerned overcoming the limitations presented by a centralized architecture. Whether it be the instruction issue rate problem, addressed with techniques such as VLIW, or the input/output scaling challenge, addressed through the serialization of signals in a semi-distributed system architecture, there are several ways in which a centralized architecture has fundamentally limited the scalability of the control system. For continued developments on the field of quantum computing, it is important that research and development efforts are focused on distributed architectures, as a means to more fundamentally address the issue of scaling control in the future. However, doing so will raise several challenges, both at the control system level, with the need for truly scalable and high-performance communication and synchronization, and at the compiler level, with the need to compile for multiple targets. The solution to these challenges will define how quantum control is exerted and fundamentally determine what will be achievable with these systems in the future.

Another important step to guarantee that the field of quantum control contin-

ues to evolve to support cutting edge research on quantum computing is the development of a fully-integrated and modular control system. Full integration of all instrumentation required for quantum control, from the sequence processor of quantum operations, to the generation of analog waveforms and the amplification and modulation devices required for tailoring control pulses to individual qubits, will allow the creation of the first standardized unit cell for quantum control. Achieving this will greatly simplify the schemes currently used for quantum control and address concerns on the maintainability and reliability of these systems. Furthermore, full integration of these systems will help reduce their footprint and cost, providing a more scalable scheme to control ever greater numbers of qubits.

An outlook into the future of quantum control would not be complete without a reference to Quantum Error Correction. QEC encompasses several techniques employed to mitigate the fragility of coherent quantum systems [23], as a way to allow the development of large scale quantum computers. However, it has long been understood that for large quantum systems these techniques are not feasible for software-managed error correction schemes [24]. Therefore, it is imperative to develop hardware-managed error correction schemes to tackle this challenge. Any future control microarchitecture should no doubt support these.

Bibliography

[1] R. P. Feynman, "Simulating physics with computers," *International Journal of Theoretical Physics*, vol. 21, pp. 467–488, 1982.

[2] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge University Press, 2010.

[3] D. P. DiVincenzo, "The physical implementation of quantum computation," *arXiv:quant-ph/0002077*, 2000.

[4] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, pp. 1484–1509, 1997.

[5] I. Kassal, J. D. Whitfield, A. Perdomo-Ortiz, M.-H. Yung, and A. Aspuru-Guzik, "Simulating chemistry using quantum computers," *Annual Review of Physical Chemistry*, vol. 62, pp. 185–207, 2011.

[6] M. Steffen, W. van Dam, T. Hogg, G. Breyta, and I. Chuang, "Experimental implementation of an adiabatic quantum optimization algorithm," *Phys. Rev. Lett.*, vol. 90, p. 067903, Feb 2003.

[7] R. O. L. S. I. B. F. S. G. V. P. M. Andrea Crespi, Roberta Ramponi, "Integrated photonic quantum gates for polarization qubits," *Nature Communications*, vol. 2, Nov 2011.

[8] J. I. Cirac and P. Zoller, "Quantum computations with cold trapped ions," *Phys. Rev. Lett.*, vol. 74, pp. 4091–4094, May 1995.

[9] R. Hanson, L. P. Kouwenhoven, J. R. Petta, S. Tarucha, and L. M. K. Vandersypen, "Spins in few-electron quantum dots," *Reviews of Modern Physics*, vol. 79, pp. 1217–1265, 2007.

[10] L. Childress and R. Hanson, "Diamond nv centers for quantum computing and quantum networks," *MRS Bulletin*, vol. 38, no. 2, p. 134–138, 2013.

[11] M. H. Devoret and R. J. Schoelkopf, "Superconducting circuits for quantum information: An outlook," *Science*, vol. 339, no. 6124, pp. 1169–1174, 2013.

[12] N. Khammassi, G. G. Guerreschi, I. Ashraf, J. W. Hogaboam, C. G. Almudever, and K. Bertels, "cQASM v1.0: Towards a common quantum assembly language," *arXiv:1805.09607*, 2018.

[13] A. JavadiAbhari, S. Patil, D. Kudrow, J. Heckey, A. Lvov, F. T. Chong, and M. Martonosi, "ScaffCC: Scalable compilation and analysis of quantum programs," *Parallel Computing*, vol. 45, pp. 2–17, 2015.

[14] D. S. Steiger, T. Häner, and M. Troyer, "ProjectQ: an open source software framework for quantum computing," *arXiv:1612.08091*, 2016.

[15] D. Wecker and K. M. Svore, "LIQ$Ui\,|\rangle$: A software design architecture and domain-specific language for quantum computing," *arXiv:1402.4467*, 2014.

[16] N. Khammassi, I. Ashraf, X. Fu, C. G. Almudever, and K. Bertels, "QX: A high-performance quantum computer simulation platform," in *Proceedings of the Conference on Design, Automation & Test in Europe (DATE)*. IEEE, 2017, pp. 464–469.

[17] X. Fu, L. Riesebos, M. A. Rol, J. van Straten, J. van Someren, N. Khammassi, I. Ashraf, R. F. L. Vermeulen, V. Newsum, K. K. L. Loh, J. C. de Sterke, W. J. Vlothuizen, R. N. Schouten, C. G. Almudever, L. DiCarlo, and K. Bertels, "eQASM: An executable quantum instruction set architecture," *arXiv:1808.02449*, 2018.

[18] R. Versluis, S. Poletto, N. Khammassi, B. Tarasinski, N. Haider, D. J. Michalak, A. Bruno, K. Bertels, and L. DiCarlo, "Scalable quantum circuit and control for a superconducting surface code," *Physical Review Applied*, vol. 8, p. 034021, 2017.

[19] C. A. Ryan, B. R. Johnson, D. Ristè, B. Donovan, and T. A. Ohki, "Hardware for dynamic quantum computing," *Review of Scientific Instruments*, vol. 88, p. 104703, 2017.

[20] Keysight, "Solution brief for Keysight Quantum Engineering Toolkit," https://literature.cdn.keysight.com/litweb/pdf/5992-3502EN.pdf?id=3023997, 2017.

[21] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, "Surface codes: Towards practical large-scale quantum computation," *Physical Review A*, vol. 86, p. 032324, 2012.

[22] Mentor Graphics, "Questa Advanced Simulator," https://www.mentor.com/products/fv/questa, 2018.

[23] P. W. Shor, "Scheme for reducing decoherence in quantum computer memory," *Physical Review A*, vol. 52, p. R2493, 1995.

[24] S. S. Tannu, Z. A. Myers, P. J. Nair, D. M. Carmean, and M. K. Qureshi, "Taming the instruction bandwidth of quantum computers via hardware-managed error correction," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-50)*. IEEE/ACM, 2017, pp. 679–691.