Quantum accelerated computer architectures

Riesebos, L.; Fu, X.; Moueddenne, A. A.; Lao, L.; Varsamopoulos, S.; Ashraf, I.; Van Someren, J.; Khammassi, N.; Almudever, C. G.; Bertels, K.

DOI
10.1109/ISCAS.2019.8702488

Publication date
2019

Document Version
Final published version

Published in
2019 IEEE International Symposium on Circuits and Systems, ISCAS 2019 - Proceedings

Citation (APA)
Riesebos, L., Fu, X., Moueddenne, A. A., Lao, L., Varsamopoulos, S., Ashraf, I., Van Someren, J., Khammassi, N., Almudever, C. G., & Bertels, K. (2019). Quantum accelerated computer architectures. In H. Yasuura, Y. Miyanaga, & H. Kiya (Eds.), *2019 IEEE International Symposium on Circuits and Systems, ISCAS 2019 - Proceedings* (Vol. 2019-May). Article 8702488 IEEE. https://doi.org/10.1109/ISCAS.2019.8702488

Quantum Accelerated Computer Architectures

L. Riesebos*, X. Fu*†, A. A. Moueddenne*, L. Lao*, S. Varsamopoulos*, I. Ashraf*, J. van Someren*, N. Khammassi*, C. G. Almudever*, K. Bertels*

* Quantum Computer Architecture Lab, Delft University of Technology, the Netherlands

† State Key Laboratory for High-Performance Computing (HPCL), National University of Defense Technology

Email: l.riesebos@tudelft.nl, xiangfu@quanta.org.cn, {a.a.moueddene, l.lao, s.varsamopoulos, i.ashraf, j.vansomeren-1, n.khammassi, c.garciaalmudever-1, k.l.m.bertels}@tudelft.nl

***Abstract*—Modern computer applications usually consist of a variety of components that often require quite different computational co-processors. Some examples of such co-processors are TPUs, GPUs or FPGAs. A more recent and promising technology that is being investigated is quantum co-processors. In this paper, we present a modern computer architecture where a quantum co-processor is included as an additional accelerator. In such an environment, the idea is to execute the application on a heterogeneous architecture where the classic processor will execute the host part, but certain components will be mapped, in our case, on the quantum accelerator. To this purpose, we define the distinct layers for the quantum computer architecture where there is a clear boundary between the host program and quantum kernel(s). We also discuss the opportunities and challenges of mapping hybrid algorithms to such a heterogeneous quantum computer architecture.**

I. Introduction

Quantum computing is an emerging technology that promises to solve problems which are intractable by classic computers by exploiting quantum phenomena. Much research focuses on quantum devices in the effort of fabricating coherent qubits and having high fidelity gates and only limited research effort has been spent on the programmability of a quantum device and closing the gap between the programming environment and the quantum devices. Therefore, full stack research reaching from programming languages down to the execution level on quantum devices is essential. To tightly integrate all levels of such a system and to interface a quantum device with existing computing infrastructure, many challenges will arise in the field of computer science and engineering. In the remainder of this paper, we will present the layers and challenges of a full-stack quantum system and discuss how to integrate such a quantum system as an accelerator in the current computing infrastructure.

This paper is organized as follows: Section II provides a brief background on the basics of quantum computing. In Section III we present our vision on the system stack and the integration with current computing infrastructure. The engineering challenges are discussed in Section IV. We conclude the paper in Section V.

II. Background

While classic bits can only be in a 0 or 1 state at a certain moment, qubits can be in a *superposition* of computational basis states $|0\rangle$ and $|1\rangle$ which is mathematically described as: $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$, or as a unit vector $|\psi\rangle = \begin{bmatrix}\alpha & \beta\end{bmatrix}^T$. Superposition can not be measured and measurements in the computational basis are projected into the $|0\rangle$ or $|1\rangle$ state with probabilities $|\alpha|^2$ or $|\beta|^2$, respectively. Multiple qubits can be combined to form a vector with 2^n states where n is the number of qubits. Qubits can also be *entangled*, which means that the superposition state of the entangled qubits cannot be represented as a tensor product of individual qubit states.

A qubit state can be manipulated by applying quantum gates which can be expressed as unitary matrices. Operations are applied on the qubits that hold the state which makes it an in-memory computing technology. Common single-qubit gates are the X, Y, Z, H, S, and T gate while the most common two-qubit gates are the controlled-NOT (CNOT) and controlled-phase (CZ) gate. Quantum algorithms can be described as quantum circuits consisting of qubits and gates operating on them (circuit model of computation). For a more exhaustive introduction into the basics of quantum computing, we refer to [1].

Various physical implementations of qubits exist (e.g., ion-traps, spin qubits, superconducting qubits), and all of them suffer from *decoherence*, meaning that qubits lose their state in a short period. For example, superconducting qubits can lose their information in tens of microseconds [2], [3]. In addition, quantum operations are not perfect and have error rates of around 0.1% [4]. To enable meaningful quantum computation with high fidelity, Quantum Error Correction (QEC) was introduced [5]. By using QEC, a quantum state can be encoded redundantly to form a *logical qubit* with lower error rates than the underlying physical qubits. A popular QEC code is the *surface code* [6]–[9]. The resource requirements for QEC are high and Fault-Tolerant (FT) quantum computing will not be possible in the next 5 to 10 years. Therefore, near-term quantum computational devices will operate directly on a relatively low number of noisy physical qubits, also known as Noisy Intermediate-Scale Quantum (NISQ) technology [10].

III. System stack

To enable a fully programmable quantum processor based on the circuit model [1], a full-stack system approach is required going from programming languages to the quantum devices. Various high-level overviews for instruction based quantum processors have been proposed so far [11]–[13] based on similarities and differences with classic processors.

978-1-7281-0397-6/19/$31.00 ©2019 IEEE

There are two apparent differences between classic and quantum processors. First, due to error-prone qubits, large-scale quantum processors require QEC to enable FT computing. QEC becomes the major source of computational activity [6], [11] and therefore plays an important role in the system stack. Second, qubits require interaction to perform two-qubit operations, and quantum devices have limited connectivity, which requires routing [14]–[16] and adds extra constraints to the execution scheme.

As proposed in [13], [15], [17], we suggest a system stack as shown in Figure 1. The two top levels are the application layers where users describe the desired algorithm embracing the circuit model [18] with programming languages such as Q# [19], Scaffold [20], Quipper [21], QCL [22], ProjectQ [23], and OpenQL [24]. Such languages can be embedded or domain-specific and should be high-level, expressive, and support libraries. Quantum programs can combine quantum gates and classic constructs. Hence, the programming language should support quantum computations, classic computations, and their interaction.

Compiler infrastructure compatible with the programming language of choice will compile the algorithm descriptions into a stream of instructions of the Quantum Instruction Set Architecture (QISA) as represented by the third and fourth level of the stack. Such cross-compilers will run on a classic computer and are responsible for optimizing the executable taking into account various parameters such as QEC code, reversible circuit design, gate decomposition, scheduling of operations, circuit mapping, and routing. Compilers often allow output to intermediate, not directly executable, formats such as QASM-HL [25], OpenQASM [26], f-QASM [27], Quil [28], or cQASM [29].

The QISA exposes the abstracted functionality of the microarchitecture to the compiler and divides the software and hardware layers. Various executable QISAs have been proposed [30], [31], for example eQASM. The microarchitecture executes the instructions of the binary executable and orchestrates the control over the quantum processor at runtime including flow control, feedback based on quantum measurement results, and potentially error correction. The bottom two layers cover the boundary between the classic digital electronics and the analog quantum device which could be based on technologies such as ion-traps, spin qubits, or superconducting qubits.

In the near future, quantum devices with tens to hundreds of qubits will be available. Due to the high error-rates and the limited amount of qubits in such systems it is not reasonable to talk about universal quantum Turing machines [18] yet. Recent insights led to the introduction of NISQ technology as described in [10]. In the near future, a few hundred noisy qubits could accelerate specific useful applications; hence we aim for a quantum accelerator. Modern computer architectures are often heterogeneous, and therefore we envision a heterogeneous computer architecture with a host processor based on existing technologies combined with co-processors such as Tensor Processing Units (TPUs), Graphical Processing

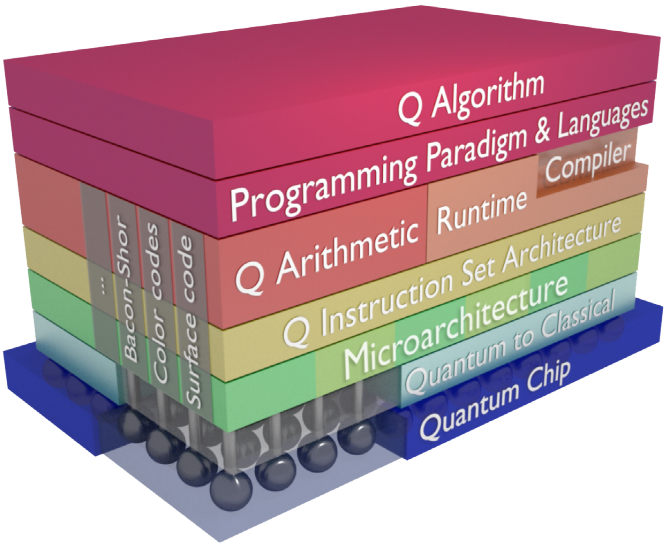

Fig. 1. System stack of a quantum processor from [13].

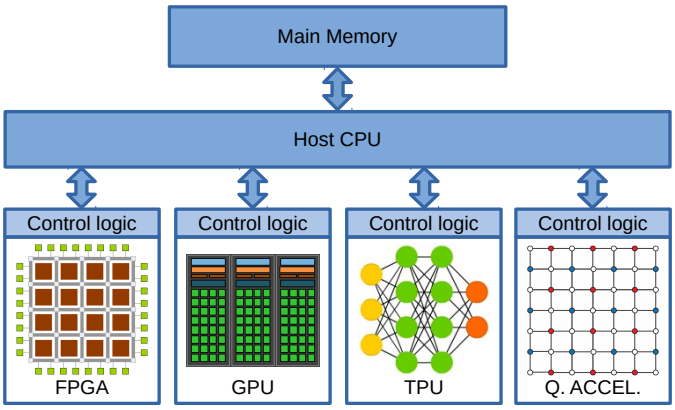

Fig. 2. A heterogeneous computer architecture with quantum accelerator.

Units (GPUs), Field- Programmable Gate Arrays (FPGAs), and, in our case, a quantum co-processor as shown in Figure 2. Specific components of applications can be mapped to the appropriate co-processors while other parts run on the host processor. The application components for the host processor will be written in existing languages as C++ while quantum kernels will exploit the features of the quantum system stack. For such an offload structure, clear boundaries have to be defined between the host program and the quantum kernels with preferably a low amount of communication required to keep the overhead minimal. The compiler infrastructure for such a heterogeneous system will consist of a classic host compiler combined with a quantum compiler as shown in Figure 3. The host program and the quantum kernel will be separately compiled and linked together in the end.

IV. Challenges

Building a NISQ accelerator based on a full system stack requires many challenges to be overcome. Based on our offload programming model we have to define a clear bound-

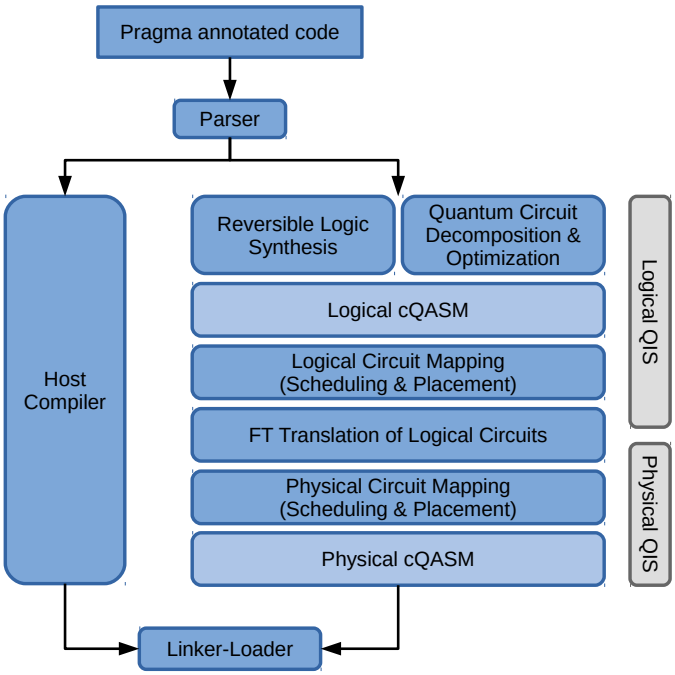

Fig. 3. Compiler infrastructure.

ary between the host program and the quantum kernel. To minimize communication overhead and to make the quantum kernel as self-contained as possible, simple classic arithmetic, flow control, and feedback based on quantum measurement results should be supported by the quantum co-processor. Preferably the quantum kernel can run independently until the final binary measurement results are returned to ensure fast execution times, minimize decoherence, and minimize the impact of communication latency between the host processor and the quantum co-processor. Despite not being NISQ suitable, Shor's factoring algorithm [32] maps very well on a heterogeneous computer architecture since it has clear program components: classic pre-processing, a quantum kernel, and classic post-processing. Proposed implementations of the factoring quantum kernel [33] raise the need for classic arithmetic, flow control, and comprehensive feedback on quantum measurement results as part of the quantum kernel and such features should, therefore, be supported by the quantum co-processor.

On the level of NISQ compilers, most challenges are found in optimizing the execution schedule. Algorithms have to be mapped and routed on the topology of the quantum device while overhead should be minimized. When taking into account device dependent gate error rates and qubit fidelity, compilers can optimize execution schedules to minimize the impact of decoherence and yield better results. Also techniques as randomized compiling [34] will contribute to enabling useful quantum computations.

For NISQ accelerator applications, the QISA should be simple concerning the classic operations and flexible concerning the quantum operations. Simple classic operations allow easy implementations while the flexibility of the quantum operations can be exploited to optimize the gate set for a specific application. Exposing low-level functionality enables the compiler to make optimal use of the hardware capabilities. As a consequence, programmable microcode will be required, and binary portability should not be aimed for during the NISQ era.

When the number of qubits scales up, an increasing number of quantum operations will have to be fetched from memory introducing a challenge on the instruction issue rate, also described as the *quantum operation issue rate problem* [17], [30], [35]. Various classic solutions such as VLIW architectures [30], [35] and SIMD like instructions [30], [36] were proposed to cope with the issue rate challenge, and other existing techniques such as caches and branch prediction can potentially contribute to the solution too. Support for classic arithmetic and flow control by the quantum co-processor will probably play an important role especially in the context of feedback based on quantum measurement results.

When scaling up the number of qubits, the interface between the control electronics and the qubits also becomes more challenging. Cryogenic control electronics [37], [38] have been proposed as a solution. Tighter integration of digital control with the quantum devices will enable further up-scaling.

Finally, major challenges for the qubits in the quantum device will be the up-scaling, coherence, and error rates. Only when the quantum physics community can overcome these, a quantum accelerator can be a meaningful component of a heterogeneous computer architecture.

V. Conclusion

In this paper, we presented a full system stack for a quantum computer architecture and mapped it onto a heterogeneous system architecture with a quantum co-processor. Our system stack defines distinct layers and reaches from algorithms and programming languages down to quantum devices. With NISQ technology and heterogeneous computer architectures in mind, we discussed the opportunities and challenges of mapping hybrid algorithms to the layers of our system stack. At the level of programming languages, clear boundaries have to be defined between the host program and the quantum kernel such that efficient offloading of workload is possible. The compiler will play a key role in transforming quantum algorithms into an efficiently scheduled program for the target quantum processor taking into account all its features and constraints, as already the case in our current experimental setups. Choosing a QISA that is expressive and allows low-level control is essential. At the level of the microarchitecture, the quantum operation issue rate will be the primary challenge when the number of qubits scale. Finally at the level of the quantum device, improving qubit fidelity and scaling up the number of qubits are key issues.

The full-stack approach and tight integration of system layers is the key to the near future development of quantum co-processors. Exposing low-level features to upper layers instead of creating strict abstraction boundaries will create

many opportunities for the compiler and allows faster development of quantum accelerators in a heterogeneous computer architecture.

REFERENCES

[1] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge university press, 2010.

[2] 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, 2015.

[3] A. Córcoles, E. Magesan, S. J. Srinivasan, A. W. Cross, M. Steffen, J. M. Gambetta, and J. M. Chow, "Demonstration of a quantum error detection code using a square lattice of four superconducting qubits," *Nature communications*, vol. 6, 2015.

[4] 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, MAR 5 2015.

[5] P. W. Shor, "Scheme for reducing decoherence in quantum computer memory," *Physical review A*, vol. 52, no. 4, p. R2493, 1995.

[6] 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, no. 3, p. 032324, 2012.

[7] A. Y. Kitaev, "Fault-tolerant quantum computation by anyons," *Annals of Physics*, vol. 303, no. 1, pp. 2–30, 2003.

[8] B. M. Terhal, "Quantum error correction for quantum memories," *Reviews of Modern Physics*, vol. 87, no. 2, pp. 307–346, 2015.

[9] C. Horsman, A. G. Fowler, S. Devitt, and R. Van Meter, "Surface code quantum computing by lattice surgery," *New Journal of Physics*, vol. 14, no. 12, p. 123011, 2012.

[10] J. Preskill, "Quantum computing in the nisq era and beyond," *Arxiv*, vol. 1801.00862, p. 20, 2018.

[11] R. Van Meter and C. Horsman, "A blueprint for building a quantum computer," *Communications of the ACM*, vol. 56, no. 10, pp. 84–93, 2013.

[12] F. T. Chong, D. Franklin, and M. Martonosi, "Programming languages and compiler design for realistic quantum hardware," *Nature*, vol. 549, no. 7671, p. 180, 2017.

[13] X. Fu *et al.*, "A heterogeneous quantum computer architecture," in *Proceedings of the ACM International Conference on Computing Frontiers*. ACM, 2016, pp. 323–330.

[14] G. K. Brennen, D. Song, and C. J. Williams, "Quantum-computer architecture using nonlocal interactions," *Physical Review A*, vol. 67, no. 5, p. 050302, 2003.

[15] C. G. Almudever, L. Lao, X. Fu, N. Khammassi, I. Ashraf, D. Iorga, S. Varsamopoulos, C. Eichler, A. Wallraff, L. Geck *et al.*, "The engineering challenges in quantum computing," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, pp. 836–845.

[16] L. Lao, B. van Wee, I. Ashraf, J. van Someren, N. Khammassi, K. Bertels, and C. Almudever, "Mapping of lattice surgery-based quantum circuits on surface code architectures," *arXiv preprint arXiv:1805.11127*, 2018.

[17] X. Fu, M. Rol, C. Bultink, J. van Someren, N. Khammassi, I. Ashraf, R. Vermeulen, J. De Sterke, W. Vlothuizen, R. Schouten *et al.*, "An experimental microarchitecture for a superconducting quantum processor," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017, pp. 813–825.

[18] D. Deutsch, "Quantum theory, the church–turing principle and the universal quantum computer," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 400, pp. 97–117, 1985.

[19] K. Svore, A. Geller, M. Troyer, J. Azariah, C. Granade, B. Heim, V. Kliuchnikov, M. Mykhailova, A. Paz, and M. Roetteler, "Q#: Enabling scalable quantum computing and development with a high-level dsl," in *Proceedings of the Real World Domain Specific Languages Workshop 2018*. ACM, 2018, p. 7.

[20] 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," Princeton University, Tech. Rep., 2012.

[21] A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron, "Quipper: a scalable quantum programming language," in *ACM SIGPLAN Notices*, vol. 48, no. 6. ACM, 2013, pp. 333–342.

[22] B. mer, "A procedural formalism for quantum computing," Tech. Rep., 1998.

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

[24] K. et al., "Openql1.0: A quantum programming language for quantum accelerators,," *QCA Technical Report*, p. 8, 2018.

[25] 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.

[26] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, "Open quantum assembly language," *arXiv preprint arXiv:1707.03429*, 2017.

[27] S. Liu, X. Wang, L. Zhou, J. Guan, Y. Li, Y. He, R. Duan, and M. Ying, "$q|si\rangle$: A quantum programming environment," in *Symposium on Real-Time and Hybrid Systems*. Springer, 2018, pp. 133–164.

[28] R. S. Smith, M. J. Curtis, and W. J. Zeng, "A practical quantum instruction set architecture," *arXiv:1608.03355*, 2016.

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

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

[31] 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.

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

[33] T. Häner, M. Roetteler, and K. M. Svore, "Factoring using 2n+ 2 qubits with toffoli based modular multiplication," *arXiv preprint arXiv:1611.07995*, 2016.

[34] J. J. Wallman and J. Emerson, "Noise tailoring for scalable quantum computation via randomized compiling," *Physical Review A*, vol. 94, no. 5, p. 052325, 2016.

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

[36] J. Heckey, *et al.*, "Compiler management of communication and parallelism for quantum computation," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2015, pp. 445–456.

[37] 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.

[38] H. Homulle, S. Visser, B. Patra, G. Ferrari, E. Prati, F. Sebastiano, and E. Charbon, "A reconfigurable cryogenic platform for the classical control of quantum processors," *Review of Scientific Instruments*, vol. 88, no. 4, p. 045103, 2017.