TU Delft

Delft University of Technology

An experimental microarchitecture for a superconducting qantum processor

Fu, X.; Rol, M. A.; Bultink, C. C.; Van Someren, J.; Khammassi, N.; Ashraf, I.; Vermeulen, R. F.L.; De Sterke, J. C.; Vlothuizen, W. J.; Schouten, R. N.

DOI
10.1145/3123939.3123952

Publication date
2017

Document Version
Final published version

Published in
MICRO 2017 - 50th Annual IEEE/ACM International Symposium on Microarchitecture Proceedings

Citation (APA)
Fu, X., Rol, M. A., Bultink, C. C., Van Someren, J., Khammassi, N., Ashraf, I., Vermeulen, R. F. L., De Sterke, J. C., Vlothuizen, W. J., Schouten, R. N., García Almudever, C., DiCarlo, L., & Bertels, K. (2017). An experimental microarchitecture for a superconducting qantum processor. In *MICRO 2017 - 50th Annual IEEE/ACM International Symposium on Microarchitecture Proceedings* (Vol. Part F131207, pp. 813-825). IEEE . https://doi.org/10.1145/3123939.3123952

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

An Experimental Microarchitecture for a Superconducting Quantum Processor

X. Fu[1,2] M. A. Rol[1,3] C. C. Bultink[1,3] J. van Someren[1,2] N. Khammassi[1,2] I. Ashraf[1,2]
R. F. L. Vermeulen[1,3] J. C. de Sterke[4,1] W. J. Vlothuizen[5,1] R. N. Schouten[1,3]
C. G. Almudever[1, 2] L. DiCarlo[1,3] K. Bertels[1,2]
[1] QuTech, Delft University of Technology, P.O. Box 5046, 2600 GA Delft, The Netherlands
[2] Computer Engineering Lab, Delft University of Technology
[3]Kavli Institute of Nanoscience, Delft University of Technology
[4] Topic Embedded Systems B.V.
[5] Netherlands Organisation for Applied Scientic Research (TNO)
{x.fu-1, m.a.rol,c.c.bultink, j.vansomeren-1, n.khammassi, i.ashraf, r.f.l.vermeulen}@tudelft.nl,
jacob.de.sterke@topic.nl, wouter.vlothuizen@tno.nl,
{r.n.schouten, c.garciaalmudever-1, l.dicarlo, k.l.m.bertels}@tudelft.nl

ABSTRACT

Quantum computers promise to solve certain problems that are intractable for classical computers, such as factoring large numbers and simulating quantum systems. To date, research in quantum computer engineering has focused primarily at opposite ends of the required system stack: devising high-level programming languages and compilers to describe and optimize quantum algorithms, and building reliable low-level quantum hardware. Relatively little attention has been given to using the compiler output to fully control the operations on experimental quantum processors. Bridging this gap, we propose and build a prototype of a flexible control microarchitecture supporting quantum-classical mixed code for a superconducting quantum processor. The microarchitecture is based on three core elements: (i) a codeword-based event control scheme, (ii) queue-based precise event timing control, and (iii) a flexible multilevel instruction decoding mechanism for control. We design a set of quantum microinstructions that allows flexible control of quantum operations with precise timing. We demonstrate the microarchitecture and microinstruction set by performing a standard gate-characterization experiment on a transmon qubit.

CCS CONCEPTS

• **General and reference → General conference proceedings**;
• **Computer systems organization → Quantum computing**; • **Hardware → Quantum technologies**;

KEYWORDS

Quantum (micro-) architecture, QuMA, quantum instruction set architecture (QISA), QuMIS, superconducting quantum processor

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
MICRO-50, October 14–18, 2017, Cambridge, MA, USA
© 2017 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-4952-9/17/10.
https://doi.org/10.1145/3123939.3123952

ACM Reference format:
X. Fu[1,2] M. A. Rol[1,3] C. C. Bultink[1,3] J. van Someren[1,2] N. Khammassi[1,2] I. Ashraf[1,2] R. F. L. Vermeulen[1,3] J. C. de Sterke[4,1] W. J. Vlothuizen[5,1] R. N. Schouten[1,3] C. G. Almudever[1, 2] L. DiCarlo[1,3] K. Bertels[1,2]. 2017. An Experimental Microarchitecture for a Superconducting Quantum Processor. In *Proceedings of MICRO-50, Cambridge, MA, USA, October 14–18, 2017,* 13 pages.
https://doi.org/10.1145/3123939.3123952

1 INTRODUCTION

To construct a fully programmable quantum computer based on the circuit model [1], a system stack [2] composed of several layers is required (Figure 1). Quantum algorithms are formulated and then described using a high-level quantum programming language [3–7]. Depending on the choice of quantum error correction code [8], such as surface code [9], the compiler [6, 10, 11] takes that description as input, performs optimization [6, 12–15] and generates a fault-tolerant implementation of the original quantum algorithm. Next, it realizes the algorithm using instructions [10, 11, 16–18] belonging to a quantum instruction set architecture (QISA). Just like in classical architectures [19], the QISA is the interface between software and hardware. A control microarchitecture is needed to decode the quantum instructions into required control signals with precise timing as well as real-time quantum error detection and correction [20, 21]. Finally, based on the specific quantum technology – e.g., superconducting qubits [22–24], trapped ions [25, 26], spin qubits [27], nitrogen-vacancy centers [28, 29], etc. – control signals are translated into required pulses, and sent to the quantum chip via the quantum-classical interface.

In current experiments, quantum processors are controlled with well-defined electrical signals, e.g., microwave-frequency and baseband pulses, which require accurate parameters and timing. To satisfy the strict requirements on control signals, dedicated electronic devices are typically used to interface with the quantum processor. However, existing control methods introduce high resource consumption, long configuration times, and control complexity, all of which scale poorly with the number of qubits [30]. Although high-level languages offer flexibility, quantum compilers typically generate instructions that are not directly executable on a quantum

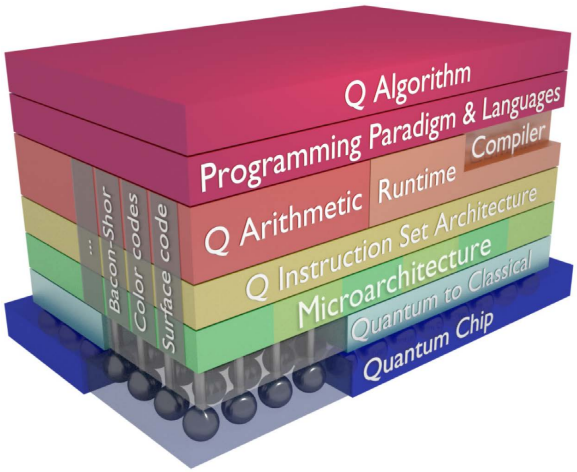


Figure 1: Overview of the quantum computer system stack from [2].

processor. It is a challenge to design a control microarchitecture that accepts a set of instructions output by a compiler and translates them into the interface required by a quantum processor.

Motivated by heterogeneous computing, we propose a control microarchitecture, named QuMA, for a superconducting quantum processor based on the circuit model. QuMA accepts quantum-classical mixed code and enables flexible and precise-timing control over a quantum processor. The four concepts at the core of QuMA are:

- Codeword-based event control scheme: every event including pulse generation and measurement is assigned with an index, which is called a codeword. These events are triggered by corresponding codewords at runtime. This scheme abstracts the control of quantum processors using complex analog pulses into a simple interface consisting of only handy binary signals, providing the foundation for flexible control via instructions.
- Queue-based event timing control: in this scheme, events with precise timing decoded from instruction execution are first buffered in a group of queues and then triggered at expected timing. It allows that events are triggered at deterministic and precise timing while the instructions are executed with non-deterministic timing.
- Multilevel instruction decoding: quantum instructions are successively translated into microinstructions, micro-operations, and finally codewords with accurate timing. It enables using technology-independent instructions to control operations on qubits.
- Quantum microinstruction set: we design and implement a low-level quantum microinstruction set (QuMIS) which enables flexible control of quantum operations.

In addition, we implement QuMA on a field-programmable gate array (FPGA). We experimentally validate QuMA by conducting a standard gate-characterization experiment on a superconducting qubit, which is called *AllXY* [31, 32]. The control, initially specified in a high-level programming language, is converted to our proposed instructions by a quantum compiler.

The paper is structured as follows. Section 2 briefly introduces the basics of quantum computing and the superconducting qubits as used in the experiment. Section 3 presents related previous work. After stating the challenges of controlling quantum processors using instructions in Section 4, Section 5 details how QuMA addresses these challenges in a systematic way with three proposed mechanisms. Section 6 discusses the advantages and scalability of QuMA. The implementation and experimental validation of QuMA and QuMIS are shown in Sections 7 and 8, respectively. Section 9 concludes.

2 BACKGROUND

2.1 Quantum Computing Basics

Quantum computing can be best viewed as computation-in-memory, in which information is stored and processed at the same place with the basic elements called qubits. A qubit can exist in a superposition of its two logical states, $|0\rangle$ and $|1\rangle$, which is mathematically described by $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where $\alpha, \beta \in \mathbb{C}$ satisfy $|\alpha|^2 + |\beta|^2 = 1$. The state of a qubit can be intuitively depicted by a vector on the Bloch sphere [1]. When measured in the logical basis, a qubit is projected onto $|0\rangle$ or $|1\rangle$ with probabilities $|\alpha|^2$ and $|\beta|^2$, respectively.

The qubit state can be modified by applying quantum gates. Every single-qubit gate is a rotation $R_{\hat{n}}(\theta)$ on the Bloch sphere along an particular axis $\hat{n}$ by an angle θ. Popular single-qubit gates include $R_x(\pi)$, $R_y(\pi)$, and $R_z(\pi)$, which are also called X, Y, and Z, respectively. There are also two-qubit gates, among which the most popular are the controlled-NOT (CNOT) and the controlled-phase (CZ). For a comprehensive introduction to quantum computing basics, we refer the interested reader to [1].

2.2 Superconducting Qubits

In this paper, we focus on transmon qubits [33] in planar circuit quantum electrodynamics [34]. This is a promising architecture for solid-state quantum computing where qubit measurement and a universal gate set [35], comprised of single-qubit gates (mainly X and Y rotations) and the CZ gate, have already achieved error rates lower than the fault-tolerance threshold for surface code [9]. Recent experiments have demonstrated basic quantum error correction for this architecture, including the repetition code [22, 23] and elements of the surface code [36].

Figure 2 shows images at various length scales of the transmon (Q) [37] that we will use in the validation. The transmon is a lumped-element nonlinear LC resonator consisting of an interdigitated capacitor in parallel with a pair of Josephson junctions providing nonlinear inductance. We use the ground state (first-excited state) of this circuit as the qubit $|0\rangle$ ($|1\rangle$) state. The transition frequency f_Q between these states can be tuned over several gigahertz on nanosecond timescales by controlling the flux through the loop between the two Josephson junctions using the proximal flux-bias line (port P_F).

Qubit measurement exploits the qubit-state dependent fundamental frequency f_R of a coplanar waveguide resonator (R) which is capacitively coupled both to the transmon and to a feedline. A pulsed measurement (typically 300 ns - 2 μs) of transmission

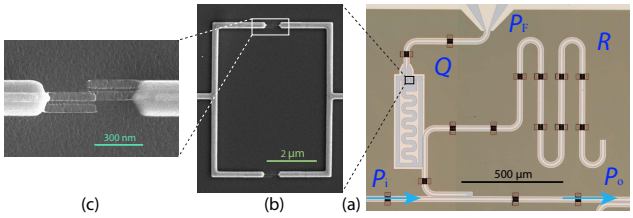


Figure 2: Images at various scales of a transmon qubit coupled to a readout resonator in a planar circuit quantum electrodynamics chip. (a) Qubit (Q), resonator (R), flux-bias line (P_F), feedline input (P_i), and feedline output (P_o). (b) Zoom-in on the two Josephson junctions of the qubit. The magnetic flux threaded through the loop sets the qubit transition frequency f_Q. (c) Zoom-in on one of the two Josephson junctions.

through the feedline (from input port P_i to output port P_o) near the fundamental of R interrogates the qubit state, projecting it to $|0\rangle$ or $|1\rangle$. Demodulation, integration, and discrimination of the transmitted signal is used to infer the measurement result.

Single-qubit gates are performed by applying calibrated microwave pulses (typically 20 ns) at f_Q to the feedline. These pulses are commonly generated by single-sideband modulation of a carrier using an I-Q mixer and envelope functions generated by an arbitrary waveform generator. The envelopes and the phase of the carrier determine the rotation axis along the equator of the Bloch sphere, and the amplitude of the pulse determines the rotation angle. Note that arbitrary single-qubit gates can be decomposed into x- and y-axis rotations albeit at the cost of longer operation sequences using some decomposition techniques, such as repeat-until-success [14].

In circuit quantum electrodynamics, the most common two-qubit gate is the CZ gate. Such a gate can be performed between qubits coupled to a common resonator or capacitor. It is realized by applying suitably calibrated pulses of typical duration $\sim$ 40 ns to the flux-bias line. We avoid going into further detail on CZ gates here as these are not part of our validation. Please see [38–40] for details.

3 RELATED WORK

Several quantum programming languages [3, 5–7, 41] and compilers [6, 10, 11] exist in which quantum algorithms can be written and compiled into a series of instructions. These quantum compilers [4, 10, 42] all generate a variant of quantum assembly language (QASM)-based instructions that belong to the quantum instruction set. Although several quantum instruction sets have been proposed, such as a von Neumann architecture-based virtual-instruction set architecture [16], quantum physical operations language (QPOL) [10], Hierarchical QASM with Loops (QASM-HL) [11], Quil [17], and OPENQASM [18], they are intermediate representations of quantum applications without considering the low-level constraints to interface with the quantum processor. They all lack an explicit control microarchitecture that implements the instructions set and allows the execution of such instructions on a real quantum processor.

Previous papers discussing quantum (micro-) architecture can be roughly divided into three groups. The first group discusses how to physically design and fabricate a quantum processor based on a specific technology, such as trapped ions [16, 26, 43, 44], superconducting qubits [45, 46], spin qubits [47], etc. The second group [15, 44, 48–51] studies how to organize qubits into multiple regions for different computational purposes to reduce the required hardware resources and communication overhead, and to maximize parallelism. The third group takes a high-level view to discuss research domains [52] and quantum abstraction [53]. All of these works use the term microarchitecture differently from this paper.

An example of control microarchitecture as viewed in this paper is [2], where emphasis is placed on the definition of technology-independent and technology-dependent functions in which the microcode unit plays an essential role. The microcode approach was first introduced by Wilkes [54] to emulate a relatively complex machine instruction as a sequence of micro-operations, called a microprogram. The microprogram can be permanently stored or cached in a control store. It enables flexible complex instruction definition using the same hardware implementation. Vassiliadis *et al.* [55] extended the microcode method to a three-level translation from machine instructions to microinstructions and finally to micro-operations. A microinstruction decoded into one (multiple) micro-operation(s) is called vertical (horizontal).

The microcode method is a computational model that also maps quite well onto quantum computing because: (1) there are frequently-used routines in quantum computing, such as error correction, which impact system performance significantly but can be well optimized via carefully tuning the microcode for these routines, as proposed by [51]; (2) most quantum algorithms frequently use more complex operations which cannot, at least in the foreseeable future, be directly implemented by a quantum processor. In this paper, we adopt the microcode approach in the proposed microarchitecture to enable flexible technology-independent instruction definition.

4 MICROARCHITECTURAL CHALLENGES

4.1 Motivational Example

We use the *AllXY* experiment [32] as an example to illustrate the microarchitectural challenges when controlling superconducting qubits. This experiment, although simple, requires flexible control over the qubit and is sensitive to control errors such as timing inaccuracy. Hence, it can reveal some of the essential features of a microarchitecture to control a superconducting quantum processor.

The *AllXY* experiment is a simple test of the calibration of single-qubit gates, which are realized by microwave pulses. Different pulse errors (amplitude, frequency, etc.) produce distinct signatures that are easily recognized. The qubit (initialized in the $|0\rangle$ state) is subjected to two back-to-back single-qubit gates and measured (Figure 3). In each round, we run 21 different gate pairs: ideally, the first 5 return the qubit to $|0\rangle$, the next 12 drive it to $\frac{1}{\sqrt{2}}\left(|0\rangle + e^{in\pi/2}|1\rangle\right)$ with $n \in \{0, 1, 2, 3\}$, and the final 4 drive it to $|1\rangle$. By averaging the measurements results for each pair over N rounds (we take $N = 25600$ in experiment), we can extract the fidelity of the qubit

to the $|1\rangle$ state, and compare to the ideal staircase signature. Algorithm 1 shows the required procedure to perform the *AllXY* experiment.

Algorithm 1: Pseudo code of the *AllXY* experiment.

```
Data: gate[21][2] = {{I, I}, {R_x(π), R_x(π)},
  {R_y(π), R_y(π)}, {R_x(π), R_y(π)}, {R_y(π), R_x(π)},
  {R_x(π/2), I}, {R_y(π/2), I}, {R_x(π/2), R_y(π/2)},
  {R_x(π/2), R_y(π/2)}, {R_x(π/2), R_y(π)},
  {R_y(π/2), R_x(π)}, { R_x(π), R_y(π/2)},
  {R_y(π), R_x(π/2)}, {R_x(π/2), R_x(π)},
  {R_x(π), R_x(π/2)}, {R_y(π/2), R_y(π)},
  {R_y(π), R_y(π/2)},{R_x(π), I}, {R_y(π), I},
  {R_x(π/2), R_x(π/2)}, {R_y(π/2), R_y(π/2)}};
for (j = 0; j < N; j++) do
    for (i = 0; i < 21; i++) do
        Init the qubit; // by waiting multiple T_1 (t_Init).
        Apply gate[i][0] on the qubit;
        Apply gate[i][1] on the qubit;
        S_{j,i} = measure(qubit);
    end
end
F_{|1⟩|meas,i} ← Σ_{j=0}^{N-1} S_{j,i}/N;
```

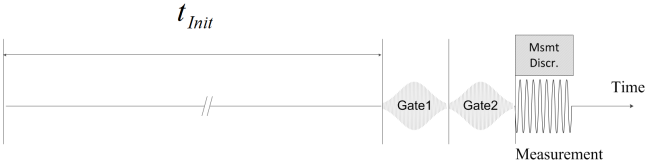


Figure 3: Waveforms and timings for one round of the *AllXY* experiment.

4.2 Complex Analog Waveform Control

In classical computers, data and control signals are both binaries. In contrast, the input and output signals of quantum processors are both complex analog signals. The measurement outcome of qubits resides in the output analog signals from the quantum processor, while quantum operations on qubits (input signals) are performed by sending analog pulses that have well-defined but variable envelope, frequency, duration, timing, etc. For example, the X gate on a transmon qubit can be implemented using a 20 ns Gaussian pulse modulated to the frequency of the qubit with a particular phase.

A popular method to produce the required pulses uses arbitrary waveform generators. Before executing quantum algorithms, the pulses are calibrated and placed in the memory of these generators as arrays of amplitude values for each sample. A pulse lasting for a time T_d requires the memory to store $N_s = 2 \cdot T_d \cdot R_s$ samples for both in-phase (I) and quadrature (Q) components, where R_s is the sampling rate, typically $\sim$ 1 GSample/s. Each sample can consist of $\sim$ 12 bits, representing the vertical resolution of the amplitude.

4.2.1 Measurement Result Discrimination. As described in Section 2.2, measurement results are contained in an analog signal $V_a(t)$. To discriminate the result for a qubit q, dedicated data-acquisition boards are commonly used to digitize $V_a(t)$ and perform integration and discrimination in software as follows:

$$S_q = \int V_a(t) W_q(t) dt, \text{ and } M_q = \begin{cases} 1 & \text{if } S_q > T_q; \\ 0 & \text{otherwise.} \end{cases}$$

Here, $W_q(t)$ and T_q are a calibrated weightfunction and threshold for q, respectively. S_q is the integration result and M_q the final binary measurement result. The software-based method is disadvantageous because of two reasons. First, the long latency of the software-based method (hundreds of microseconds) makes real-time feedback control for superconducting qubits impossible, since latency well below the typical qubit coherence time ($< 100\ \mu s$) is required. The feedback control determines the next operations based on the result of measurements and is critical in many quantum algorithms, e.g., a specific implementation [56] of Shor's factoring algorithm [57]. Second, the implied hardware resource consumption cannot scale up to a large number of qubits. A scalable measurement discrimination method with short latency constitutes a challenge.

4.2.2 Flexible Combination of Operations. Quantum algorithms and even basic quantum experiments, such as *AllXY*, require combining multiple quantum operations. To generate the required operation combinations, current arbitrary waveform generators first upload long waveforms combining different pulses with appropriate timing and later play them. A drawback of this method is that even a small change to the operations requires a new upload of the entire waveform which costs significant memory and upload time. To generate the 21 combinations in the *AllXY* experiment, 21 different waveforms must be uploaded. With more qubits and more complex algorithms, the combination of operations can be more, which asks for more waveforms, leading to more memory consumption and larger uploading latency. Therefore, this method does not easily scale to a large number of qubits.

Furthermore, the execution of quantum programs requires more flexible feedback control, which cannot be supported by the autonomous arbitrary waveform generators as these devices cannot change a waveform to incorporate dynamically determined operations. Therefore, it is a requirement to define a flexible and scalable way to combine multiple smaller pulses, such that any sequence can be easily programmed, changed and executed when necessary.

4.2.3 Accurate Timing Control. Instructions in classical processors are usually executed with non-deterministic timing on a nanosecond timescale due to (1) process switching and system calls in the software layer, (2) indefinite communication latency including memory access, (3) static and dynamical instruction reorder, (4) pipeline stall and flushing, etc. However, the non-deterministic timing typically does not matter and the program can run correctly as long as the relative order of inter-dependent instructions is preserved.

In contrast, precise timing on nanosecond timescales is critical to quantum operations. As discussed in Section 2.2, when a fixed single-sideband modulation is used, the timing of pulses must be accurate to maintain the carrier phase, which sets the rotation axis of single-qubit gates. For example, given a fixed 50 MHz

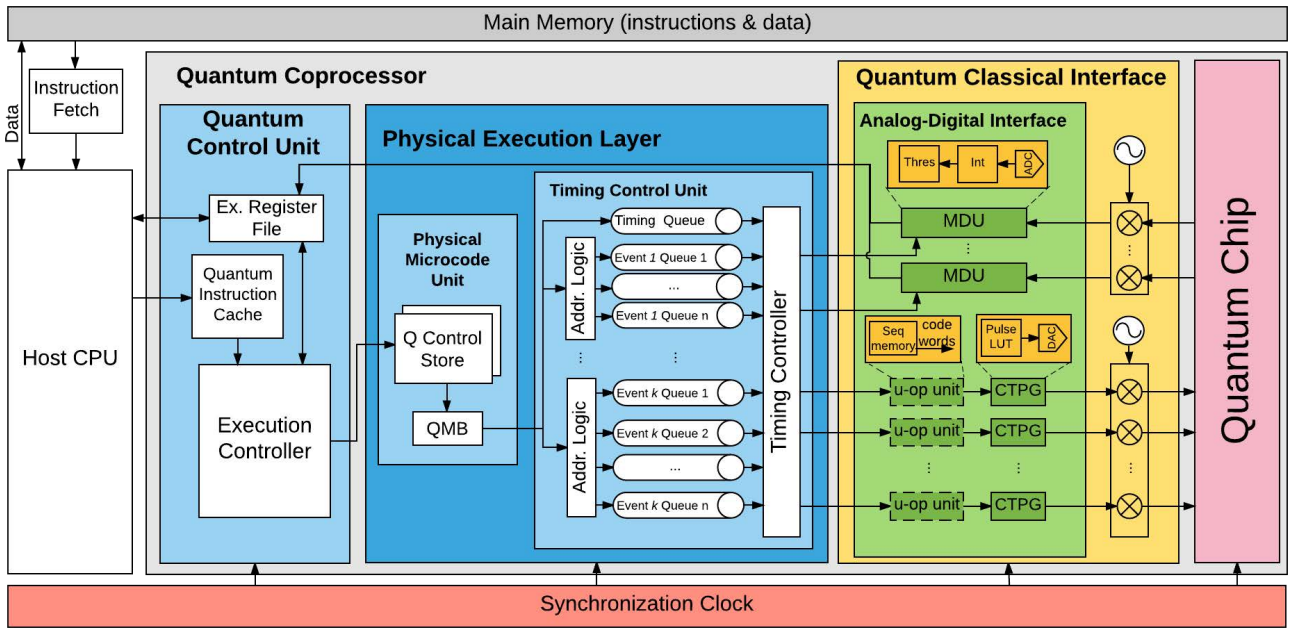


Figure 4: Overview of the Quantum MicroArchitecture (QuMA).

single-sideband modulation in the *AllXY* experiment, applying the modulation envelope of an x rotation 5 ns later will produce a y rotation instead. Besides, some quantum experiments require operations to be applied at a particular point in time. For example, the pulses implementing the two single-qubit gates and the measurement must be applied on the qubit back-to-back. To provide the appropriate timing precision, dedicated hardware is needed where again scalability in terms of the number of qubits is an additional requirement.

Using instructions to specify the timing of operations is more promising. However, it is challenging to use non-deterministic instruction execution to generate pulses with deterministic and precise timing.

4.3 Instruction Definition

The instruction set architecture is the interface between hardware and software and is essential in a fully programmable classical computer. So is QISA in a programmable quantum computer.

As explained in Section 3, existing instruction set architecture definitions for quantum computing mostly focus on the usage of the description and optimization of quantum applications without considering the low-level constraints of the interface to the quantum processor. It is challenging to design an instruction set that suffices to represent the semantics of quantum applications and to incorporate the quantum execution requirements, e.g., timing constraints.

It is a prevailing idea that quantum compilers generate technology-dependent instructions [4, 10, 42]. However, not all technology-dependent information can be determined at compile time because some information can only be generated at runtime due to hardware limitations. An example is the presence of defects on a quantum processor affecting the layout of qubits used in the algorithm. In addition, the following observations hold: (1) quantum technology is rapidly evolving, and more optimized ways of implementing the quantum gates are continuously explored and proposed; a way to easily introduce those changes, without impacting the rest of the architecture, is important. (2) depending on the qubit technology, the kind, number and sequence of the pulses can vary. Hence, it forms another challenge to microarchitecturally support a set of quantum instructions which is as independent as possible of a particular technology and its current state of the art.

5 QUANTUM MICROARCHITECTURE

In this section, we describe the Quantum MicroArchitecture (QuMA) as shown in Figure 4. QuMA is a heterogeneous architecture which includes a classical CPU as a host and a quantum coprocessor as an accelerator.

As proposed in [2], the input of QuMA is a binary file generated by a compiler infrastructure where classical code and quantum code are combined. The classical code is produced by a conventional compiler such as GCC and executed by the classical host CPU. Quantum code is generated by a quantum compiler and executed by the quantum coprocessor.

As shown in Figure 4, the host CPU fetches quantum code from the memory and forwards it to the quantum coprocessor. In the quantum coprocessor, executed instructions in general flow through modules from left to right. The execution controller performs register update, program flow control and streams quantum instructions to the physical execution layer. The physical microcode unit translates quantum instructions into microinstructions using the Q

control store. These are further decomposed into micro-operations by the quantum microinstruction buffer (QMB). The timing of each micro-operation is also determined by the physical microcode unit. Based on the output of quantum microinstruction buffer, the timing control unit triggers micro-operations at a deterministic timing. The analog-digital interface converts digitally represented micro-operations into corresponding analog pulses with precise timing that perform quantum operations on qubits, as well as analog signals containing measurement information of qubits into binary signals. Required modulation and demodulation with radio-frequency carrier waves are also carried out in the quantum-classical interface.

In order to address the challenges described in the previous section, three schemes are introduced in QuMA. (i) The codeword-based event control scheme is implemented by the codeword-triggered pulse generation unit (CTPG), which produces analog input to the quantum processor based on the received codeword triggers, and the measurement discrimination unit (MDU) converting the analog output from the quantum processor into binary results. (ii) The queue-based event timing control scheme is implemented by the timing control unit, which issues event triggers with precise timing to the measurement discrimination unit and the micro-operation unit (u-op unit). (iii) A multilevel instruction decoding scheme, which successively decodes a quantum instruction into microinstructions at the Q Control Store, micro-operations at the quantum microinstruction buffer, and finally codeword triggers at the micro-operation unit. The complex analog waveform control challenge is addressed by (i) and (ii) whereas the instruction definition is addressed by (iii).

5.1 Codeword-Based Event Control

The analog-digital interface (Figure 4) is at the boundary of analog signals and digital signals in QuMA, which is technology-dependent. As shown in Figure 4, from left to right , the micro-operation unit and the codeword-triggered pulse generation unit translate codeword triggers into pulses representing quantum operations on the qubits with a fixed latency. From right to left, analog measurement waveforms from the quantum processor are discriminated into binary results by the measurement discrimination unit. In this way, the analog-digital interface abstracts the complex analog waveform generation and puts forward the responsibility of codeword control with precise timing to the upper digital layers. Therefore, it enables controlling analog pulse generation using instructions. Fast and flexible feedback control is also possible in principle because the codeword-triggered pulse generation scheme does not require the waveform to be uploaded at runtime and codeword triggers with precise timing can be efficiently generated dynamically.

5.1.1 Codeword-Triggered Pulse Generation. From experiments, we observe that the pulses for a fixed and small set of quantum operations can be well defined and used after calibration. They are also called primitive operations because they are sufficient for many quantum computing experiments. Based on this, we introduce the codeword-triggered pulse generation scheme in QuMA to generate pulses corresponding to primitive operations. In codeword-triggered pulse generation, well-defined primitive pulses instead of entire waveforms are uploaded to the memory. The memory is organized as a lookup table and each entry in the lookup table, indexed by means of a codeword, contains the sample amplitudes corresponding to a single pulse. The codeword-triggered pulse generation unit converts a digitally stored pulse into an analog one only when it receives a codeword trigger. An example of the lookup table content for single-qubit operations is shown in Table 1.

Table 1: An example of the lookup table content of a codeword-triggered pulse generation unit for single-qubit gates.

Codeword	0	1	2	3
Pulse	I	$R_x(\pi)$	$R_x(\frac{\pi}{2})$	$R_x(-\frac{\pi}{2})$
Codeword	4	5	6	$\cdots$
Pulse	$R_y(\pi)$	$R_y(\frac{\pi}{2})$	$R_y(-\frac{\pi}{2})$	$\cdots$

The codeword-triggered pulse generation scheme has a modest memory requirement since it only needs to store a small number of pulses for the well-defined primitive operations. In the *AllXY* experiment, only the pulses for 7 operations need to be stored, which only consumes the memory for $7 \times 2 \times 20$ ns $\times R_s$ samples (in total 420 Bytes), instead of 21 waveforms each containing two operations, that are $21 \times 2 \times 2 \times 20$ ns$\times R_s$ samples (in total 2520 Bytes). When more complex combination of operations is required, the memory consumption will remain the same and the memory saving will be more significant. The small memory footprint provides a scalable path for controlling a larger number of qubits.

The delay between the codeword trigger and the pulse generation is required to be fixed and short in the codeword-triggered pulse generation unit. The fixed delay ensures that the flexible combination of the pulses with precise timing can be achieved by flexibly generating the corresponding codeword triggers at precise timing. In the *AllXY* experiment, by issuing the codeword triggers for the two gates with an interval of 20 ns, the pulses for the two gates can be played out exactly back to back.

5.1.2 Measurement Discrimination. Recent experiments have demonstrated measurement discrimination using a customized FPGA [37], achieving a short latency $< 1\ \mu$s which enables real-time feedback control. This method also costs modest hardware exhibiting better scalability. Adopting this idea, we introduce hardware-based measurement discrimination units in the analog-digital interface. The measurement discrimination unit translates the analog signal containing measurement information of a single qubit into a binary measurement result. Once the measurement discrimination unit for qubit q receives a codeword trigger, it starts the measurement discrimination process and generates a binary result R_q. R_q can be subsequently forwarded to the quantum control unit for feedback control or reading back.

Recent experiments have also demonstrated combining the measurement result of multiple qubits into one analog signal [23, 58]. This can reduce the number of required measurement discrimination units and exhibits better scalability.

5.2 Queue-Based Event Timing Control

The timing control unit divides the microarchitecture into two timing domains: the non-deterministic timing domain and the deterministic timing domain, which are on the left and right side

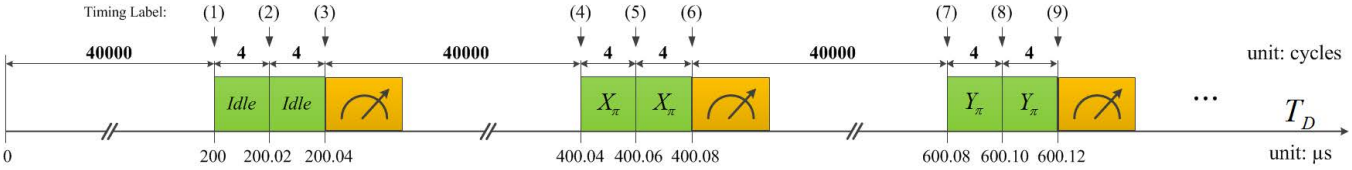


Figure 5: Operations of the *AllXY* experiment in the timeline. Measurement pulse generation and measurement result discrimination overlap in time and are shown using the same meter box.

of the timing control unit in Figure 4, respectively. In the non-deterministic timing domain, the quantum control unit and physical execution layer execute instructions and feed quantum operations to the queues in an as-fast-as-possible fashion. In the deterministic timing domain, quantum operations in the queue are emitted to the analog-digital interface with deterministic and precise timing. To this end, queue-based event timing control is introduced.

To illustrate the working principle of queue-based event timing control, the operations of the *AllXY* experiment with corresponding timing are shown in Figure 5. The horizontal axis labels mark the time points in microseconds when a corresponding operation takes place. Each time point is assigned a timing label, which is the number in brackets on the top. The bold numbers above the double-arrow lines indicate intervals between two time points in cycles. Here and throughout the rest of the paper, a cycle time of 5 ns is used.

The timing control unit implements queue-based event timing control in QuMA. It consists of a timing queue, multiple event queues, and a timing controller. The timing queue buffers the time points with corresponding timing labels. The location of the time points can be designated in the timeline, e.g., by specifying the intervals between consecutive time points as shown in Figure 5 and the first column of Table 2. Each event queue buffers a sequence of events with a time point at which the event is expected to take place. The time point is indicated by the aforementioned timing label. An event can be a quantum gate, measurement, or any other operation. The timing controller maintains the clock of the deterministic timing domain (T_D), which can be started by an instruction or another source, e.g., an external trigger. When T_D reaches the assigned time point, the timing controller signals the queues to fire the events matching that time point and emits them to the analog-digital interface.

In order to better illustrate how queue-based event timing control works, we use the *AllXY* experiment. Three event queues are used in this experiment (see Table [2-4]): the Pulse Queue for single-qubit operations, the MPG Queue for measurement pulse generation, and the MD Queue for measurement discrimination. Besides the timing label for each event, the pulse queue contains the single-qubit operations, e.g., the I or X_π operation, to be triggered, and the MD queue contains the destination register, e.g., $r7$, to write back the measurement result. After executing a couple of instructions in the program and before T_D is started, the state of the queues is as shown in Table 2. The bottom of the table corresponds to the front of the queues. After T_D is started, a counter in the timing controller starts counting. When the counter reaches the first interval value in the timing queue, i.e., 40000, the corresponding timing label, i.e., 1, is broadcast to all event queues. At the same time, the counter resets and restarts. Since the pulse queue contains that same label, 1, at the front of the queue, the operation I is fired to the analog-digital interface. The queue state then turns into Table 3. The second I operation is issued in the same way when the counter reaches the next interval value, 4. After the counter reaches the third interval value, 4, the timing label 3 is broadcast and the MG Queue triggers the measurement pulse generation and the MD queue triggers a measurement discrimination process of which both associated timing labels are 3. The queue state then turns into Table 4. The rest can be done in the same manner.

Table 2: Queue state of the *AllXY* experiment when $T_D = 0$.

Timing Queue	Pulse Queue	MPG Queue	MD Queue
⋮	⋮	⋮	⋮
(4, 6)			
(4, 5)			
(40000, 4)	(X_π, 5)		
(4 , 3)	(X_π, 4)		
(4, 2)	(I, 2)	(6)	($r7$, 6)
(40000, 1)	(I, 1)	(3)	($r7$, 3)

Table 3: Queue state of the *AllXY* experiment when T_D = 40000.

Timing Queue	Pulse Queue	MPG Queue	MD Queue
⋮	⋮	⋮	⋮
(4, 6)			
(4, 5)			
(40000, 4)	(X_π, 5)		
(4, 3)	(X_π, 4)	(6)	($r7$, 6)
(4, 2)	(I, 2)	(3)	($r7$, 3)

Table 4: Queue state of the *AllXY* experiment when T_D = 40008.

Timing Queue	Pulse Queue	MPG Queue	MD Queue
⋮	⋮	⋮	⋮
(4, 6)			
(4, 5)	(X_π, 5)		
(40000, 4)	(X_π, 4)	(6)	($r7$, 6)

5.3 Multilevel Instruction Decoding

Combining the codeword-based event control scheme and queue-based event timing control enables other stages in QuMA to focus on flexibly decoding the quantum instructions and filling the queues as fast as possible without worrying about complex analog waveform control with rigid timing constraints. In this subsection, we first give an overview of the instruction definition and then discuss the multilevel decoding scheme for the quantum instructions.

5.3.1 Instruction Definition. The quantum code is written with instructions in the Quantum Instruction Set (QIS). An example of QIS instructions is shown in Table 5. QIS contains auxiliary classical instructions and quantum instructions. Auxiliary classical instructions are used for basic arithmetic and logic operations and program flow control. Quantum instructions describe which and when quantum operations will be applied on qubits. By including auxiliary classical instructions, QIS can support feedback control based on measurement results and a hierarchical description of quantum algorithms which can significantly reduce the program code size [13].

5.3.2 Instruction Decoding. To support a technology-independent quantum instruction set definition, we adopt a multilevel instruction decoding approach in which quantum instructions, especially that for quantum gates, are successively decoded into quantum microinstructions, micro-operations and finally codeword triggers to control codeword-triggered pulse generation to generate pulses. For example, Table 5 shows four decoding steps for the instructions of the *AllXY* experiment. From the QIS on, time is calculated in cycles. Due to the simplicity of the *AllXY* experiment and for the sake of code efficiency, the inner loop as shown in Algorithm 1 is unrolled. The execution of quantum instructions starts from the execution controller.

Execution Controller. This unit executes the auxiliary classical instructions in the QIS and streams quantum instructions to the physical microcode unit. By executing the auxiliary classical instructions in the execution controller, the same quantum instruction can be issued to the physical microcode unit multiple times and each time with expected parameters computed at runtime. For example, the *QNopReg r15* instruction in the QIS is used to specify the initialization time. Each of the 21 *QNopReg r15* instructions will be issued once per round. Every time it is issued, it reads a waiting time from the register r15, which results in a *Wait 40000* instruction. If the register value is updated using auxiliary classical instructions, the waiting time specified in the *Wait* instruction can be calculated at runtime. In this way, it enables a compact and flexible description of quantum algorithms.

Physical Microcode Unit. Quantum instructions are translated into a sequence of microinstructions in the physical microcode unit based on the microprograms uploaded into the Q control store. The timing for each quantum operation is also determined at this stage. For now and as shown in Table 6, the microinstruction set, QuMIS, consists of the following instructions: i) the **Wait** instruction used to specify the interval between consecutive time points, ii) the **Pulse** instruction used to apply quantum gates on qubits; iii) the **MPG** instruction used to generate the measurement pulse; iv) the **MD** instruction used to trigger the measurement discrimination process.

Table 5: The format of QIS instructions, quantum microinstructions, micro-operations and codeword triggers. Taking the *AllXY* experiment as an example.

QIS	QuMIS
# Input to the execution controller	# Input to the QMB
mov r1, 0	# round 0:
mov r2, 25600	Wait 40000
mov r3, ResultMemAddr	Pulse {q0}, I
mov r15, 40000	Wait 4
Outer_Loop:	Pulse {q0}, I
QNopReg r15	Wait 4
Apply I, q0	MPG {q0}, 300
Apply I, q0	MD {q0}, r7
Measure q0, r7	# round 1:
Load r9, r3[0]	Wait 40000
Add r9, r9, r7	Pulse {q0}, X180
Store r9, r3[0]	Wait 4
	Pulse {q0}, X180
QNopReg r15	Wait 4
Apply X180, q0	MPG {q0}, 300
Apply X180, q0	MD {q0}, r7
Measure q0, r7	. . .
Load r9, r3[1]	
Add r9, r9, r7	
Store r9, r3[1]	
...	
add r1, r1, 1	
bne r1, r2, Outer_Loop	

Micro-operations	Codeword Triggers
# Input to the u-op units	# Input to the MDU or CPTG
	# Δ is the delay of the u-op unit
T_D = 40000:	T_D = 40000 + Δ:
I sent to u-op unit0	CW 0 sent to CTPG0
T_D = 40004:	T_D = 40004 + Δ:
I sent to u-op unit0	CW 0 sent to CTPG0
T_D = 40008:	T_D = 40008:
# MPG and MD bypass this stage	CW 7 sent to CTPG5 # Msmt
T_D = 80008:	*MD(r7)* sent to MDU0
X_π sent to u-op unit0	T_D = 80008 + Δ:
T_D = 80012:	CW 1 sent to CTPG0
X_π sent to u-op unit0	T_D = 80012 + Δ:
T_D = 80016:	CW 1 sent to CTPG0
# MPG and MD bypass this stage	T_D = 80016:
. . .	CW 7 sent to CTPG5 # Msmt
	MD(r7) sent to MDU0
	. . .

In the quantum microinstruction buffer (QMB), quantum microinstructions for quantum gates are decomposed into separate micro-operations with timing labels and push them into the queues in the timing control unit as shown in Table 2. Due to the simplicity of measurements in terms of instruction control, quantum microinstructions for measurement pulse generation or measurement discrimination can be directly translated into codeword triggers to control the codeword-triggered pulse generation unit or the measurement discrimination unit bypassing the micro-operation unit. The timing control unit then emits the micro-operations at

Table 6: QuMIS instructions.

Assembly Format	Description
Wait *Interval*	Wait for the number of cycles indicated by the immediate value *Interval*.
Pulse ($QAddr_0$, uOp_0)[, ($QAddr_1$, uOp_1), ...]	Apply the micro-operation uOp_i on each of the qubit(s) specified by the address $QAddr_i$.
MPG *QAddr*, *D*	Generate the measurement pulse for the qubits specified by the address *QAddr*. *D* indicates the duration of the measurement pulse in number of cycles.
MD *QAddr*, *$rd*	Discriminate the measurement results of the qubits specified by *QAddr* and store the result into register *$rd*.

the expected timing. The **Pulse** and **MPG** instructions are both horizontal instructions, which can trigger the operation on multiple qubits at the same time.

Let us illustrate these concepts using the CNOT gate. A CNOT gate with a control qubit c and a target qubit t can be decomposed in the following way [1]:

$$\mathrm{CNOT}_{c,t} = R_y(\pi/2)_t \cdot CZ \cdot R_y(-\pi/2)_t .$$

Adopting the microcoded approach for the instruction *CNOT qt, qc* applying on superconducting qubits results in Algorithm 2.

Algorithm 2: Microprogram for the physical *CNOT q1, q2*.

```
1  Pulse   {qt},      Ym90
2  Wait     4
3  Pulse   {qt, qc}, CZ
4  Wait     8
5  Pulse   {qt},      Y90
6  Wait     4
```

By utilizing horizontal microcode, one quantum instruction can be translated into multiple microinstructions and one microinstruction into multiple micro-operations. This allows flexible emulation of complex, technology-independent instructions using technology-dependent primitives.

Micro-Operation Unit. At the micro-operation unit, each micro-operation is translated into a sequence of codeword triggers with predefined latency, which further makes associated codeword-triggered pulse generation units generate primitive operation pulses. For each predefined micro-operation uOp_i, the micro-operation unit stores a sequence Seq_i comprising of codewords and timing. Seq_i has the following format:

$$\mathrm{Seq}_i : \ ([0,\ cw_0];\ [\Delta t_1,\ cw_1];\ [\Delta t_2,\ cw_2];\ \ldots),$$

where Δt_j represents the interval between codeword triggers cw_{j-1} and cw_j. Once the micro-operation uOp_i is triggered, the micro-operation unit starts to output codeword cw_j after waiting for Δt_j cycles sequentially as defined in the sequence Seq_i. Since the timing controller fires the micro-operation at precise timing, the codeword triggers are also generated at precise timing.

For example, a Z gate can be decomposed into a Y gate followed by an X gate since $Z = X \cdot Y$ (up to an irrelevant global phase). The micro-operation unit can perform the translation for superconducting qubits using the following sequence given the lookup table content as listed in Table 1:

$$\mathrm{Seq}_Z : \ ([0,\ 1];\ [4,\ 4]).$$

The micro-operation unit allows the emulation of commonly-used quantum operations which are not directly implementable using primitive operations. Moreover, it reduces the communication between the timing control unit and the analog-digital interface. This is especially helpful when the timing control unit and the analog-digital interface are implemented in different electronic devices for performance and scalability.

6 EVALUATION

To evaluate QuMA, we make a comparison between QuMA and the architecture of the Raytheon BBN APS2 system, which is a commercial device that has been recently demonstrated [58, 59] for superconducting qubits. Then we discuss the scalability limitation of QuMA.

The APS2 system has a distributed architecture consisting of nine individual APS2 modules and a trigger distribution module (TDM) that can fully control up to eight qubits. A quantum application is translated into multiple binary executables running in parallel on each of the APS2 modules. A binary is composed of separated program flow control instructions and output instructions. Instead of instructions with explicit quantum semantics, low-level output instructions are used, such as waveform with a physical memory address. Idle waveforms are used to implement precise timing between operations, and the TDM distributes trigger signals to perform parallelism/synchronization of multiple outputs via an interconnect network. The main disadvantage are that no output instructions can be processed when synchronization is required, and the interconnect network is cumbersome and fragile when scaling up to tens of qubits where multiple APS2 systems are required [58].

In contrast, QuMA employs a centralized architecture, in which: (i) only one binary executable is required for controlling multiple qubits, (ii) quantum semantics and timing of operations are explicitly defined at the instruction level, (iii) parallelism/synchronization of outputs is achieved by triggering events at specific timing points, which is neither dependent on another module nor limited by the interconnect network. These three points contribute to a relatively simple compilation model for QuMA. As explained in Section 5.2, QuMA decouples the timing of executing instructions and performing output. So it can maintain fully deterministic timing of the output and maximally process instructions during waiting. Since data is gathered in a single place (the register file), it is natural to extend QuMA to a heterogeneous computing platform by adding extra data exchange instructions to interact with the host CPU and the main memory.

Regarding scalability, QuMA is not limited by the analog-digital interface and the timing control unit, as their size scales linearly to the number of qubits and can be implemented in a distributed way. However, the limited time for executing instructions in quantum computers may form a challenge in QuMA when more qubits ask for a higher operation output rate while only a single instruction stream is used. A Very-Long-Instruction-Word (VLIW) architecture [19] can be adopted to provide much larger instruction issue rate. In

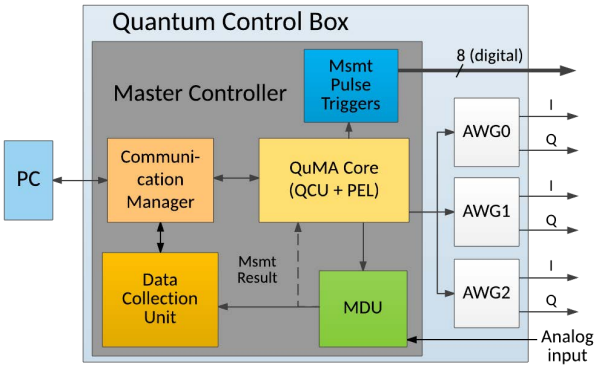


Figure 6: Schematic of the CBox firmware architecture. The QuMA core is implemented in the Master Controller. Dashed lines indicate functionality to be added in the future.

addition, by optimizing the microcode unit and the micro-operation unit, it is possible to use less quantum instructions to describe more quantum operations, which can relax the instruction issue rate requirement.

7 IMPLEMENTATION

In this section, we discuss the quantum control box, where the aforementioned mechanisms have been implemented.

7.1 Quantum Control Box

The quantum control box, as shown schematically in Figure 6, consists of four FPGA boards. One board implements the Master Controller and the other three boards implement a two-channel arbitrary waveform generator (AWG) each.

The master controller is implemented using an Arrow BeMicro CV A9 board holding an Altera Cyclone V 5CEFA9 FPGA chip. It connects to two 8-bit resolution analog-to-digital converters (ADC) that digitize analog measurement signals from the quantum chip. The master controller has eight digital outputs used for triggering measurement pulse generation and triggers the pulse generation of each AWG via a pair of Low-Voltage-Differential-Signaling wires.

Inside the MC, the QuMA core implements the quantum control unit and the physical execution layer of QuMA. The digital output unit converts the measurement operation tuple $(QAddr, D)$ received from the QuMA core into '1' state with a duration of D cycles for the eight digital outputs masked by $QAddr$. The measurement discrimination unit (MDU) can discriminate the measurement result of a single qubit. The data collection unit can collect K consecutive integration results of a single qubit for N rounds, calculate and store the average of K integration results across the N rounds:

$$\bar{S_i} = \left(\sum_{j=0}^{N-1} S_{i,j} \right) / N \ , \quad i \in \{0, 1, \cdots, K-1\}.$$

After the data collection process is done, the PC can retrieve the averaging integration results $\{\bar{S_i}\}$.

Each AWG is implemented using a Terasic DE0-Nano board holding an Altera Cyclone IV EP4CE22F FPGA chip and uses two 14-bit resolution digital-to-analog converters (DAC) to generate the in-phase and quadrature components of qubit control pulses. Each AWG includes a micro-operation unit and a codeword-triggered pulse generation unit. The implemented codeword-triggered pulse generation unit has a fixed delay of 80 ns from the codeword trigger to the output pulse.

All FPGAs, ADCs, and DACs are clocked at 200 MHz, except for communication and data collection, which run at 50 MHz. The MC communicates with the PC via USB. The MC communicates to the AWGs, e.g., uploading the lookup table content of the codeword-triggered pulse generation unit.

7.2 QuMA Implementation

The QuMA implementation in the control box in shown in Figure 7. In view of the running physics experiments, it slightly differs from the microarchitecture presented in Section 5. We have partially implemented the system including the quantum instruction cache, the execution controller, part of the physical microcode unit, the timing control unit and the quantum classical interface. The rest is planned for future release. Due to the absence of a fully functioning physical microcode unit, the high-level quantum instructions of the QIS are not implemented yet. A combination of the auxiliary classical instructions in the QIS and QuMIS (see Table 6) is loaded into the quantum instruction cache.

We have designed a quantum programming language OpenQL based on C++ with a compiler that can translate the OpenQL description into the auxiliary classical instructions and QuMIS instructions.

The execution controller incorporates a classical pipeline to execute auxiliary classical instructions. The register file in this pipeline contains runtime information related to quantum program execution. QuMIS instructions are dispatched to the physical microcode unit after reading register values. The physical microcode unit can determine the timing of QuMIS instructions and decompose QuMIS instructions into micro-operations. A full implementation of the physical microcode unit is still under development. The timing control unit implements the queue-based event timing control scheme (as described in Section 5.2). The measurement pulse triggers pulse modulated microwave carrier generators in the other devices block to produce the measurement pulse for qubits.

8 EXPERIMENTAL RESULTS

We have performed various quantum experiments on a qubit to validate and verify the design of QuMA and QuMIS, including T_1, T_2 Ramsey, T_2 Echo, *AllXY*, and randomized benchmarking [60] experiments. Considering the readability and page limitation, we only show the *AllXY* experiment in the paper.

Figure 8 shows the experimental setup. All classical electronics are at room temperature. The quantum chip, operating at 20 mK, contains 10 transmon qubits with dedicated readout resonators all coupled to a common feedline. The measured qubit (labeled 2) has transition frequency $f_Q = 6.466$ GHz, and the coupled resonator has fundamental $f_R = 6.850$ GHz (for qubit in $|0\rangle$) (further detailed in [37]). To perform single-qubit gates, we use one microwave

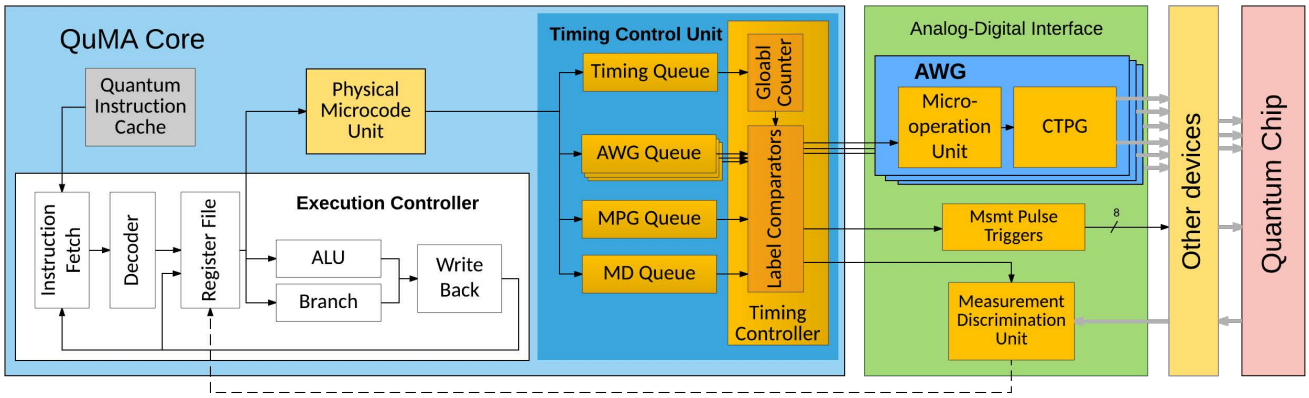


Figure 7: Schematic of the implemented QuMA. The thick gray lines are analog signals while the dark thin lines are digital signals. Dashed lines indicate functionality to be added in the future.

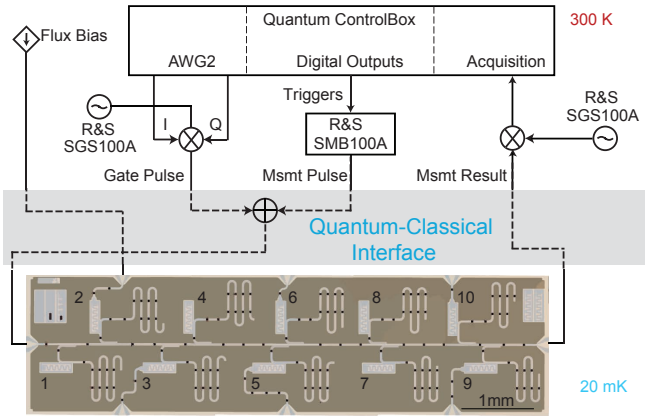


Figure 8: Experimental setup used for validation of the microarchitecture.

source [Rohde & Schwarz (R&S) SGS100A] to generate a 6.516 GHz carrier and control box AWG 2 to produce the in-phase and quadrature components (including −50 MHz single-sideband modulation) that define the pulse envelope. To generate the measurement pulse, we trigger a 6.849 GHz carrier (generated by a R&S SMB100A) using the control box digital output 1. The transmitted feedline signal is demodulated to an intermediate frequency of 40 MHz using a 6.809 GHz local oscillator (another R&S SGS100A). Prior to the experiment, the qubit pulses are calibrated and uploaded into control box AWG 2. Since the operations in the *AllXY* experiment are primitive, the micro-operation unit simply forwards the codewords to the wave memory without translation.

The QuMIS program used to perform the *AllXY* experiment is generated from a OpenQL description and is shown in Algorithm 3. In this experiment, each of the 21 combinations is measured twice to make a direct visual distinction between systematic errors and low signal-to-noise ratio. Figure 9 shows the measurement results. The red staircase shows the ideal signature of perfect pulsing. The

Algorithm 3: QuMIS Program to perform *AllXY* experiment.

```
1  mov   r15,   40000      # 200 us
2  mov   r1,    0          # loop counter
3  mov   r2,    25600      # number of averages
4
5  Outer_Loop:
6    QNopReg r15          # Identity, Identity
7    Pulse   {q2}, I
8    Wait    4
9    Pulse   {q2}, I
10   Wait    4
11   MPG     {q2}, 300
12   MD      {q2}
13 (repeat the previous 7 instructions once again)
14
15   QNopReg r15          # X180, X180
16   Pulse   {q2}, X180
17   Wait    4
18   Pulse   {q2}, X180
19   Wait    4
20   MPG     {q2}, 300
21   MD      {q2}
22 (repeat the previous 7 instructions once again)
23
24   QNopReg r15          # Y180, Y180
25   Pulse   {q2}, Y180
26   Wait    4
27   Pulse   {q2}, Y180
28   Wait    4
29   MPG     {q2}, 300
30   MD      {q2}
31 (repeat the previous 7 instructions once again)
32
33   ...
34
35   addi    r1, r1, 1
36   bne     r1, r2, Outer_Loop
```

results of the 0-th (18-th and 19-th) combination are taken as the calibration point $\bar{S}_{|0\rangle,r}$ ($\bar{S}_{|1\rangle,r}$). Using the calibration points to rescale

the signal, we obtain the fidelity $F_{|1\rangle}|_i$ corrected for readout error:

$$F_{|1\rangle}|_{\text{meas},i} = \left(\bar{S}_i - \bar{S}_{|0\rangle,r}\right) / \left(\bar{S}_{|1\rangle,r} - \bar{S}_{|0\rangle,r}\right).$$

We loop over these $K = 42$ pulse combinations over $N = 25600$ rounds. The data acquisition unit performs the required averaging of measurement results for each K.

This experiment uses the instructions generated from the high-level language OpenQL description to control the operations on the qubit. Only 7 pulses including the *Identity* operation are stored in the lookup table of the codeword-triggered pulse generation unit, regardless of the number of combinations of operations. It has a moderate memory consumption to store 140 ns $\times R_s$ samples exhibiting a better scalability compared to the conventional method. From the experiment result, we can see that the measured fidelity for each combination matches well with the ideal readout fidelity. Since the *AllXY* experiment is sensitive to imperfection of the pulses and the timing, it demonstrates that the right pulses are generated and the precise timing of operations is well preserved.

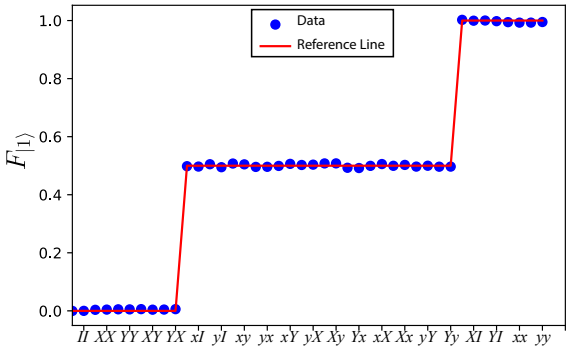


Figure 9: The *AllXY* result of qubit 2. In the label, each X/Y (x/y) denotes a rotation by π ($\pi/2$) around the x/y axis of the Bloch sphere.

9 CONCLUSION

We have proposed and developed QuMA, a microarchitecture that takes the compiler generated instructions as input to flexibly control a superconducting quantum processor. Three mechanisms are introduced in QuMA to enable flexible control over quantum processors : i) codeword-based event control, ii) precise queue-based event timing control, and iii) multilevel instruction decoding pulse control mechanism. We have also designed and implemented the quantum microinstructions set QuMIS which can well describe quantum operations on qubits with precise timing.

We implemented a QuMA processor prototype on a FPGA. We have validated this microarchitecture by performing a successful *AllXY* experiment on a superconducting qubit, using a combination of the auxiliary classical instructions and QuMIS instructions which are generated by OpenQL. QuMA enables flexible definition of quantum experiments by a straightforward change in the input program.

Future work will involve implementing a QuMA supporting a VLIW instruction set, and extending the microcode unit to enable the definition of quantum instructions and the execution of real-time feedback control.

ACKNOWLEDGMENTS

We thank M. Tiggelman, S. Visser, J. Somers, L. Riesebos, E. Garrido Barrabés, and E. Charbon for contributions to an early version of the CBox, A. Bruno for fabricating the quantum chip, H. Homulle for drawing Figure 1, and L. Lao, H. A. Du Nguyen, R. Versluis and F. T. Chong for discussions. We acknowledge funding from the China Scholarship Council (X. Fu), Intel Corporation, an ERC Synergy Grant, and the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via the U.S. Army Research Office grant W911NF-16-1-0071. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

REFERENCES

[1] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge University Press, 2010.
[2] X. Fu, L. Riesebos, L. Lao, C. Almudever, F. Sebastiano, R. Versluis, E. Charbon, and K. Bertels, "A heterogeneous quantum computer architecture," in *Proceedings of the ACM International Conference on Computing Frontiers*. ACM, 2016, pp. 323–330.
[3] B. Omer, "Structured quantum programming," *Information Systems*, p. 130, 2003.
[4] A. J. Abhari, A. Faruque, M. J. Dousti, L. Svec, O. Catu, A. Chakrabati, C.-F. Chiang, S. Vanderwilt, J. Black, and F. Chong, "Scaffold: Quantum programming language," DTIC Document, Tech. Rep., 2012.
[5] A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron, "An introduction to quantum programming in quipper," in *International Conference on Reversible Computation*. Springer, 2013, pp. 110–124.
[6] D. Wecker and K. M. Svore, "LIQUi|>: A software design architecture and domain-specific language for quantum computing," *arXiv:1402.4467*, 2014.
[7] D. S. Steiger, T. Häner, and M. Troyer, "ProjectQ: an open source software framework for quantum computing," *arXiv:1612.08091*, 2016.
[8] B. M. Terhal, "Quantum error correction for quantum memories," *Reviews of Modern Physics*, vol. 87, p. 307, 2015.
[9] 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.
[10] K. M. Svore, A. V. Aho, A. W. Cross, I. Chuang, and I. L. Markov, "A layered software architecture for quantum computing design tools," *Computer*, pp. 74–83, 2006.
[11] 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.
[12] M. Amy, M. Roetteler, and K. Svore, "Verified compilation of space-efficient reversible circuits," *arXiv:1603.01635*, 2016.
[13] D. Kudrow, K. Bier, Z. Deng, D. Franklin, Y. Tomita, K. R. Brown, and F. T. Chong, "Quantum rotations: a case study in static and dynamic machine-code generation for quantum computers," in *ACM SIGARCH Computer Architecture News*. ACM, 2013, pp. 166–176.
[14] A. Paetznick and K. M. Svore, "Repeat-Until-Success: Non-deterministic decomposition of single-qubit unitaries," *Quantum Information & Computation*, vol. 14, no. 15-16, pp. 1277–1301, 2014.
[15] J. Heckey, S. Patil, A. JavadiAbhari, A. Holmes, D. Kudrow, K. R. Brown, D. Franklin, F. T. Chong, and M. Martonosi, "Compiler management of communication and parallelism for quantum computation," in *ACM SIGARCH Computer Architecture News*. ACM, 2015, pp. 445–456.
[16] S. Balensiefer, L. Kregor-Stickles, and M. Oskin, "An evaluation framework and instruction set architecture for ion-trap based quantum micro-architectures," in *ACM SIGARCH Computer Architecture News*, vol. 33. IEEE Computer Society, 2005, pp. 186–196.
[17] R. S. Smith, M. J. Curtis, and W. J. Zeng, "A practical quantum instruction set architecture," *arXiv:1608.03355*, 2016.
[18] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, "QISKit OPENQASM," *arXiv:1707.03429*, 2017.

[19] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[20] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, "Topological quantum memory," *Journal of Mathematical Physics*, vol. 43, pp. 4452–4505, 2002.

[21] A. G. Fowler, "Minimum weight perfect matching of fault-tolerant topological quantum error correction in average o (1) parallel time," *Quantum Information and Computation*, vol. 15, pp. 145–158, 2015.

[22] J. Kelly, R. Barends, A. G. Fowler, A. Megrant, E. Jeffrey, T. C. White, D. Sank, J. Y. Mutus, B. Campbell, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, I. C. Hoi, C. Neill, P. J. J. O'Malley, C. Quintana, P. Roushan, A. Vainsencher, J. Wenner, A. N. Cleland, and J. M. Martinis, "State preservation by repetitive error detection in a superconducting quantum circuit," *Nature*, vol. 519, no. 7541, pp. 66–69, 2015.

[23] D. Ristè, S. Poletto, M.-Z. Huang, A. Bruno, V. Vesterinen, O.-P. Saira, and L. DiCarlo, "Detecting bit-flip errors in a logical qubit using stabilizer measurements," *Nature Communications*, vol. 6, p. 6983, 2015.

[24] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, J. M. Chow, and J. M. Gambetta, "Hardware-efficient quantum optimizer for small molecules and quantum magnets," *arXiv:1704.05018*, 2017.

[25] C. Monroe, D. Meekhof, B. King, W. M. Itano, and D. J. Wineland, "Demonstration of a fundamental quantum logic gate," *Physical Review Letters*, vol. 75, p. 4714, 1995.

[26] S. Debnath, N. Linke, C. Figgatt, K. Landsman, K. Wright, and C. Monroe, "Demonstration of a small programmable quantum computer with atomic qubits," *Nature*, vol. 536, pp. 63–66, 2016.

[27] 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.

[28] G. De Lange, Z. Wang, D. Riste, V. Dobrovitski, and R. Hanson, "Universal dynamical decoupling of a single solid-state spin from a spin bath," *Science*, vol. 330, no. 6000, pp. 60–63, 2010.

[29] J. Cramer, N. Kalb, M. A. Rol, B. Hensen, M. S. Blok, M. Markham, D. J. Twitchen, R. Hanson, and T. H. Taminiau, "Repeated quantum error correction on a continuously encoded qubit by real-time feedback," *Nature Communications*, vol. 7, 2016.

[30] J. M. Hornibrook, J. I. Colless, I. D. Conway Lamb, S. J. Pauka, H. Lu, A. C. Gossard, J. D. Watson, G. C. Gardner, S. Fallahi, M. J. Manfra, and D. J. Reilly, "Cryogenic control architecture for large-scale quantum computing," *Physical Review Applied*, vol. 3, p. 024010, 2015.

[31] J. M. Chow, L. DiCarlo, J. M. Gambetta, F. Motzoi, L. Frunzio, S. M. Girvin, and R. J. Schoelkopf, "Optimized driving of superconducting artificial atoms for improved single-qubit gates," *Physical Review A*, vol. 82, p. 040305, 2010.

[32] M. D. Reed, "Entanglement and quantum error correction with superconducting qubits," Ph.D. dissertation, Yale University, 2013.

[33] J. Koch, M. Y. Terri, J. Gambetta, A. A. Houck, D. I. Schuster, J. Majer, A. Blais, M. H. Devoret, S. M. Girvin, and R. J. Schoelkopf, "Charge-insensitive qubit design derived from the cooper pair box," *Physical Review A*, vol. 76, p. 042319, 2007.

[34] A. Blais, R.-S. Huang, A. Wallraff, S. Girvin, and R. J. Schoelkopf, "Cavity quantum electrodynamics for superconducting electrical circuits: An architecture for quantum computation," *Physical Review A*, vol. 69, p. 062320, 2004.

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

[36] M. Takita, A. Córcoles, E. Magesan, B. Abdo, M. Brink, A. Cross, J. M. Chow, and J. M. Gambetta, "Demonstration of weight-four parity measurements in the surface code architecture," *Physical Review Letters*, vol. 117, p. 210505, 2016.

[37] C. C. Bultink, M. A. Rol, T. E. O'Brien, X. Fu, B. Dikken, R. Vermeulen, J. C. de Sterke, A. Bruno, R. N. Schouten, and L. DiCarlo, "Active resonator reset in the nonlinear dispersive regime of circuit QED," *Physical Review Applied*, vol. 6, p. 034008, 2016.

[38] L. DiCarlo, J. M. Chow, J. M. Gambetta, L. S. Bishop, B. R. Johnson, D. I. Schuster, J. Majer, A. Blais, L. Frunzio, S. M. Girvin, and R. J. Schoelkopf, "Demonstration of two-qubit algorithms with a superconducting quantum processor," *Nature*, vol. 460, pp. 240–244, 2009.

[39] L. DiCarlo, M. D. Reed, L. Sun, B. R. Johnson, J. M. Chow, J. M. Gambetta, L. Frunzio, S. M. Girvin, M. H. Devoret, and R. J. Schoelkopf, "Preparation and measurement of three-qubit entanglement in a superconducting circuit," *Nature*, vol. 467, pp. 574–578, 2010.

[40] R. Barends, J. Kelly, A. Megrant, A. Veitia, D. Sank, E. Jeffrey, T. C. White, J. Mutus, A. G. Fowler, B. Campbell, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, C. Neill, P. O'Malley, P. Roushan, A. Vainsencher, J. Wenner, A. N. Korotkov, A. N. Cleland, and J. M. Martinis, "Superconducting quantum circuits at the surface code threshold for fault tolerance," *Nature*, vol. 508, no. 7497, pp. 500–503, 2014.

[41] S. Bettelli, T. Calarco, and L. Serafini, "Toward an architecture for quantum programming," *The European Physical Journal D-Atomic, Molecular, Optical and Plasma Physics*, vol. 25, pp. 181–200, 2003.

[42] T. Häner, D. S. Steiger, K. Svore, and M. Troyer, "A software methodology for compiling quantum programs," *arXiv:1604.01401*, 2016.

[43] D. Kielpinski, C. Monroe, and D. J. Wineland, "Architecture for a large-scale ion-trap quantum computer," *Nature*, vol. 417, pp. 709–711, 2002.

[44] D. D. Thaker, T. S. Metodi, A. W. Cross, I. L. Chuang, and F. T. Chong, "Quantum memory hierarchies: Efficient designs to match available parallelism in quantum computing," in *ACM SIGARCH Computer Architecture News*, vol. 34. IEEE Computer Society, 2006, pp. 378–390.

[45] D. P. DiVincenzo, "Fault-tolerant architectures for superconducting qubits," *Physica Scripta*, vol. 2009, p. 014020, 2009.

[46] T. Brecht, W. Pfaff, C. Wang, Y. Chu, L. Frunzio, M. H. Devoret, and R. J. Schoelkopf, "Multilayer microwave integrated quantum circuits for scalable quantum computing," *NPJ Quantum Information*, vol. 2, p. 16002, 2016.

[47] C. D. Hill, E. Peretz, S. J. Hile, M. G. House, M. Fuechsle, S. Rogge, M. Y. Simmons, and L. C. Hollenberg, "A surface code quantum computer in silicon," *Science Advances*, vol. 1, p. e1500707, 2015.

[48] M. Oskin, F. T. Chong, and I. L. Chuang, "A practical architecture for reliable quantum computers," *Computer*, vol. 35, pp. 79–87, 2002.

[49] T. S. Metodi, D. D. Thaker, and A. W. Cross, "A quantum logic array microarchitecture: Scalable quantum data movement and computation," in *Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2005, pp. 305–318.

[50] E. Chi, S. A. Lyon, and M. Martonosi, "Tailoring quantum architectures to implementation style: a quantum computer for mobile and persistent qubits," in *ACM SIGARCH Computer Architecture News*, vol. 35. ACM, 2007, pp. 198–209.

[51] L. Kreger-Stickles and M. Oskin, "Microcoded architectures for ion-tap quantum computers," in *35th International Symposium on Computer Architecture*. IEEE, 2008, pp. 165–176.

[52] R. Van Meter and C. Horsman, "A blueprint for building a quantum computer," *Communications of the ACM*, vol. 56, pp. 84–93, 2013.

[53] N. C. Jones, R. Van Meter, A. G. Fowler, P. L. McMahon, J. Kim, T. D. Ladd, and Y. Yamamoto, "Layered architecture for quantum computing," *Physical Review X*, vol. 2, p. 031007, 2012.

[54] M. V. Wilkes, "The best way to design an automatic calculating machine," in *The early British computer conferences*. MIT Press, 1989, pp. 182–184.

[55] S. Vassiliadis, S. Wong, and S. Cotofana, "Microcode processing: Positioning and directions," *IEEE Micro*, vol. 23, no. 4, pp. 21–30, 2003.

[56] S. Beauregard, "Circuit for Shor's algorithm using $2n + 3$ qubits," *arXiv:quant-ph/0205095*, 2002.

[57] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, 1994, pp. 124–134.

[58] C. A. Ryan, B. R. Johnson, D. Ristè, B. Donovan, and T. A. Ohki, "Hardware for dynamic quantum computing," *arXiv:1704.08314*, 2017.

[59] R. BBN, "Bbn technologies arbitrary pulse sequencer 2," 2017.

[60] J. M. Epstein, A. W. Cross, E. Magesan, and J. M. Gambetta, "Investigating the limits of randomized benchmarking protocols," *Physical Review A*, vol. 89, no. 6, p. 062321, 2014.