A Heterogeneous Quantum Computer Architecture

Invited Paper

X.Fu, L.Riesebos, L.Lao, C.G.Almudever, F.Sebastiano, R.Versluis, E.Charbon, K.Bertels
QuTech, Delft University of Technology, the Netherlands
x.fu-1@tudelft.nl, L.Riesebos@student.tudelft.nl,
{L.Lao, C.GarciaAlmudever-1, F.Sebastiano, R.Versluis, E.Charbon, K.l.m.Bertels@tudelft.nl}

ABSTRACT

In this paper, we present a high level view of the heterogeneous quantum computer architecture as any future quantum computer will consist of both a classical and quantum computing part. The classical part is needed for error correction as well as for the execution of algorithms that contain both classical and quantum logic. We present a complete system stack describing the different layers when building a quantum computer. We also present the control logic and corresponding data path that needs to be implemented when executing quantum instructions and conclude by discussing design choices in the quantum plane.

Categories and Subject Descriptors

C.1 [**Computer systems organization**]: Quantum computing; B.10 [**Hardware**]: Quantum error correction and fault tolerance

Keywords

Quantum Computer (Micro-)architecture

1. INTRODUCTION

Research on quantum computing started in 1982 when Richard Feynman suggested to use a quantum system to simulate another quantum system [11]. The basic idea is to exploit two fundamental phenomena of quantum mechanics, superposition and entanglement, providing exponential compute capacity, as will be explained later, such that quantum computers can solve some class of problems that are intractable by classical computers [27].

Superposition A classical bit has two exclusive states, 0 or 1 and can only be in one state at any point in time. In contrast, the elementary unit of quantum computers, the quantum bit or qubit, can reside not only in a single basis state $|0\rangle$ or $|1\rangle$ but also in a superposition of both states, $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. $\alpha, \beta \in \mathbb{C}$ are probability amplitudes that satisfy $|\alpha|^2 + |\beta|^2 = 1$. $|\alpha|^2$ and $|\beta|^2$ represent the probability of getting the measurement result $+1$ or -1, corresponding to states $|0\rangle$ or $|1\rangle$ respectively, when measuring the qubit in the computational basis. The act of measuring the qubit will project the state of the qubit onto one of the basis states, implying that a quantum state cannot be measured directly without losing the stored information.

Entanglement In classical computing, a system composed by n classical bits can only store and process **one** of the 2^n possible states at a time. However,in quantum computing multiple qubits can be combined, resulting in a new state that is a superposition of all 2^n possible states $|\psi\rangle = \alpha_0|0\cdots00\rangle + \alpha_1|0\cdots01\rangle + \cdots + \alpha_{2^n-1}|1\ldots11\rangle$, where $\alpha_i \in \mathbb{C}, \sum|\alpha_i|^2 = 1$. Entanglement is a special case of such combination meaning that the combined qubit state cannot be decomposed into separate states. When applying a (quantum) operation on those combined qubits, the operation is applied on those 2^n possible states at the same time.

The exponential large state space offered by superposition and entanglement and a universal set of quantum operations are the foundation for the exponential speedup of a quantum computer.

Even though the potential of quantum computing is huge, the achilles heel of quantum technology is the fragility of the qubits. Through interaction between the qubits and the environment, the information of qubits leaks to the environment, called *decoherence*. The qubits' fragility is therefore one of the main challenges for building and using a quantum computer as this behaviour causes errors during computation. Quantum error correction (QEC) mechanisms are needed to make quantum computing fault-tolerant and is therefore a key component of any quantum computer architecture, as will be discussed in section 3.

The challenges to build a circuit-model based quantum computer - called the standard universal quantum computer- are huge. One of the main physics challenges is to increase the number of qubits per chip that can be entangled as well as to make their lifetime longer and the operation fidelity higher. The engineering challenges focus on the technology necessary to provide high speed control logic in a way which is scalable and flexible. This paper focuses on the architectural and system design challenges [1]. The main contributions of the paper are the following:

- This paper provides the first systematic discussion of

CF'16, May 16-18, 2016, Como, Italy
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-4128-8/16/05...$15.00
DOI: http://dx.doi.org/10.1145/2903150.2906827.

[1]This work is funded by Intel in the context of the QuTech-Intel collaboration.

the functionality that fills the gap between algorithms and the quantum physical layer;

- To this purpose, a multi-layered system stack for a quantum computer is defined;
- A heterogeneous micro-architecture is presented for the control logic and corresponding data path based on a real experimental quantum device [32];
- We discuss how such an architecture can be made as technology independent as possible and define a hardware mechanism that substantially reduces the code-size of the executable and reduces the execution overhead.

The paper is organized as follows. We begin by introducing related research in Section 2 followed by the basic idea of quantum error correction in Section 3. Then we discuss the different system layers of a quantum computer in section 4. In section 5, we introduce the classical architecture that provides the necessary support for the execution of quantum instructions with corresponding error correction. We conclude the paper by briefly discussing the architectural design choices that also need to be made for the quantum plane.

2. RELATED RESEARCH

The first quantum algorithm was proposed by David Deutsch in 1985 along with the Quantum Turing Machine [9]. A lot of work has been done since then and nowadays we can find more than 50 basic quantum algorithms [37]. The most representative example of a quantum algorithm is the famous Shor's Factoring algorithm [34] that is used to factorize a very large number. It is the first quantum algorithm that has an application to real world problems, e.g. decryption, and shows exponential speed up over its classical counterparts. However, Shor's algorithm requires millions if not billions of physical qubits for factorizing big numbers [12]. That is why it becomes increasingly appealing to investigate quantum algorithms with low number of qubits, such as the variational eigenvalue solver [40].

In terms of QEC, the first quantum error correction code (QECC) called Shor's code was proposed by Peter Shor in 1995 [35]. Since then, many other codes have been proposed such as CSS codes [7]. In 1997, Kitaev introduced the idea of topological methods for implementing QEC [22], forming the basis of Surface Code [12]. Other promising topological codes are topological subsystem codes and color codes [3,5].

Since the ground-breaking experimental work of Haroche and Wineland demonstrating the measurement and manipulation of individual quantum systems [38], different quantum technologies have been investigated such as quantum dots [1], trapped ions [26], superconductors [8,21,32] and photons [42]. Current state-of-the-art quantum chips contain around 10 qubits. In the rest of the paper, we assume superconducting qubits given its potential for scalability.

Along with the progress in superconducting quantum chips, classical control in the experiment has been advancing from simple pulse generation by various AWGs and digitizers to customized devices for feedback control [32]. However, these control schemes suffer from low speed feedback loops and limited scalable control logic. No comprehensive solution with a scalable and flexible (micro-)architecture has been proposed yet.

Several quantum programming languages [15, 28, 33, 41] and compilers for e.g. circuit decomposition and reversible circuit design [19, 31, 41] have been developed.

Few papers have tried to formulate in a systematic way what the different components or layers are of a quantum computer. Van Meter *et al.* [39] proposed a high level view consisting of related research domains rather than an implementable architecture. Jones *et al.* [20] defined a layered control stack of a quantum computer architecture but focuses more on the gate abstractions rather than the architectural support.

3. QUANTUM ERROR CORRECTION

As we already mentioned, the main handicap of quantum technology is its fragility. First, the coherence time of qubits is extremely short. For example, superconducting qubits may lose its information in tens of microseconds [8,32]. Second, quantum operations are unreliable, error rate around 0.1% [21]. It is therefore inconceivable to think about building a quantum computer without error correction.

QEC is more challenging than classical error correction because of the following: 1) unknown quantum states cannot be copied (no-cloning theorem), 2) errors are continuous and 3) measurement may destroy the information stored in qubits.

The basic idea of QEC techniques is to use several physical imperfect qubits to compose more reliable units called logical qubits based on a specific quantum error correction code [4, 7, 14, 35, 36] - e.g. surface code [12]. Such encoding does not need to clone the qubit state, as it is done by entangling several qubits called **data qubits**. Furthermore, possible errors in the logical qubit are detected by measuring some 'helper' qubits called **ancilla qubits**. In this way, the information in data qubits can be preserved. Measurement forces continuous errors into discrete errors and allows identifying whether there are errors, and if yes what kind (bit-flip, phase-flip or both) and in which qubit(s) the error(s) are. This kind of measurement is called error syndrome measurement (ESM). Since quantum errors accumulate as time elapses, ESM has to be done repeatedly.

It is worth noting that errors do not need to be corrected immediately. Instead, errors are tracked in classical logic using a technique called **Pauli Frame** [23]. Quantum operations and measurements are translated by the Pauli Frame, preserving the correctness of quantum computation.

One of the most promising and currently very popular error correction techniques for quantum systems is **surface code** [12]. In surface code, qubits are arranged in a regular 2D lattice which only enables nearest-neighbour (NN) interaction (Figure 1). The NN lattice architecture is one of the most-promising structure from theory to experiment [8, 10, 17, 21, 32]. Surface code has an error threshold rate of $\sim 1\%$ [12] meaning that it can tolerate a physical error rate up to 0.01.

In Figure 1, open circles represent data qubits and green and red filled circles correspond to Z and X ancilla qubits, respectively. In surface code, the ESM is done by performing the circuits shown in Figure 2. They can detect phase-flip errors (through the X ancilla) and bit-flip errors (through the Z ancilla). As we mentioned, these circuits will be repeatedly applied during computation. The interval between the starting point of two consecutive ESM is called a Surface Code Cycle (SC cycle). In surface code, the measurement results $+1/-1$ coming from several rounds of ESM, are

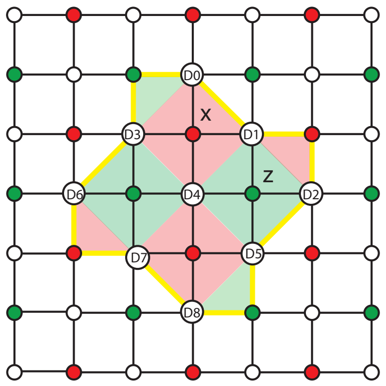

Figure 1: Implementation of the surface code 17, also called Ninja star.

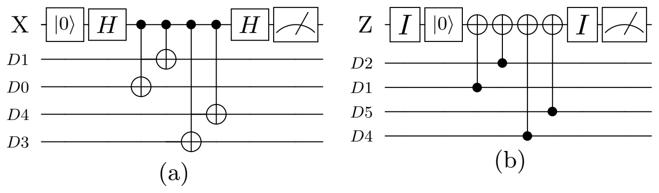

Figure 2: ESM circuit for (a) X ancilla (b) Z ancilla with surrounding data qubits, taking as example an X and Z ancilla as shown in Figure 1.

then forwarded into classical logic where errors are identified (quantum error detection, QED) by using decoding algorithms, such as Blossom algorithm [13].

In this paper, we restrict our discussion to planar surface code. An important metric is the code distance which is the minimum number of physical operations required to perform a logical operation, or the length of the shortest error chain that is undetectable [12]. The distance-3 surface code is shown with a yellow frame in Figure 1. It contains 17 qubits, nine of them are data qubits and eight of them are ancilla qubits. It is also called a Ninja star due to their similarity in shape. The implementation of logical operations for the Ninja star is listed in Table 1. For instance, a logical X operation is performed by applying three physical X operations in qubits 0, 4 and 8, $X_0X_4X_8$.

QEC allows fault tolerant computation and is thus a fundamental part of any quantum computing system. The drawback is that it dramatically increases the number of physical qubits needed for running any quantum algorithm. It also creates a large control overhead and requires a continuous and close interaction between the quantum chip and the classical platform that makes the quantum computing process more complex in terms of integration, architecture, and run-time control as will be described in the following sections.

4. QUANTUM SYSTEM STACK

A quantum computer will always consist of both quantum and conventional computing components because of the following two reasons: the quantum algorithms and consequently the quantum applications that will be executed, consist of both classical as well as quantum parts and will thus be executed by their respective computing blocks [27].

Table 1: Logical Operations for Ninja Star

Logical	**Physical Implementation**
X / Z	$X_0X_4X_8$ / $Z_2Z_4Z_6$
H	1. $H_0H_1\cdots H_8$ 2. Z / X ancilla turn into X / Z ancilla
Msmt	1. Measure all data qubit with result $M_i\|_{i=0,1,\cdots 8}$ 2. Perform 3 rounds of ESM only for Z ancilla. 3. Correct errors in M_i 4. Logical measurement result $M_L = \prod_{i=0}^{8} M_i$
Init	1. Reset all data qubits into the $\|0\rangle$ or $\|1\rangle$ state. 2. Perform 3 rounds of ESM. 3. Correct errors appearing in step 1 & 2. 4. The final logical state is $\|0_L\rangle$ or $\|1_L\rangle$.
T	State inject, see [18].
CNOT	Perform a CNOT on each data qubits q_i of the target Ninja star with each controlled by q_i of the control Ninja star.

The second reason is that, as is explained in Section 3, a quantum computer requires very close monitoring and, if necessary, correction by classical logic. Figure 3 provides a high-level view of the quantum system stack consisting of the following layers. The top layers represent the algorithms for which specific language constructs and compilers need to be developed such that the algorithms can exploit the underlying quantum hardware. Here the qubits are defined as logical qubits. Figure 4 depicts the compiler infrastructure consisting of a conventional host compiler and a quantum accelerator compiler. The former compiles the classical logic and the latter will produce the quantum circuits. The quantum compiler will perform quantum gate decomposition, reversible circuit design, and circuit mapping and translates the logical quantum operations to a series of physical operations. As represented by the third dimension in the figure, the logical-to-physical quantum instruction translation is driven by choices regarding the error correction encoding scheme for the logical qubits.

The next layer is the Quantum Instruction Set Architecture (QISA) which is the dividing line between hardware and software. The algorithm designer and programmer are offered a logical instruction set with the possibility to opt for certain encoding schemes, thus exposing the relevant error correction functionality. As stated above, the compiler will translate logical instructions into the physical instructions that belong to the QISA and for which architectural support is provided. Examples of physical instructions in the QISA are initialization, measurements and quantum gates such as Hadamard and CNOT.

The quantum execution (QEX) block will execute the quantum instructions that are generated by the compiler infrastructure. It will also provide the necessary hardware support such as the insertion of ESM circuits or the use of Pauli Frames for software based error tracking. These operations are finally sent to the quantum-classical interface, which will apply the proper electrical signals to the quantum chip. Note that the quantum-classical interface is responsible for all the conversions between the analog qubit plane and the digital layers in the system stack. The QEC layer, is responsible for the error detection and correction. It will receive from the quantum-classical interface layer the ESM data which it will process to identify possible errors. Then,

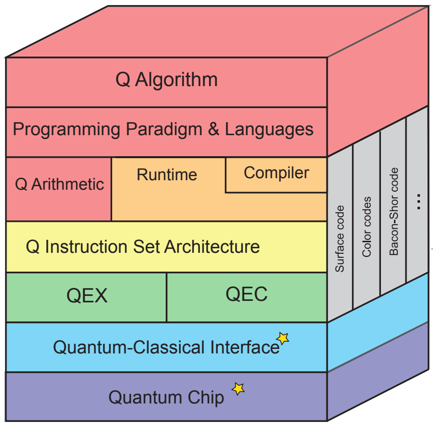

Figure 3: Overview of quantum computer system stack

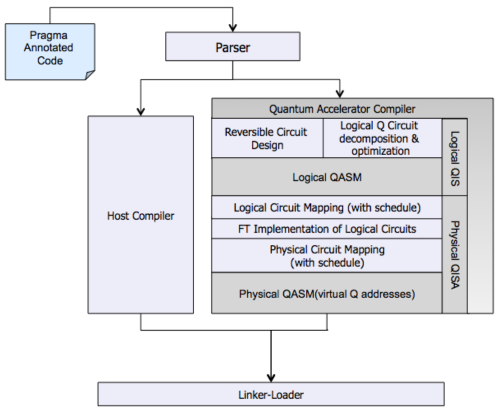
Pragma Annotated Code
Parser
Host Compiler
Quantum Accelerator Compiler
Reversible Circuit Design
Logical Q Circuit decomposition & optimization
Logical QASM
Logical QIS
Logical Circuit Mapping (with schedule)
FT Implementation of Logical Circuits
Physical Circuit Mapping (with schedule)
Physical QASM(virtual Q addresses)
Physical QISA
Linker-Loader

Figure 4: Compiler infrastructure

it will make the required corrections by updating the Pauli frame or by sending the appropriate corrective operations when required.

It is worth noting that the classical-quantum interface together with the quantum chip are technology dependent represented by the star symbol in the figure. In the next section, we will further specify from where on the architecture is technology independent. Defining a layered system stack allows to develop the functionality of the quantum computer as independently as possible from other layers and independent of the underlying quantum technology.

5. CLASSICAL CONTROL

This section describes at a high level the micro-architecture for a quantum computer to support the execution of the physical quantum instructions. The overall architecture, as shown in Figure 5 is heterogeneous where the rightmost block of the figure represents the quantum chip (QUBE) and the other blocks represent the classical logic needed to control the quantum chip. As was explained above, a lot of conventional processing is necessary to provide the required error correction feedback after each quantum operation. In addition, quantum algorithms will always be a mixture of classical logic and quantum routines which can finally be executed on the quantum chip. In the following paragraphs, we discuss the functional blocks that are needed to execute instructions in QISA. These blocks are based on the control logic as developed for the transmon processor as described in [32].The green parts are underlying technology dependent wave control modules. Digital-to-Analog Converters (DAC) are used to generate analog waveforms to drive the quantum chip and Analog-to-Digital Converters (ADC) to read the measurement analog waveform. They receive or send signals to the Flux and Wave Control Unit and the Measurement Discrimination Unit. The Physical Execution Layer (PEL) serves as middle layer between the Quantum Control Unit (QCU) and the wave control modules, making the Quantum Control Unit technology-independent. The QCU decodes the instructions belonging to the QISA and performs the required quantum operations, feedback control and QEC. The QCU can also communicate with the host CPU where classical computation is carried through the eXchange Register File (ERF).

5.1 Quantum Control Unit

We assume that one binary is loaded in memory and the instruction fetch unit fetches the instructions. Based on the opcode of the instruction, the arbiter sends the instruction either to the host CPU or to the QCU. In the remainder of the text, we focus on the architectural support for the execution of quantum instructions and not on the execution of instructions on the classical CPU. As explained above, the compiler maps a circuit using logical and virtual qubit addresses for the logical and physical qubits respectively. Instructions from the Quantum Instruction Cache are first address-translated by the Q-Address Translation module. This means that the compiler-generated, virtual qubit addresses are translated into physical ones. This is based on the information contained in the Q Symbol Table which provides the overview of the exact physical location of the logical qubits and provides information on what logical qubits are still alive.

Q Symbol Table

The Qubit Symbol Table, with a sample entry in Table 2, provides the following information:

- The mapping of logical qubits to the virtual and physical addresses of the physical qubits (*Physical/ Virtual/ Logical QID* and *Phys. Addr* field);
- What physical qubits are available to be allocated to a logical qubit (*valid* field);
- Bookkeeping of operations for every round of error syndrome measurement (*Associated Data Qubit* field).
- The type of qubit: data qubit, Z ancilla or X ancilla (*Type* field).

Runtime information about the state of the logical qubits is essential for logical feedback control and high execution speed. An example of algorithms requiring such feedback facility is Shor's Factoring algorithm where the number of qubits can be substantially reduced by mapping the inverse Quantum Fourier Transform on n qubits into multiple Binary-Controlled (BC) Z-Rotations on a single qubit at n different

Table 2: An example of Q Symbol table

Valid	Phys. QID	Phys. Addr	Virtual QID	Logical QID	Associated Data Qubit	Type
1	11	(3, 3)	22	1	1, 0, 4, 3	Z

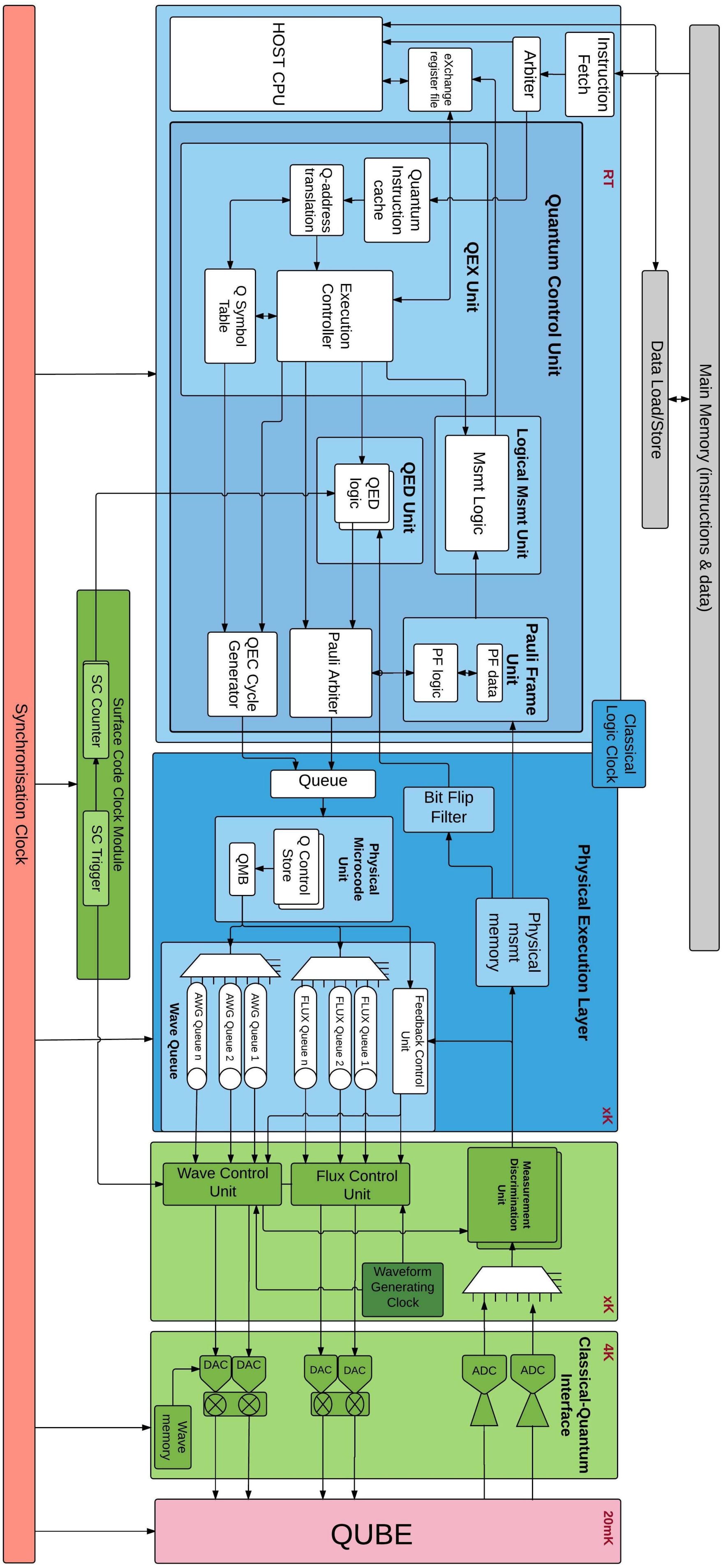

Figure 5: Overview of Quantum Computer Micro-architecture

time points [2]. Also, fast ESM circuit generation and QED is better done in hardware than software, which requires layout information of logical qubits in surface code.

Execution Controller

The Execution Controller can be seen as the brain of the Quantum Control Unit. The Execution Controller then decodes the various instructions that are fetched:

- **Physical gate/measurement/reset**: The Execution Controller sends these instructions to the Pauli Arbiter for further processing.
- **Update Q Symbol Table**: based on a series of instructions such as the ones performing a logical Hadamard or a logical measurement, the Q Symbol Table needs to be updated. Also for deallocating qubits such an update is needed.
- **ECC slot**. The Execution Controller sends this instruction to the QEC cycle Generator, which is triggered to generate a round of error syndrome measurement circuit based on Figure 2.

QEC Cycle Generator

As far as error correction is concerned, the necessary ESM instructions for the entire qubit plane are added at runtime by the QEC Cycle Generator, based on the information stored in Q Symbol Table. In Figure 6, we plot the number of instructions as a function of the number of physical qubits. The top plot shows how many ESM instructions are needed as generated by the compiler and the bottom plot shows the substantial reduction in codesize through HW generation. In addition, it reduces substantially the datapath for the execution of these instructions.

QED Unit

The responsibility of the QED Unit is to detect errors based on ESM results. The decoder will use decoding algorithms such as Blossom algorithm. QED starts to work only when d rounds of error syndrome are collected, where d is the Surface Code distance.

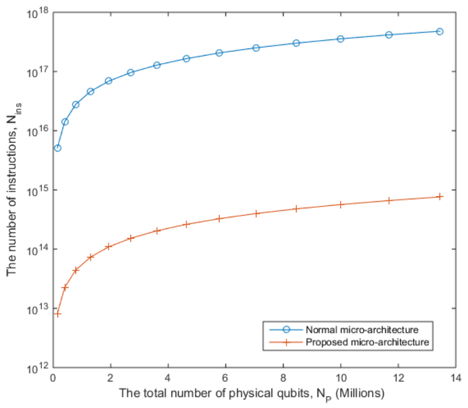

Figure 6: Reduction of number of instructions through HW support

The Pauli Frame Unit and Pauli Arbiter

The Pauli Frame mechanism [23] allows us to classically track Pauli errors without physically correcting them. The Pauli Frame Unit manages the Pauli records for every data qubit. The Pauli Arbiter receives instructions from the Execution Controller and the QED Unit. It skips all operations on ancilla qubits and sends them directly to the PEL, regardless of the operation type. Operations on data qubits are processed as follows based on instruction type:

- Pauli gate: The Pauli Arbiter sends this instruction directly to the Pauli Frame Unit, and no operation is sent to the PEL. The Pauli Frame Unit updates the Pauli records of the corresponding data qubit(s).
- Clifford gate: The Pauli Arbiter sends the operation to both PEL and the Pauli Frame Unit. The Pauli Frame Unit updates the Pauli records accordingly.
- Non-Clifford gate: When executing a non-Clifford gate, the involved data qubits need to be physically updated (corrected) by applying the Pauli gate information stored in the PF unit. The PF Logic unit will generate the necessary instructions and sends them to the arbiter who the forwards them to the PEL, followed by the non-Clifford instructions.
- Measurement: The Pauli Arbiter sends this instruction to the PEL to trigger physical qubit measurements. After the measurement results comes back from the PEL, they are translated based on the Pauli records for the target qubits. The translated results are then sent to the Logic Measurement Unit for further processing. Finally, the Pauli Frame Unit removes the Pauli records for the target qubits.

Logical Measurement Unit

The function of the Logical Measurement Unit is to combine the data qubit measurement results into a logical measurement result for a logical qubit. Once the Execution Controller receives a logic measurement instruction on a specified logical qubit, it notifies the Logical Measurement Unit to wait for measurement results to arrive from the Measurement Discrimination Unit.

Based on this information the Pauli Frame Unit updates the Pauli records. After all translated data qubit measurement results come, the Logical Measurement Unit computes the logical measurement result. The QED Unit also contributes to this procedure by identifying possible errors that may appear during the execution of quantum operations. After that, the Logical Measurement Unit sends the logical measurement result to the ERF, where it can be used in Binary Control by Execution Controller, or picked up by the host processor and used e.g. in branch decisions.

5.2 Physical Execution Layer

Physical Microcode Unit

The next hardware block is the Physical Microcode Unit. There are three main reasons to introduce a microcoded approach. Each quantum technology will have its own way to perform the different (universal) quantum gates. When defining an architecture, it should be as independent as possible of any particular technology and using micro-instructions is one of way of isolating the technology impact. The second and highly related reason is that not all physical quantum gates can be directly applied on the qubits but rather require an (sometimes approximating) sequence of elementary operations. In addition, the microcode approach allows that these sequences are easily changed as technology evolves. As an example, consider the Hadamard gate that needs to be applied on transmon qubits. A Hadamard gate can be defined as state rotations around the X-axis or Y-axis for π or $\pi/2$ angles or $H = iR_x(\pi)R_y(\pi/2)$. As these rotations are well defined for transmon superconducting qubits, the microcode for the transmons Hadamard gate will therefore consist of the sequence of these two rotations. When the underlying quantum technology changes, it suffices to change the microcode unit to enable the new technology.

The Q Control Store in the Physical Microcode Unit generates, for a particular physical quantum instruction, the sequence of micro-instructions (or pulses) with detailed timing information based on the underlying technology. The timing is relative to a signal generated by the Surface Code Clock indicating the starting point of the SC cycle the physical operations should lie in. The output from the Q Control Store is first buffered in the Quantum Microcode Buffer (QMB) before being sent to the Flux and AWG Queues for execution in the form of the tuple (codeword, timing). The codeword identifies the specific AWG and the timing is relative to the SC trigger.

Flux and Wave Control Units

The Flux Control Unit puts the qubits in a certain frequency by tuning the bias flux strength and the Wave Control Unit sends the pulse sequence received from the Physical Microcode Unit to apply operations on qubits. Flux Queues store the flux control information, which is used to tune the qubit and cavity frequency at run-time. The Flux Control Unit works in the same way as the Wave Control Unit.

Feedback Control Unit

Physical feedback is needed for two reasons. First, when initialising a logical qubit, an active physical qubit reset is performed (as opposed to a passive one wich implies waiting for the qubit to relax). Second, recent experiments [6] show that feedback on ancilla qubits can improve the overall QEC quality, which is essential to quantum computing.

The QMB sends the microcode to the Feedback Control Unit.After the measurement on the target qubit is done, the Feedback Control Unit bypasses the AWG or Flux Queues and triggers directly the Flux and Wave Control Unit to perform certain operations based on the physical measurement result.

Measurement Discrimination Unit

After sending a measurement pulse to the quantum circuit, it is required to discern the measurement result from the analog waveform reflected by the quantum circuit to finish a physical measurement. The ADC unit converts the reflected analog waveform into a digital signal. The Measurement Discrimination Unit then removes the offset in the digital signal, and integrates it with a particular set of weights. The results from the integration are then compared with the integration thresholds after some linear transformation to get the final measurement result, which are either +1s or -1s.

6. QUBE: THE QUBIT PLANE

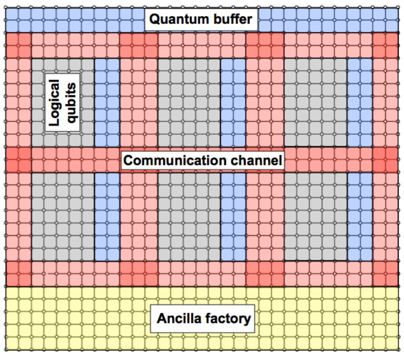

Figure 7: Example QUBE architecture.

Finally, we discuss the way the quantum plane can be organised and what infrastructure is necessary for fast execution and error correction. A first point that needs to be addressed is whether or not a Von Neumann like architecture should be implemented in the quantum plane as advanced by many papers [24, 29, 30]. Just like with classical Von Neumann architectures, the quantum plane is then split in in specialised regions for processing and communication and others to store quantum states. There are two main arguments to not go in that direction: first, the computational paradigm of quantum logic can be seen as the dual of the classical one. The qubit states represent the information as it has been computed up to now and all future operations are applied on these qubits. In principle, there is no movement required of the qubit states to an ALU like component as the quantum gates are directly applied on the qubits. So the logic is streaming through the qubits rather than the opposite. A second reason is that by introducing a Von Neumann architecture, we also introduce all the parallelisation challenges, such as the memory wall issue, that have proven to be very difficult to solve for conventional architectures.

However, as shown in Figure 7, it may still be useful to create specialised regions, for instance to transport qubit states to different parts of the quantum plane. Ancilla factories may also be necessary to create, e.g. EPR pair used for teleporting states [25] and special ancilla states used for implementing fault-tolerant T and S gates [12]. What functional specialisation of the qubit is necessary and how rigid or flexible that should be is still an open issue. The two extreme views are that the qubit plane can be seen as completely undefined for which an ad hoc infrastructure will be generated, or that the architecture is completely pre-defined as proposed in [16,30]. In this sense, it is important to investigate what the trade-offs are for both choices given different benchmarks or applications.

7. CONCLUSION

In this paper, we presented the first systematic description of a heterogeneous architecture for a quantum computer and we defined the system layers of such a computing platform. We discussed the different system layers that are needed to build a quantum computer and we have described the datapath of quantum instructions as far as execution and error correction are concerned. We also showed how hardware choices can substantially reduce not only the codesize but also the datapath for these ESM instructions.

Future work involves not only the development of the different hardware blocks that were described but also the development of a digital quantum processor such that the control logic can be easily tested on a number of qubits which is larger than current day devices can offer.

8. REFERENCES

[1] T. A. Baart et al. Single-spin ccd. *Nature nanotechnology*, 2016.

[2] S. Beauregard. Circuit for shor's algorithm using 2n+ 3 qubits. *arXiv preprint quant-ph/0205095*, 2002.

[3] H. Bombin and M. Martin-Delgado. Topological computation without braiding. *Phys. Rev. Lett.*, 98(16):160502, 2007.

[4] H. Bombin and M. A. Martin-Delgado. Topological quantum distillation. *Phys. Rev. Lett.*, 97(18):180501, 2006.

[5] S. Bravyi, G. Duclos-Cianci, D. Poulin, and M. Suchara. Subsystem surface codes with three-qubit check operators. *arXiv:1207.1443*, 2012.

[6] C. Bultink et al. In preparation, 2016.

[7] A. R. Calderbank and P. W. Shor. Good quantum error-correcting codes exist. *Phys. Rev. A*, 54(2):1098, 1996.

[8] A. Córcoles et al. Demonstration of a quantum error detection code using a square lattice of four super-conducting qubits. *Nature Comm.*, 6, 2015.

[9] D. Deutsch. Quantum theory, the church-turing principle and the universal quantum computer. In *Proc. of the Royal Society of London A: Math., Physical and Engineering Sciences*, 1985.

[10] D. P. DiVincenzo. Fault-tolerant architectures for superconducting qubits. *Physica Scripta*, 2009(T137):014020, 2009.

[11] R. P. Feynman. Simulating physics with computers. *Intern. J. of Theoretical physics*, 21(6):467–488, 1982.

[12] Fowler et al. Surface codes: Towards practical large-scale quantum computation. *Phys. Rev. A*, 86(3):032324, 2012.

[13] Fowler et al. Towards practical classical processing for the surface code: Timing analysis. *Phys. Rev. A*, 86(4):042313, 2012.

[14] D. Gottesman. Class of quantum error-correcting codes saturating the quantum hamming bound. *Phys. Rev. A*, 54(3):1862, 1996.

[15] A. S. Green et al. An introduction to quantum programming in quipper. In *Reversible Computation*, pages 110–124. Springer, 2013.

[16] J. Heckey et al. Compiler management of communication and parallelism for quantum computation. In *Proc. of the 20th Int. Conf. on Architectural Support for Prog. Languages and Oper. Systems*, pages 445–456. ACM, 2015.

[17] C. D. Hill et al. A surface code quantum computer in silicon. *Science advances*, 1(9):e1500707, 2015.

[18] Horsman et al. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, 2012.

[19] A. JavadiAbhari et al. Scaffcc: A framework for compilation and analysis of quantum computing programs. In *Proc. of the 11th ACM Conference on Computing Frontiers*, page 1. ACM, 2014.

[20] N. C. Jones et al. Layered architecture for quantum computing. *Phys. Rev. X*, 2(3):031007, 2012.

[21] J. Kelly et al. State preservation by repetitive error detection in a superconducting quantum circuit. *Nature*, 519(7541):66–69, 2015.

[22] A. Y. Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303(1):2–30, 2003.

[23] E. Knill. Quantum computing with realistically noisy devices. *Nature*, 434(7029):39–44, 03 2005.

[24] M. Mariantoni et al. Implementing the quantum von neumann architecture with superconducting circuits. *Science*, 334(6052):61–65, 2011.

[25] Meter and other. Arithmetic on a distributed-memory quantum multicomputer. *ACM J. on Emerging Technologies in Computing Systems*, 3(4):2, 2008.

[26] T. Monz et al. 14-qubit entanglement: Creation and coherence. *Phys. Rev. Lett.*, 106(13):130506–, 03 2011.

[27] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge university press, 2010.

[28] B. Omer. Structured quantum programming. *Information Systems*, page 130, 2003.

[29] M. Oskin, F. T. Chong, and I. L. Chuang. A practical architecture for reliable quantum computers. *Computer*, 35(1):79–87, 2002.

[30] W. otherss. A fault tolerant, area efficient architecture for shor's factoring algorithm. *ACM SIGARCH Computer Architecture News*, 37(3):383–394, 2009.

[31] A. Paler, I. Polian, K. Nemoto, and S. J. Devitt. A compiler for fault-tolerant high level quantum circuits. *arXiv:1509.02004*, 2015.

[32] D. Riste, S. Poletto, M. Z. Huang, et al. Detecting bit-flip errors in a logical qubit using stabilizer measurements. *Nat Commun*, 6, 04 2015.

[33] J. W. Sanders and P. Zuliani. Quantum programming. In *Mathematics of Program Construction*, pages 80–99. Springer, 2000.

[34] P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Foundations of Computer Science, 1994, 35th Annual Symp. on*, pages 124–134. IEEE, 1994.

[35] P. W. Shor. Scheme for reducing decoherence in quantum computer memory. *Phys. Rev. A*, 52(4):R2493, 1995.

[36] A. Steane. Multiple-particle interference and quantum error correction. In *Proceedings of the Royal Society of London A: Math., Phys. and Eng. Sciences*, 1996.

[37] J. Stephen. Quantum algorithm zoo. *list available at http://math.nist.gov/quantum/zoo*, 2011.

[38] Measuring and manipulating individual quantum systems, 2012.

[39] R. Van Meter and C. Horsman. A blueprint for building a quantum computer. *Comm. of the ACM*, 56(10):84–93, 2013.

[40] D. Wecker, M. B. Hastings, and M. Troyer. Towards practical quantum variational algorithms. *arXiv preprint arXiv:1507.08969*, 2015.

[41] D. Wecker and K. Svore. Liqui|>: A software design architecture and domain-specific language for quantum

computing. *arXiv preprint arXiv:1402.4467*, 2014.

[42] X.-C. Yao et al. Experimental demonstration of topological error correction. *Nature*, 482(7386):489–494, 02 2012.