CC-Spin: A Microarchitecture design for Control of Spin-Qubit Quantum Accelerator

Amitabh Yadav[1,2,*], Nader Khammassi[2,§], Koen Bertels[2]

[1]Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Berkeley, CA 94720, USA
[2]Quantum & Computer Engineering Laboratory, Delft University of Technology, The Netherlands

ABSTRACT

Quantum Computer Architecture research is directed towards developing a 'Full-Stack' heterogeneous quantum control infrastructure for NISQ-era quantum computing. In this overview paper, we analyze the control infrastructure requirements for Spin-Qubits, propose a Central-Control (CC-Spin) micro-architecture and a parameterized waveform generation methodology to integrate the physical Quantum chip with the OpenQL Quantum Compilation Tool-Chain. We have presented the quantum instruction execution flow via the CC-Spin micro-architecture, and the methodology of generating timing-precise I/Q and DC-waveform using Programmable Arbitrary Waveform Generators (AWG) hardware connected to the FPGA-based Micro-architecture in star-topology. Our work identifies the control signal requirements for Spin-Qubits, proposes an Instruction Set Architecture and presents it's implementation on a Cyclone-V SoC-FPGA hardware for quantum control of NISQ-era quantum devices. We further present a scalable architecture for obtaining waveform generation over multiple waveform generators; and a preliminary design of Micro-architecture for implementing the Quantum Approximate Optimization Algorithm (QAOA).

KEYWORDS

Quantum Computer Architecture, Spin-Qubit Control, Quantum Instruction Set Architecture, Micro-architecture, SoC-FPGA, Waveform Generator

1 INTRODUCTION

Present day Quantum Computers have matured enough to demonstrate early noisy experimental realizations using different qubit technologies on which, low-depth quantum algorithms can be executed. The most popular implementations include Superconducting Transmon Qubits, Spin-qubits in semiconductor Quantum Dot, Trapped-ion Qubits, Nitrogen-Vacancy centers in diamond etc. The concept of utilizing noise-prone qubits i.e. without Quantum Error Correction, with limited-depth quantum circuit definition capable of performing useful calculations is popularly termed as Noisy Intermediate Scale Quantum (NISQ) computing [17].

Given the proposition of NISQ-era technology, the Quantum Processing Unit (QPU) would prove to be elemental in accelerating calculations for numerous applications such as, Simulation of Molecules, Protein Folding, Genome Sequencing, Quantum Field Theory simulations, Quantum Machine Learning and many others. Alongside physical qubit representation, an active research is being pursued in the development of quantum algorithms, quantum-classical high-level programming languages, compilers and simulators. And as we progress towards higher qubit count, we would require a hardware/software co-design framework to address the challenge of integrating the physical qubits to the quantum compilation framework. In such a framework, a quantum processor can be viewed as a *heterogeneous accelerator*, as shown in Figure 1.

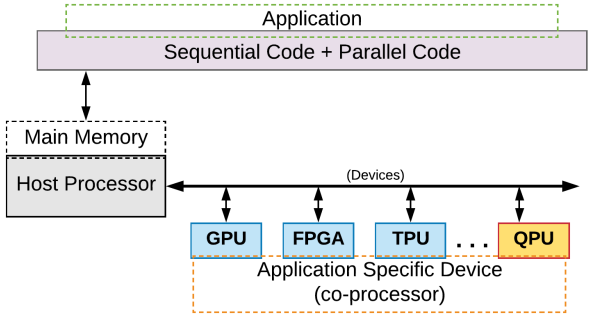

Figure 1: Quantum Processor as Heterogeneous Accelerator

In order to execute quantum algorithms in such a heterogeneous framework, we require a system architecture connecting the high-level quantum circuit description to the low-level electrical pulses operating on the physical qubits. In other words, we require a *Full-Stack* system architecture that establishes abstraction layers that hides the low-level (qubit technology) details from the programmer and helps to integrate the software (OpenQL) to the quantum hardware (control electronics and physical qubits). Figure 2 is such an example of a *Full-stack* system architecture developed by the Quantum Computer Architecture (QCA) Lab at TU Delft [1, 24].

In our quantum accelerator framework, the compilation toolchain consists of the OpenQL compiler [7, 8, 10], that translates high-level quantum programs in C++/Python to a hardware-agnostic Quantum Assembly Language (QASM) description. A low-level compiler then translates the QASM description to a platform-specific Quantum Instruction Set Architecture (QISA) (such as, eQASM [5]). QASM and QISA both contain classical and quantum instructions, however the former is platform-independent whereas, the later is platform-dependent.

*amitabhydv@gmail.com.
§Currently Affiliated to Intel Labs, Intel Corporation, Oregon, USA.

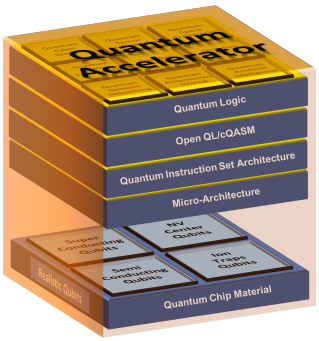

Figure 2: Abstraction layers of Quantum Control

The quantum assembly language such as, cQASM [11] and OpenQASM [3], allows user to specify the complete quantum algorithm - containing both classical computational elements and a *quantum kernel*. The classical processor keeps the control over the execution flow and delegates the execution of the quantum kernel to the Quantum Processing Unit (QPU). The micro-architecture layer constitutes the implementation of QISA, and is responsible for translating the eQASM instructions to analog waveform with precise-timing specifications while being synchronous over multiple channels.

In this paper, we address the architectural challenges for the quantum-classical hardware for controlling the NISQ-era quantum devices and beyond. We take into account two of the promising circuit-model based quantum technologies namely, Superconducting Transmon Qubits and Spin-Qubits in Quantum-Dot. We analyze the control infrastructure requirements and propose a micro-architecture to integrate the physical device with the quantum compilation toolchain. The architecture prototype is developed on a Cyclone-V SoC-FPGA platform in the form of a Central-Controller hardware for Spin-Qubits, called *CC-Spin*. The central controller interfaced to modular AWG units performs the waveform generation such that, the number of DAC channels scale linearly with the number of AWG modules on the central controller. The integration of such a micro-architecture is relevant not only to the theorists and programmers but to the experimentalist as well. The micro-architecture reduces the high-resource consumption, control complexity and cost, and allows faster prototyping and execution of experiments.

The paper is organized as follows: Section 2, presents a background of Circuit-Model Quantum Computing and control requirements for Spin-qubit and Superconducting qubit quantum hardware. Section 3 covers the Quantum/Classical Instruction execution flow in the QCA Full-Stack implementation. In section 4, System Design, we present a comprehensive overview of the CC-Spin Microarchitecture and it's various sub-components. We further discuss the implementation methodology for CC-Spin, and present an architecture to implement Quantum Approximate Optimization Algorithm (QAOA). Section 5 discusses Arbitrary Waveform Generation using FPGA-based AWGs using CC-Spin microarchitecture as the Central Controller. In Section 6, we present the initial results of the micro-architecture as waveform generations applicable to Spin-Qubit achieved on an oscilloscope, thereby validating the applicability of the architecture to a Spin-qubit quantum processor. In the final section, we present our conclusions and future directions to scale and improve upon our Micro-architecture implementation.

2 QUANTUM COMPUTING BACKGROUND

As we scale down to the nanometers (nm) length-scales, the electromagnetic action scales approach Planck's constant ($h = 6.626 \times 10^{-34} J.s$). At that point, circuit design must be based on quantum principles. A quantum computer inherently uses the laws of quantum mechanics notably, *quantum superposition*, *quantum entanglement* and *quantum tunnelling* to solve classically intractable problems.

2.1 Circuit Model for Quantum Computation

The circuit model of quantum computing uses a sequence of quantum gates and measurement operators, to transform and measure the quantum states, respectively. Just like classical logic gates (e.g. AND, OR, NAND, NOT etc.) transform bits, the quantum logic gates are used to perform logical operations on qubits.

In quantum computing, a qubit is a quantum two-state physical system where (usually) the lower energy state is represented as Boolean 0, and the higher energy state is represented as Boolean 1. These states are written in the Dirac notation or *'bra-ket'* notation – $|0\rangle$ and $|1\rangle$. The qubit can also be in a *superposition* of the two states and this is represented as a linear combination of these states, as follows:

$$|\psi\rangle = \alpha_0|0\rangle + \alpha_1|0\rangle$$

The variables α_0 and α_1 are *complex numbers* and are called *probability amplitudes*. According to Born's Rule, the probability of finding the qubit in state $|0\rangle$ is given by $|\alpha_0|^2$ and the probability of finding the qubit in state $|1\rangle$ is given by$|\alpha_1|^2$. The sum of these probabilities must be equal to 1.

$$|\alpha_0|^2 + |\alpha_1|^2 = 1$$

A qubit can be represented as a column vector in a 2-dimensional complex vector space. This means that state $|0\rangle$ and $|1\rangle$ can be represented in the vector notation, as follows:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

The quantum states described are for a two-state system – $|0\rangle$ and $|1\rangle$. These states together form a *basis* and this pair is called the *computational basis states*. If the system is a d-level systems, we call these *qudits*.

2.1.1 Multiple Qubits. With an n-qubit system we can have an arbitrary superposition over 2^n basis-states with normalized complex amplitudes as coefficients and an irrelevant global phase. We can represent this as a vector in an n-dimensional Hilbert space. For example, with 2-qubits, we will have $2^n = 4$ computational basis states — $|00\rangle, |01\rangle, |10\rangle$and$|11\rangle$. This is represented as follows:

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$$

In more general terms, a quantum state is represented as a sum of basis states, $|n_i\rangle$.

$$|\psi\rangle = \Sigma_i \alpha_i |n_i\rangle, \qquad \text{where } \alpha_i \in C^n$$
$$\Sigma_i |\alpha_i|^2 = 1$$

The sets of basis states are always *orthonormal*. This means that these basis states are unit vectors ($|\alpha_0|^2 + |\alpha_1|^2 = 1$) and are orthogonal to each other ($\langle\psi|\phi\rangle = 0$). With n, the dimension of the vector space C^{2n} increases exponentially. Therefore, the space C^d where $d = 2^n$ is also called the *state space* of n qubits. This is because the state space of two quantum states is combined by the *tensor product*.

$$|\psi_1\psi_2\rangle = |\psi_1\rangle|\psi_2\rangle = |\psi_1\rangle \otimes |\psi_2\rangle = \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_d \end{pmatrix} \otimes |\psi_2\rangle = \begin{pmatrix} \alpha_0|\psi_2\rangle \\ \vdots \\ \alpha_d|\psi_2\rangle \end{pmatrix}$$

With multiple qubits, we can also create *Entangled States*. Entangled states do not have a classical equivalent. A joint state of two qubits can be $|\psi_1\rangle \otimes |\psi_2\rangle$. This state can be also called *separable state*, as we can break the components as tensor product of two individual states. Entangled state on the other hand, are states that are *not separable*. Therefore, $|\psi\rangle$ is entangled iff, $|\psi\rangle \neq |\psi_1\rangle \otimes |\psi_2\rangle$, for any $|\psi_1\rangle$ and $|\psi_2\rangle$.

2.1.2 Measurements. Quantum measurements are probabilistic in nature. This also presents the fact that qubits really are fundamentally different. A measurement, or readout operation, in some basis state 'collapses the wave-function' to one of the basis states with some probability. This holds true for multiple qubits in computational basis as well. For example, measuring a general quantum state $|\psi\rangle = \Sigma_i \alpha_i |n_i\rangle$, we would readout the state $|x_i\rangle$ with probability $p_x = |\alpha_x\rangle|^2$.

2.1.3 Quantum Gates. Quantum Gates (also called, Quantum Operator or Unitary) in the circuit model quantum computing are unitary matrices that act on the data encoded in the qubits. A quantum gate acting on n-qubits is a unitary matrix of size $2n \times 2n$. One of the important properties of circuit model is that the quantum gates are reversible. This means that the number of inputs to the quantum gate is equal to the number of outputs.

For further reference, [15] presents a detailed reference for Quantum Information fundamentals.

In practical systems, a qubit is a physical object and is realized through different physical quantum mechanical systems. The Micro-architecture layer is the interface between the high-level programming infrastructure and the low-level signal generation systems. This layer is interfaced to the qubits (at the Physical layer) via quantum-classical interface responsible for code-translation, instruction sequencing and waveform generation. We present the control requirements for two of the prominent technologies for realizing qubits, namely, Spin-Qubit in Semiconductor Quantum Dot and Superconducting Transmon Qubits, based on which is the design of *CC-Spin*.

2.2 Spin-Qubits in Quantum Dots

Spin Qubits, first proposed in 1997 [13], is one of the Solid-state quantum computing methods that uses spins of electrons confined in quantum dots. For quantum dot based qubits, the charge and nuclear spin noises lead to decoherence and gate errors. It has been demonstrated that these noises can be tackled by dynamical decoupling [2] and decoherence free subspaces[14, 16]. Noise can further be reduced and coherence time extended to seconds by growing better oxides and heterostructures[26], and by fabricating the spin qubit in Silicon (^{28}Si), reason being, it's naturally low abundance of nuclear spin isotopes which can be removed by isotopic purification[20]. Using such methodologies have resulted in high gate fidelities (99%) and the demonstration of two-qubit CZ (controlled phase) gate. Thereby having these ingredients, a two-qubit quantum processor can be fabricated in silicon.

At TU Delft, two single-electron spin qubits in natural Si/SiGe double quantum dot (DQD) combining initialization, single- and two-qubit gate rotations and measurement has been demonstrated in 2018 [22]. Spin qubit in quantum dot based processors have demonstrated 99.9% fidelity in single qubit gates [25] and implementation of two-qubit gates [23]. While they have a huge potential for scalability, the current best quantum dot based processors are limited to two-, three- and four-qubit processors.

2.2.1 2-Qubit processor in Si/SiGe Quantum Dot. This section discusses the design for a two-qubit quantum processor in silicon as described in [22]. The schematic for the device is shown in Figure 3.

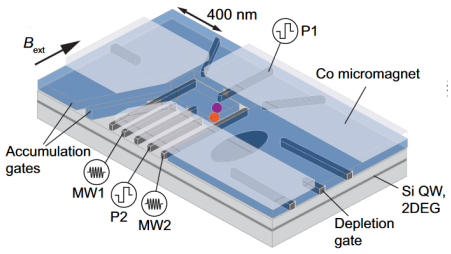

Figure 3: Schematic of Si/SiGe double quantum dot device [22]

Structural description of the Quantum Processor:

(1) Accumulation Gates are used for the formation of two-dimensional electron gas (2DEG) in the quantum well of the Si/SiGe heterostructure.
(2) Applying negative voltages to the Depletion Gates allows formation of double quantum dot (DQD) in the 2DEG. D1 and D2 are the approximate positions of the quantum dot (orange and purple dots, in Figure 3).
(3) The external magnetic field $B_{ext} = 617mT$ causes the Zeeman Split (spin-down $|0\rangle$ and spin-up $|1\rangle$) in single electrons at D1 and D2.
(4) The initialization and read out of Q2 is done by spin-selective tunneling to a reservoir. Whereas, Q1 is initialized by spin-relaxation hotspot and measured via Q2 using controlled rotation (CROT). The respective fidelities are given 1.
(5) Three cobalt micromagnets are present on the top, that provide a magnetic field gradient having a component perpendicular to B_{ext}. This is used for Electric Dipole Spin Resonance (EDSR).

The magnetic field gradient from the cobalt micromagnets also helps separate the qubit frequencies ($f_{Q1} = 18.4GHz$ and $f_{Q2} = 19.7Ghz$) which allows individual control of the qubits. Rabi Frequencies of $f_R = \omega_R/2\pi = 2MHz$ is reported.

Single-Qubit gates are performed by IQ vector modulation of microwave drive signals. X (and Y) gate rotations are $\pi/2$ rotations and X^2 (and Y^2) are π rotations, around $\hat{x}$ (and $\hat{y}$) on this processor platform.

The Two-qubit gate CZ (controlled phase) is also performed similarly. A table summarizing various qubit parameters is shown below, in table 1.

	Q1	**Q2**
Initialization Fidelity	>99%	>99%
Measurement Fidelity	73 %	81 %
T_1	$50ms$	$3.7 \pm 0.5ms$
T_2^*	$1.0 \pm 0.1\mu s$	$0.6 \pm 0.1\mu s$
T_{2Hahn}	$19 \pm 3\mu s$	$7 \pm 1\mu s$
Average Clifford Gate Fidelity	98.8 %	98.0 %

Table 1: Properties of qubits (Q1 and Q2) in (1,1) configuration (1 electron each in D1 and D2). Average Clifford Gate Fidelity is obtained by randomized benchmarking.

2.2.2 Experimental Setup and Control. The experimental setup to control and measure two-qubit silicon-spin is shown in Figure 4. We apply DC voltages to all the gate electrodes using room-temperature DACs via filtered lines. Tektronix 5014C AWG with 1 GHz clock rate is used to apply voltage pulses to plunger gates P1 and P2. These signals from AWGs pass through an RT low-pass filter and attenuators through different stages of the dilution fridge and get added to the DC signals via bias tees mounted on the PCB.

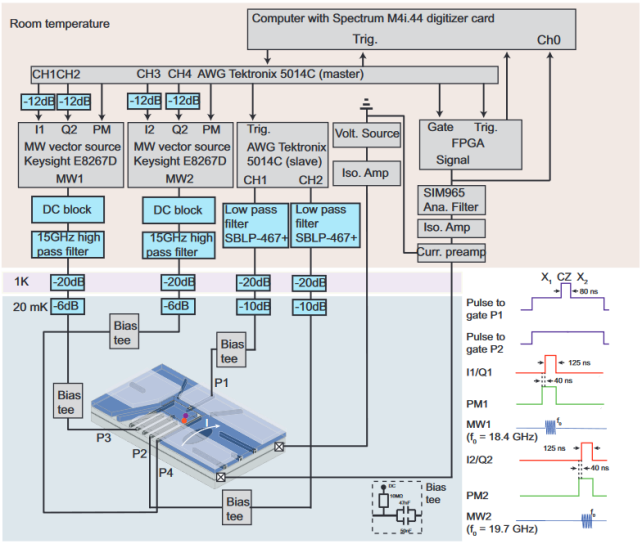

Figure 4: Experimental Setup for Control of Two-Qubit Quantum Processor [22]

To perform EDSR on Q1 and Q2, two Keysight E8267D vector microwave sources, MW1 and MW2, are employed to generate microwave pulses (18 - 20 GHz) and are applied to respective qubits. The microwave signals pass through RT DC blocks, homemade 15 GHz high-pass filters, and attenuators at different stages of the fridge and are added to the DC signals via bias tees mounted on the PCB. I/Q vector modulation is used to control the output of the MW source (phase, frequency, amplitude, duration). Another Tektronix 5041C is used to generate I/Q signals which acts as the master device for the entire setup and provides trigger signals for the other devices. In addition to the vector modulation we employ pulse modulation to give an on/off microwave power output ratio of 120 dB.

While I/Q modulation can be used to output multiple frequencies, the bandwidth of the AWG was not enough to control both qubits with one microwave source due to their large separation in frequency (1.3 GHz). The sensor current, I, is converted to a voltage signal with a home-built preamplifier and an isolation amplifier is used to separate the signal ground with the measurement equipment ground to reduce interference. Following this, a 20 kHz Bessel low-pass filter is applied to the signal using a SIM965 analog filter.

An FPGA is used to analyze the voltage signal during the readout which assigns the trace to be spin-up if the voltage falls below a certain threshold. We can also use a Digitizer card via the PC to measure the voltage signal. The shape of the pulses generated by the AWGs and MW sources during qubit control with the typical timescales is also shown in Figure 4. Square pulses are used to perform the CZ gate and as the input for the I/Q modulation to generate MW pulses. The pulse modulation is turned on 40ns before turning on the I/Q signal due to the time needed for the modulation to switch on.

2.3 Superconducting Qubit Control

Different Superconducting qubits can be made using different encoding methods such as using Flux, Phase or Charge of a superconducting resonator. The *Transmon* is one such superconducting qubit that encodes information in the *Charge* state of the non-linear LC resonator made from Josephson Junctions in a loop and a capacitor in parallel. The information is encoded in the two-lowest energy levels ($|0\rangle$ and $|1\rangle$). A unit Transmon qubit is large (typically, 0.1-1mm^2), is typically operated at below 100mK and has coherence times typically, $18.7\mu s$. The frequency of qubit is externally controllable over several gigahertz using the proximal flux-bias lines (P_F). The Transmon qubit is shown in Figure 5.

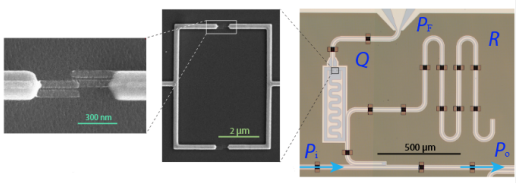

Figure 5: Transmon qubit in planar circuit-QED chip and the zoom in view of the Josephson Junction. Qubit (Q), resonator (R), flux-bias line (P_F), feedline input (P_i), and feedline output (P_o) [6].

2.3.1 Single Qubit Gates. Single qubit gates (such as, X) are performed by applying calibrated microwave pulses at resonance frequency (typically, 4-6GHz and 20ns duration) to the feedline. The I/Q envelopes are created using Arbitrary Waveform Generators. The I/Q mixer performs Single Sideband (SSB) modulation of the carrier microwave signal. Example envelopes for performing X_π and Y_π rotation is shown in Figure 6.

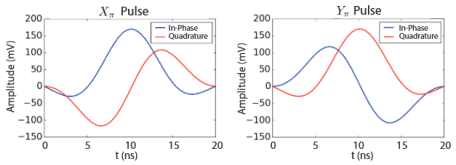

Figure 6: I/Q envelopes for X_π and Y_π rotations

2.3.2 Two-Qubit Gates. Controlled Phase (CZ) and SWAP are the most common gates that can be performed on Transmon qubits that are coupled to a common resonator. Pulses of length ~40ns are applied to the flux-bias lines which causes the frequencies of qubits to come close and interact.

2.3.3 Measurement. The co-planar waveguide resonator is capacitively coupled to the qubit and to the feedline. Frequency shift occurs to the resonator depending on the qubit state which can be measured by sending a 7-8GHz pulse for (300ns - 2μs). The result is then inferred after demodulation, integration and discrimination of the readout signal.

A typical control setup of superconducting qubits, as implemented at TU Delft, is shown in Figure 7. For a complete description of Superconducting Hardware, you may refer [12].

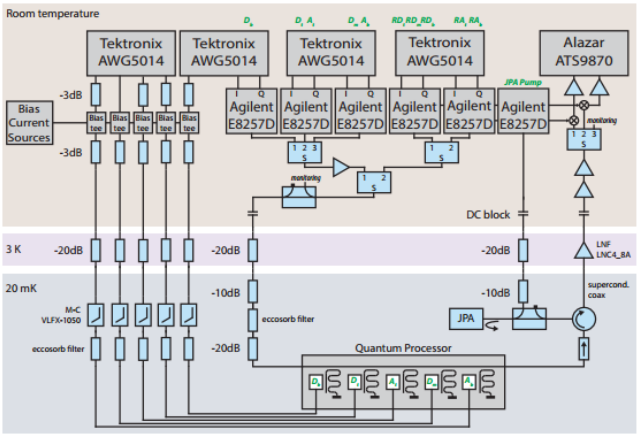

Figure 7: Experimental Setup for Control of Superconducting Quantum Processor [18]

2.4 Requirement Summary for Si-Spin and Superconducting Qubit Control

Usually multiple signals are required to control a Transmon qubit or a single electron qubit in quantum dot. The following are commonly recurring requirements for the control signals:

(1) an independently calibrated and tuned DC gate voltage on every site (typically up to $2V_{pp}$)
(2) independently calibrated and tuned gate voltage pulses on every site (typically, few hundred mV and with sub-nanoseconds rise times)
(3) independently calibrated and tuned microwave magnetic or electric fields at every site (typically -40 to -20 dBm, 1-50 GHz bursts of 10ns to 1μs duration)
(4) a high precision of each of the control signals to achieve error rates comfortably below the 1% accuracy threshold.
(5) Initialization, operations and read-out on timescales must be short compared to the relevant decoherence time.

Typically, precisely timed voltage pulse envelopes in baseband as I/Q signals and microwave sources are essential for single- and two- qubit gates. The measurement requires demodulation at microwave and baseband frequencies. Further, faster (low-latency) classical control feedback is required to perform feed-forward control (branching quantum instructions) on the quantum processor. For example, Quantum Error Correction requires typically 100ns - 1μs measurement time before the qubit decoheres.

A table of signal requirements for various qubit technologies is summarized in Figure 8 [21].

Technology	T_2^*	1-Qubit gate	2-Qubit gate	Qubit read-out	DC-Biasing
Superconducting qubits (Transmons)	2.5 us	~ 6 GHz 20 ns	12ns 40 ns	7-8 GHz, ~ 1 us	flux-bias current
Single-electron spin qubits in a quantum dot	120 us	13-40 GHz ~ 1 us	~ 100 ns		gate voltage
Single-electron spin qubits in a donor system	160 us	30-50 GHz ~ 1 us	~ 100 ns		gate voltage
Singlet-triplet qubit	700 ns	~ 1 ns	~ 1 us		gate voltage
Exchange-only qubit	2.3 us	10 ns ...1 us	Sequence of pulses between different quantum dots		gate voltage
Hybrid qubit	< 10 ns	~ 100 ps	Sequence of pulses between different quantum dots		gate voltage

Figure 8: Summary of Signals used in varous Qubit Technologies [21]

3 EXECUTION FLOW

For the quantum algorithms to be executed on the quantum accelerator, we require a low-level representation of the quantum and classical mixed instructions that the classical control hardware controlling the quantum chip can understand and delegate its execution to the quantum chip. This instruction representation forms the hardware-software interface and is known as the Quantum Instruction Set Architecture (QISA). Development of middle-ware necessary to connect the high-level programming language to efficiently control the physical qubits is therefore elemental to developing a *Full-Stack* Quantum Accelerator (Figure 9).

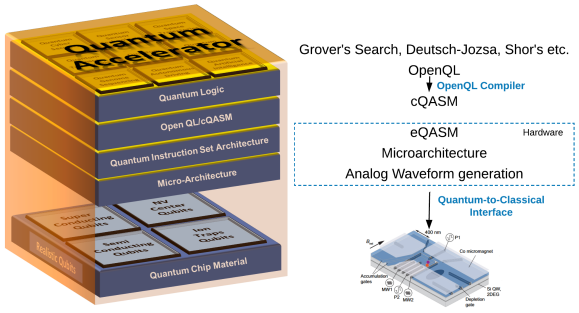

Figure 9: Full-Stack Quantum Instruction Execution Flow

We require two instruction representation formats namely, cQASM and eQASM, because some of the hardware dependent timing constraints can be generated only at run-time. In the Full-Stack implementation, the OpenQL compiler performs Quantum Circuit Optimization, Scheduling, Qubit Mapping and Fault-Tolerant Circuit Implementation of the quantum program. The output of the compiler is the cQASM instructions (common-QASM). cQASM is a quantum and classical mixed code that is a general instruction representation which allows this code to be executable on both, a quantum simulator and real quantum hardware, and is therefore a technology independent representation.

The OpenQL compiler also consists of a low-level compiler that generates eQASM instructions (executable-QASM). An assembler translates the compiled eQASM instructions to the the QISA Binary format for the CC-Spin hardware. eQASM (QISA) is therefore, the sufficient instruction representation to provide to the control hardware, responsible for initializing, controlling and performing readout of the qubits. The ultimate goal of the OpenQL-compiler, therefore, is to produce two kinds of outputs. Based on cQASM, we generate an assembly version that can be executed on a simulator such as, the QX-simulator [9]. If an experimental chip is available, we can translate the cQASM into an executable version, called eQASM, that takes the low-level requirements of the Spin-quantum chip into account. It can also take the requirements of a superconducting quantum chip into account and thus producing a different eQASM version.

The OpenQL Compiler also generates a Control Store File is used to update the Microcode Decode Unit in the CC-Spin Micro-architecture that allows our hardware to decode quantum instructions to target different qubit realization technology. The CC-Spin control hardware then accepts eQASM instructions and performs precise and time-deterministic control-signal generation via the Quantum-to-Classical Interface for Waveform Generation. A detailed description of Quantum Instruction Execution Flow is illustrated in Figure 10.

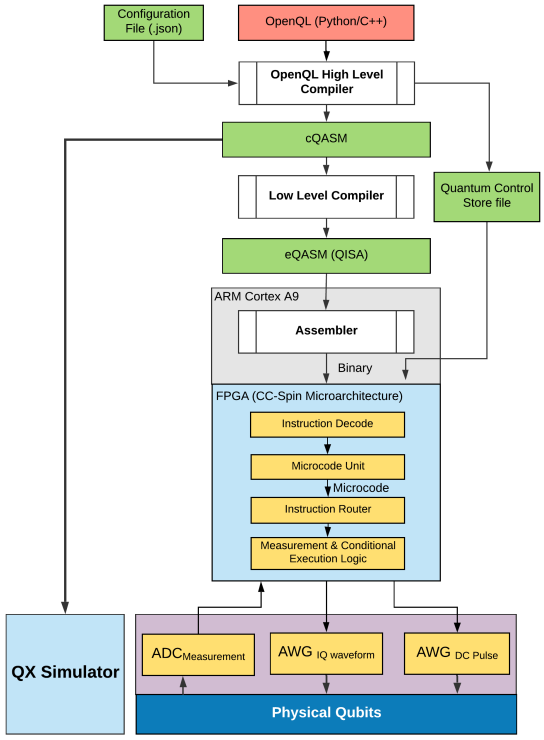

Figure 10: Quantum Instruction Execution Flow

4 SYSTEM DESIGN

The micro-architecture describes a way to leverage digital design techniques such as, Datapath design, pipeline stages, number of registers etc. to implement the defined Quantum Instruction Set Architecture (QISA). In this section we describe the *Central-Controller for Spin Qubit (CC-Spin) Micro-architecture* and it's specifications that is developed and implemented.

The Micro-architecture has a major role to play in the full-stack implementation of the quantum accelerator. The functions executed by the quantum accelerator are:

(1) It creates the abstraction layer between the high-level programming and low-level control implementation on the quantum chip.
(2) It can (ideally) be independent to the technology implementation (such as, superconducting qubit, silicon spin-qubit etc.) of the quantum chip; and abstracts away the low-level hardware details from the user.
(3) Depending on specific requirements of the technology implementation, it can be reconfigured.
(4) Generates high-speed signals (DC, Sinusoidal or Arbitrary) for controlling qubits and implements the timing control for precise and deterministic signal-based control of quantum chip. It is the layer on the full stack that separates the time-deterministic and time-non-deterministic parts of task execution.
(5) Implements fast readouts, and readout-based control (feed-forward control) of the quantum chip.
(6) Implements the classical control logic for quantum error decoding and correction.

A systems view of CC-Spin implementation for qubit control is shown in Figure 11. The flow of instruction is from left to right where the Host PC is responsible for quantum compilation and generation of QISA (eQASM) instructions. The QISA, containing both classical and quantum instructions, is received by the *Master Controller*. The micro-architecture design CC-Spin, containing classical and quantum pipelines, execute unit, microcode unit, instruction router and sequencer is implemented as the Master Controller on a Cyclone-V SoC-FPGA platform. The execute unit is the control unit that directs the program flow for instruction sequencing and execution. The microcode unit is responsible for translating the QISA (binary) instructions (or, macro-instructions) from instruction memory to micro-instructions based on a *quantum control store* and further, to micro-operations. These micro-operations are routed to different SERDES (Serializer-Deserializer) units that transmit the 8b/10b encoded microcode to the slave FPGAs in the Waveform Generation Block.

The output of the micro-architecture is LVDS serial_out data (micro-operations) that is communicated to Waveform Generation Unit. The Waveform generation unit consists of slave FPGAs that implement the microcode buffer and the timing-precise execution unit for time-deterministic instruction execution (waveform generation), as shown in the figure. The Slave FPGA drives the AWG units using serial (SPI for DDS) or parallel (14-bit parallel data for DAC-based) drivers. The AWG units are responsible for I/Q (envelopes) and DC waveform generation.

The implementation of CC-Spin micro-architecture is based on using commercial off-the-shelf (COTS) components for waveform generation. These waveform generation/AWG units can be either: (a) Modular, in-house and custom-developed AWGs with limited memory and DC and I/Q waveform generation capability; (b) a DDS-DAC based solution for waveform generation, such as AD9910 AWG; or, (c) Commercial Programmable Arbitrary Waveform Generators, with multi-channels and more flexibility. While commercial AWGs are more flexible and programmable, they are expensive and bulky, and therefore not a long term solution for large-scale quantum control. In the following sections, we will discuss the implementation and working of each of the functional blocks of CC-Spin Micro-architecture.

4.1 QISA

We have proposed a 32-bit QISA and its implementation is capable of controlling from a few qubits up to 100s of qubits in Spin-based Quantum Processors in NISQ-era quantum computing. As mentioned in Section 2, quantum control comprises of a set of IQ and AC/DC electrical control signal generation. These control signals can be triggered by the micro-architecture (with deterministic timing, Delay of Trigger to Waveform Output being < 100 ns). To specify this triggering operation at the QISA layer, we propose a general instruction representation format that essentially specifies four things: *Instruction Type* (quantum/classical), *Opcode* (the quantum operation), *Quantum Channel Mask* (qubit mapping to channel number) and *Payload* (amplitude/phase/frequency information of (to be) generated waveform). An example of this general instruction format is as shown in figure 12 (b).

To understand this, let's take an example of the pseudo-code shown in figure 12 (a). *Pulse* and *wait* operations, are specified as Opcode. In any practical application, Pulse can be any of the specified quantum operations (such as, X, Y, H, $R_x(\theta)$, $R_x(\theta)$, CNOT [1] etc.) and Wait instruction can be specified in either, Register addressing or Immediate addressing modes. The channel number is specified through the 8-bit quantum channel mask for Channel 1 to 8. For example, the `8'b10011011` asserts the Opcode specified pulse on channels 1, 2, 4, 5 and 8. Finally, the Payload contains different parameters of the waveform such as the DC voltage level to be generated, the phase for IQ signals etc.

As shown in the figure, all channels are initialized with 0V and upon assertion of the pulse operation, the operation stays active on both specified channels for the duration specified by the wait operation (2 units[2]), thus implementing a Single-Instruction Multiple-Qubit control. Next, only the pulse on channel 2 is modified, and 3 wait units. This does not affect channel 1's voltage output and it continues to hold it's value until next pulse operation is asserted viz. at time unit 5. Therefore, using an 8-bit channel mask, we can address up to 8 qubits. The mapping of channel number to a qubit can be fetched through the quantum control store (in the microcode unit) which is generated from the compilation step.

[1] Note that, in spin qubit implementation, CNOT gate requires triggering two channels with precise timing (phase) constraints.
[2] number of clock cycles

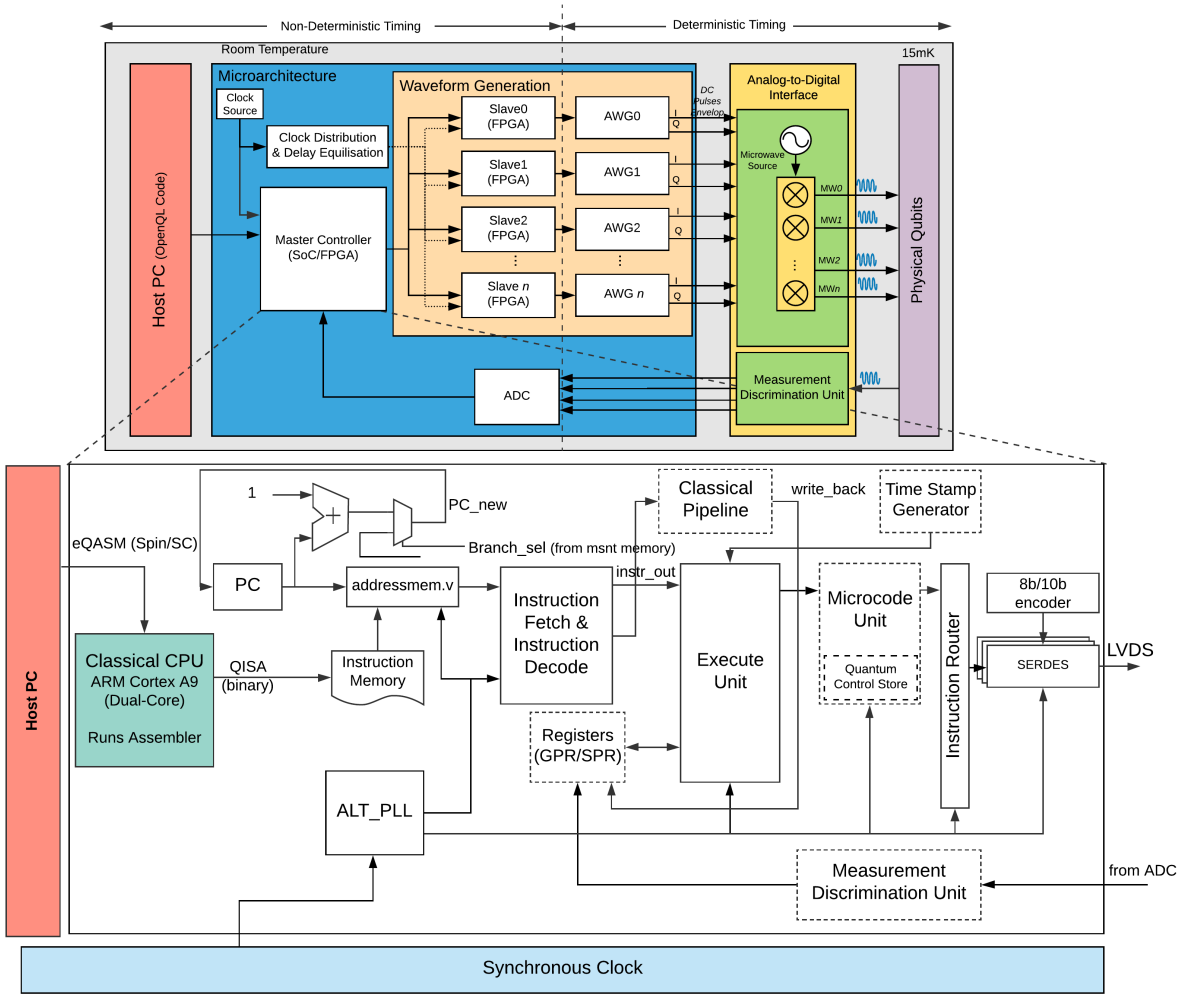

Figure 11: The system view of CC-Spin implementation using Arbitrary Waveform Generators (AWGs) - Modular AWG and/or DDS-DAC

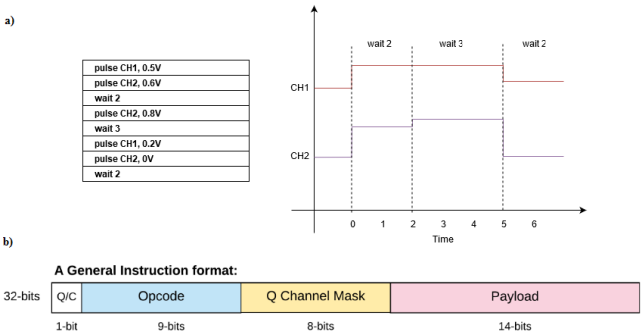

Figure 12: (a) Pulse Generation Timing Control, and (b) A general Instruction Format

Motivated by the previous example of the general QISA representation, we have have proposed the following QISA representation format for addressing a large number of qubits. For simplicity reasons, we will only focus on the quantum instructions and the *wait* operation. The classical operations can be represented in a standard format by setting bit-31 to `0`. Note that, while addressing a large number of qubits simultaneously is essential for NISQ-era quantum control, other factors also need to be considered such as, efficient encoding of waveform parameters in the payload bits, Single-Instruction-Multiple-Qubit (SIMQ) control, high instruction-issue rate etc.

In a variant of the above QISA, here we introduce the channel mask bits that are represented in standard binary number system that point to the channel address in a Look-up Table. The channel mask can then be fetched from the look-up table. The channel address is the input, and the channel mask value at that address is the output.

For example,
`8'b00000001` ⟶ `Mask: 256'b0000...0001`
`8'b00000010` ⟶ `Mask: 256'b0000...0010`
`8'b00000011` ⟶ `Mask: 256'b0000...0100`, until...
`8'b11111111` ⟶ `Mask: 256'b1000...0000`

LUT occupies memory but helps to realize complex digital logic functions fast, as it requires looking up a value from the table which occurs in a single clock cycle. Further, we can also use the LUTs for payload bits to specify different waveform parameters with a higher resolution.

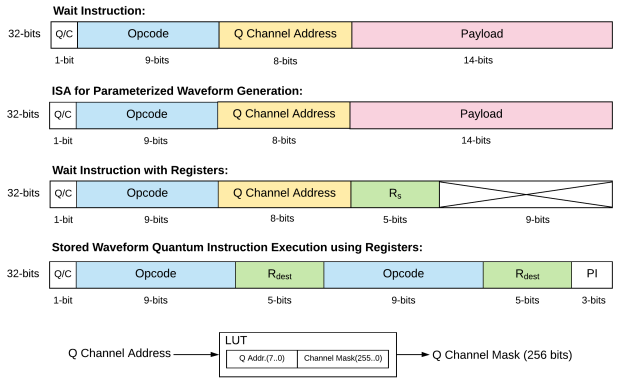

Figure 13: 32-bit CC-Spin QISA - Using LUT for Channel Mask Expansion (10 bit address fetches the channel mask for 2^8 = 256 qubit control channels)

In figure 13, we have shown the CC-Spin QISA instruction format for the wait-instruction (immediate addressing (first) and register addressing (third)) and quantum operation execution (waveform generation based on specified parameters (second), and triggering stored waveform on AWG (fourth). The Q Channel Address is 8-bits which maps to a 2^8 = 256 qubit channels. The disadvantage of this instruction format is that in one clock-cycle only one channel can be addressed. For n-qubits, n instructions need to be fetched in n clock-cycles. However, this occurs in the time-indeterministic side of the quantum pipeline and these operations can be buffered before executing them on the waveform generator.

This instruction format is helpful primarily for addressing large number of qubits (up to 256 qubits) and, specifying waveform parameters in QISA i.e. if an arbitrary waveform generator provides complex waveform shaping capability, the payload bits can be modified accordingly to supply the parameters for the same. The QISA can also supports indirect qubit addressing mechanism using SMIS and SMIT instructions for *Codeword-Triggered Pulse Generation*, such as proposed in [4].

4.2 Quantum Pipeline

In this section, we will discuss the main components of the quantum pipeline. The implementation of the quantum pipeline is based on the general QISA format as described in 12. Using the general format allows us to stick to a simpler pipeline for testing the hardware components and their performance. Moreover, in the HDL-code the pipeline is implemented by setting standard *parameters* for specifying variable elements in the code such as: bus_width, addr_width, memory_depth etc. This allows easy modification of the code for scaling of the same logic to accommodate hardware extensions. Further, all components are described in a *structural logic* which allows easy integration and continuous development on and around the pipeline.

4.2.1 Implementation. A detailed picture of the quantum pipeline is shown in Figure 14. The quantum instruction is specified in the QISA-format and is communicated to the ARM processor on the SoC-FPGA platform through TCP/IP Ethernet port. The executable `.qisa` file contains quantum instructions along with execution hardware timing constraints file. The ARM processor runs an assembler to translate the `.qisa` instructions to binary. The translation here is a direct one-to-one mapping. The assembler then communicates the 32-bit binary instructions `[31..0]` to the instruction memory on the FPGA side. The instruction-fetch stage of the firmware is implemented in the `addressmem.v`. The binary instruction contain both, the classical and quantum instructions, which is identified by the bit-31 of each `[31..0]`instruction where the number 0 represents a classical instruction and the number 1 refers to a quantum instruction. The instruction representation format of classical and quantum instructions are different and therefore requires two different pipelines for execution. An arbiter in the Instruction Decode stage separates the classical code from the quantum code.

The Instruction Decode stage accepts the quantum instructions, decodes the Opcode and Payload and sends the data to the microcode unit. The microcode unit translates the quantum instructions to sets of micro-instructions using the micro-instructions available in the quantum control store. The data in the quantum control store is configurable through an external file that is uploaded along with the `.qisa` file. The micro-instruction are then communicated to the Instruction Router which, based on the qubit-mask routes the data to different SERDES units, each communicating to different waveform generation units via high-speed 3.3V LVDS links.

For the practical implementation, we are using simpler 8-bits parallel data at this step, but depending on the size of micro-instruction definition, we can increase the micro-instruction definition size to 64-bits and even 128-bits. The SERDES converts the incoming parallel data to serial data. Additionally, the SERDES encodes the data in 8b/10b encoding scheme. This is required because in cases when there is a long stream of 0's or 1's, it tends the transmitted signal to look like a DC signal in the communication channel. The 8b/10b encoder ensures that the encoded data stream has at least a certain amount of data transitions. Variant of the same, 64b/66b and 128b/130b can also be used. This is an essential component, as it helps in clock recovery and receiver side alignment of the data stream by providing enough state transitions. This portion of the quantum pipeline is implemented on the Master Controller SoC-FPGA and is not timing-dependent. The time-dependent execution is carried out by the timing execution unit implemented on FPGAs on the modular waveform generation units.

The Waveform Generation units can be implemented using different waveform generation techniques, such as, using AWGs

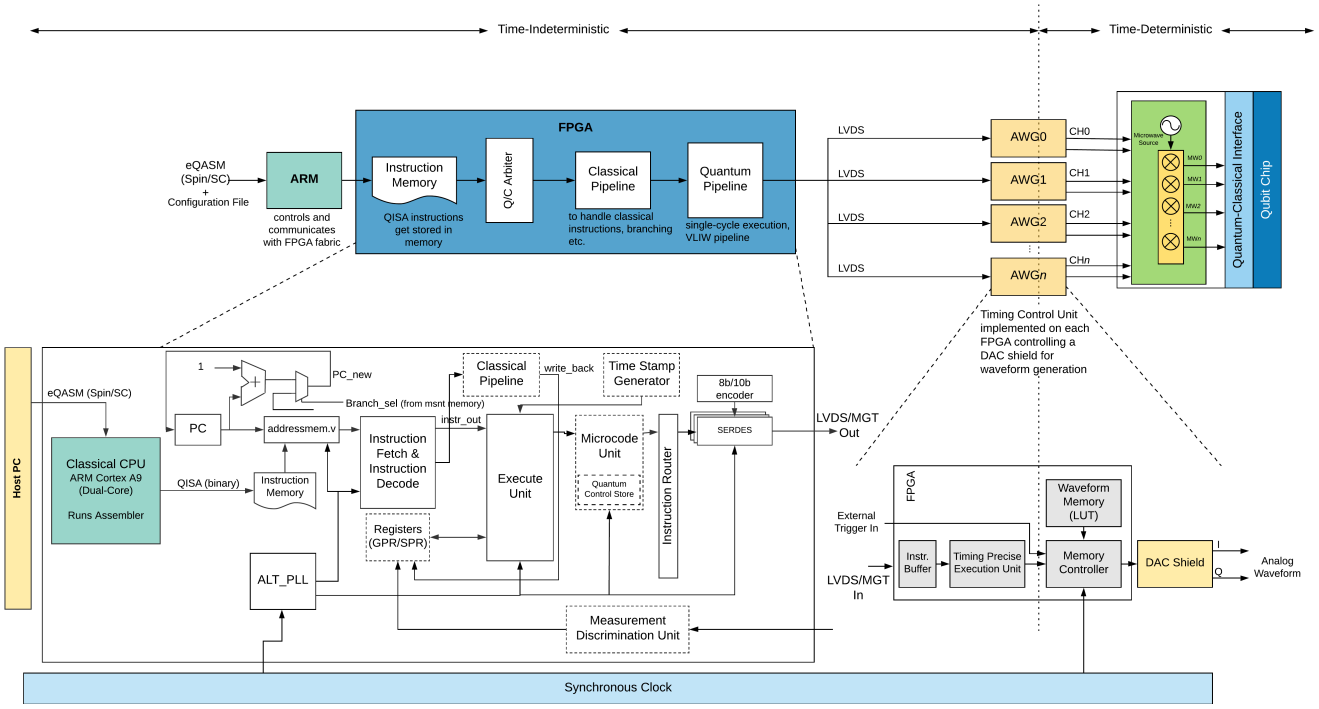

Figure 14: Implementation of Quantum control Pipeline using Arbitrary Waveform Generators.

(lookup table based, Numerically Controlled Oscillator-based etc.), DDS-based etc. The waveform generation units requires waveform parameters to generate waveform and these are provided by the Slave FPGAs [3]. The Slave FPGA's receive the LVDS data_in from the Master Controller SoC-FPGA, deserialize and decode using SERDES and 8b/10b decoder and send the data to Timing Precise Execution Unit (TPEU)[4]. The TPEU contains FIFO buffers, on which the output is triggered after precise number of cycle counts. From this point on, the instruction execution is time-deterministic. The FIFO output is an 8-bit codeword that triggers the memory controller on the AWGs to generate the control waveform. The final output of the waveform generation unit is IQ and DC waveform pulses with precise timing.

While testing the various sub-components of the quantum pipeline, we assumed only the quantum operations (Q/C bit is set to 1) and an arbitrary payload. We do not assume timing information and code-translation to micro-code is disabled. Figure 15 shows the timing diagram of a simulation of the implemented quantum pipeline. In this example, `ch_mask` and `op_code` are decoded by the Instruction Decode unit and only the quantum instructions are routed to the appropriate qubit channel. The size of the channel mask (8) specifies the number of qubits that are addressed simultaneously. For example, at time 50 ns, the mask of channel 1, 4 and 6 are active and the payload is transmitted to those channels. The payload in this example need not necessarily be the payload specified in the QISA. It could be the translated micro-instruction from the microcode unit. The HDL-code also allows the flexibility to change payload sizes and number of channels by simply adjusting the parameter values.

[3]Slave with respect to Master Controller

[4]Timing information of each qubit channel is generated from a Time Stamp Generator implemented on the master controller.

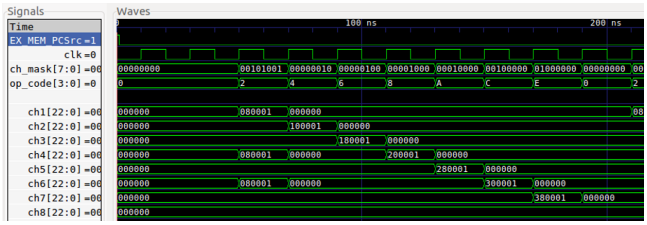

Figure 15: Simulation of Instruction Router of Quantum Pipeline

4.3 Quantum Microcode Unit

The Microcode Unit is an on-hardware abstraction layer over the physical execution units or sequencing units. Compared to microcode, the ISA can be considered as 'macro-instructions'. An efficient microcode takes less memory space, helps in code optimization, minimizes execution time and allows an update mechanism through BIOS/UEFI or the OS. We developed a proof of concept hardware for using LVDS/MGT-based communication and waveform generation for control of qubits. Implementing the Microcode unit is an extension and hence, beyond the scope of the current work. For clarity reasons, we discuss the implementation strategy for Microcode Unit by means of an example.

Based on Instruction Register's contents, the Instruction Decode Unit (IDU) generates/routes control signals to the execution units. IDU can be *Hardwired* or *Micro-coded*.

`Hardwired Decode Unit` generates instruction-specific action sequences which is implemented on the hardware using sequential digital logic such as, Finite State Machines (FSM). Being hardwired, it is faster in terms of speed but it inhibits flexibility to make changes to the design.

In the quantum pipeline, we implement a `Micro-coded Decode Unit`. The microcode unit does not generate the control signals directly, but breaks down the QISA-instructions to micro-instructions. A 'microcode store' implemented as on-chip Read-Only Memory (ROM) is used from where the Micro-instructions are fetched. The Opcode of the decoded QISA instruction are used to generate an initial address, which becomes the entry point into microcode store. Each micro-instruction is followed by a sequence word, that holds the address to the next micro-instruction. Please note that, one QISA instruction can issue multiple micro-instruction. The microcode sequencer carries out the decoding process by successively selecting micro-instructions until decode-complete is indicated by a signal. Conditional microcode branches can also be handled by the microcode unit of the micro-architectures. Pre-computing and storing the control words helps make the micro-code unit design more flexible. Therefore, changes/adding new instructions can be added even at the later stages. This makes the design extension more simplified and feasible because now, to change the decode logic, we only need to adapt the microcode ROM content. The Microcode store in the quantum pipeline is referred to Quantum Control Store. Using this multi-level instruction decoding process, we can update the Quantum Control Store to include the decoding logic micro-instructions. This helps is adapting the the same micro-architecture for multiple qubit technologies.

An example of the Quantum Microcode Unit is from [6]. In the quantum pipeline, the microcode is responsible for translating the QISA-instructions to micro-instructions based on the Quantum Control Store contents. The micro-instructions contains the following components: *wait, pulse, measurement pulse* and *measurement result*. This is then converted to micro-operations with timing labels in the Quantum Micro-instruction Buffer and sent to the timing queues, where appropriate codewords are triggered with precise timing. The same model can be used in the CC-Spin Microcode Unit for instruction decoding. Such a multilevel instruction decoding scheme allows development of a *qubit technology-independent* micro-architecture. The Quantum Control Store content can be modified using an external file that specifies the technology dependent parameters, such as timing information of pulse sequences; and the micro-architecture appropriately decodes the QISA instructions for any specified hardware.

4.4 Qubit Measurement

The quantum pipeline implements the Measurement Discrimination Unit to perform read-out and storage of the quantum states. A typical measurement is performed by sending long DC-pulses to the qubit chip. The current flow is detected and converted to voltage. A typical qubit measurement unit comprises of noise filtering, demodulation, integration and thresholding stages to determine the final qubit state. It is important to perform qubit readout within nanoseconds up to a few microseconds. An efficient readout system is essential for implementing comprehensive feedback control and for performing quantum error correction. The measurement discrimination unit takes the ADC output, performs Integration and Thresholding and record the readout state in the measurement register. From the registers, the execution unit can save the qubit state in external memory such as SDRAM. The value from the measurement register is used to for branching operations, which enables the micro-architecture to perform feed-forward circuit implementations.

4.5 Control for NISQ-era Algorithms

A typical example of NISQ-era algorithm for performing genome sequencing [19] is shown in Figure 16.

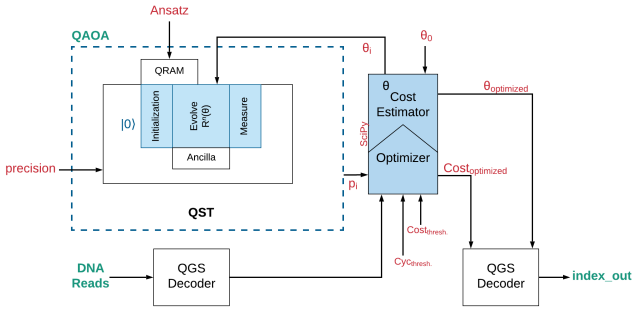

Figure 16: A general Quantum Approximate Optimization Algorithm architecture for Genome Sequencing Application

The quantum implementation of a typical Quantum Approximate Optimization/Variational Quantum Algorithm comprises of a quantum circuit that is initialized with a trial state, the state is evolved on a quantum processor and final state is read-out by efficient measurement. This is represented by QAOA block. The readout after the quantum circuit execution is a histogram of probability of qubit states. This result is passed to matrix multiplication subroutine to generate a real value. This real-value is fed to the classical *optimizer* and the *Cost Estimator* that contains high-level optimization steps. The optimizer generates a new set of parameters that is used to adjust the rotation gate angles (parameters) in the Evolve-step of the quantum QAOA block. Since the optimization step can be composed of SciPy libraries, it is not possible execute them in the same classical pipeline. To support the execution of variational quantum algorithms, the micro-architecture needs to fetch the compiled python binaries at every iteration. The CC-Spin micro-architecture can support the execution of these sub-routines by executing the python-optimization subroutines through the on-board ARM processor running Linux. The FPGA communicates with the ARM processor via AXI bus interface which allows fast ARM-FPGA communication. The optimized quantum circuit can be stored once on the FPGA memory or the on-board SDRAM which allows the user to bypass the openQL compiler for every

iteration of the VQ algorithm. The evolution step of the vq circuit stored in the memory contains rotation gates with arbitrary parameters that is required to be updates after each classical optimization step. The ARM processor running the python compiler can pass the optimized rotation-gate parameters that can be updated in the stored quantum circuit's QISA. This implementation therefore allows run-time tweaking of circuit parameters. Therefore, in order to implement the support for executing VQ algorithms, the CC-Spin micro-architecture needs to support a python compiler implemented on the on-board ARM processor and an AXI bus communication line with the quantum pipeline. The modified version of CC-Spin Micro-architecture to implement QAOA/VQ algorithms is shown in figure 17.

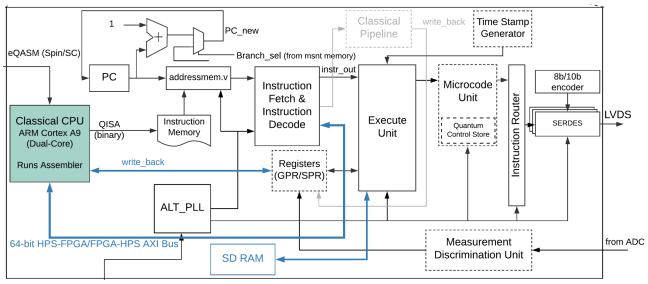

Figure 17: A general microarchitecture for Quantum Approximate Optimization Algorithms

5 WAVEFORM GENERATION

In classical computers, input and output operations occur in the digital domain and expressed as a binary (expressed in Hexadecimal) representation for data and instructions. However qubit control is performed through analog pulses. Arbitrary Waveform Generators/Function Generators are programmed by writing an equation and/or uploading arrays containing waveform points in the AWG waveform memory. These waveform are applied to the qubits with precise timing using external (8-bit) codeword-based triggers. So, for each AWG channel, an external control micro-architecture can trigger $2^8 = 256$ different waveform from the channel's memory. A digital-to-analog converter (DAC) converts the waveform to analog domain, which are modulated with a microwave carrier signal and applied to the qubits. Similarly, the readout signal is obtained by measuring the (very weak) currents from the qubit chip in the analog domain and converted to the digital domain using a Digitizer card.

Operations such as initialization, control and readout, all require efficient waveform generation. The key role of the Arbitrary Waveform Generator is for generating IQ-waveform pulse envelopes and DC Pulses. However, AWGs are not a long-term solution for efficient quantum computation due to limited memory and long wait times for waveform uploads. Furthermore, they limit the number/types of operations that can be performed and do not support active qubit reset or feed-forwarding in quantum computation.

Certain requirement complying with Arbitrary Waveform Generators[5] for quantum processor control are given in the Table 2, below:

Pulse Length	Usually 10 to 100,000 ns
Codeword length	8-bits/AWG IQ pair channel
Delay from Trigger to Waveform Output	within 100 ns
Time between two consecutive waveform triggers on the same AWG channel	20 ns
Number of waveform of length 100 [µs] @ max sample rate that fit in memory	256 waveform IQ pair/AWG channel
Maximum output voltage (DC coupled)	1 V_{pp} 50Ω impedance
Number of channels required	2 for IQ signal pair and 14 for DC pulses (for few qubit experiments)

Table 2: Requirement compliance for Arbitrary Waveform Generators (AWGs)

5.1 Timing Precise Execution Unit

The microarchitecture implementation consists of a Queue-Based-Event-Timing Control Unit which is responsible for accurate and deterministic issue of code word based instructions to the waveform generators. These queues are asynchronous FIFO buffers that are controlled by a *queue manager/timing controller* that uses the instructions in the *timing queue* to activate *multiple event code word buffers*. This implementation usually consists of Mega functions from the vendors and hides away the implementation details of the the timing queues. This hinders further development in ASIC, or porting of platforms during microarchitecture design. The alternative is to design asynchronous FIFOs.

Based on the trigger from the queue manager, the FIFO addressed by the channel mask sends `data_out`. The trigger is generated by counting number of clock cycles specified in the micro-instruction.

5.2 Using Programmable AWG

Digital to Analog Converters (DAC) are devices that convert digital signals to analog signals. In this implementation, we use ADA (analog-to-digital-to-analog) converter available from Terasic. The board contains 2-channel, 14-Bit, 125 MSPS Digital-to-Analog Converter (AD9767). The DAC can be operated with two separate data ports, or with a single interleaved high speed port, which is specified through a mode select input.

The programmable AWG is basically an FPGA feeding 14-bit digital data to a DAC. The 14-bit resolution determines the output waveform. The FPGA receives micro-operations from the Master

[5]Phase Noise, Jitter, and Spurious-Free Dynamic Range (SFDR) are the performance metrics for waveform generators.

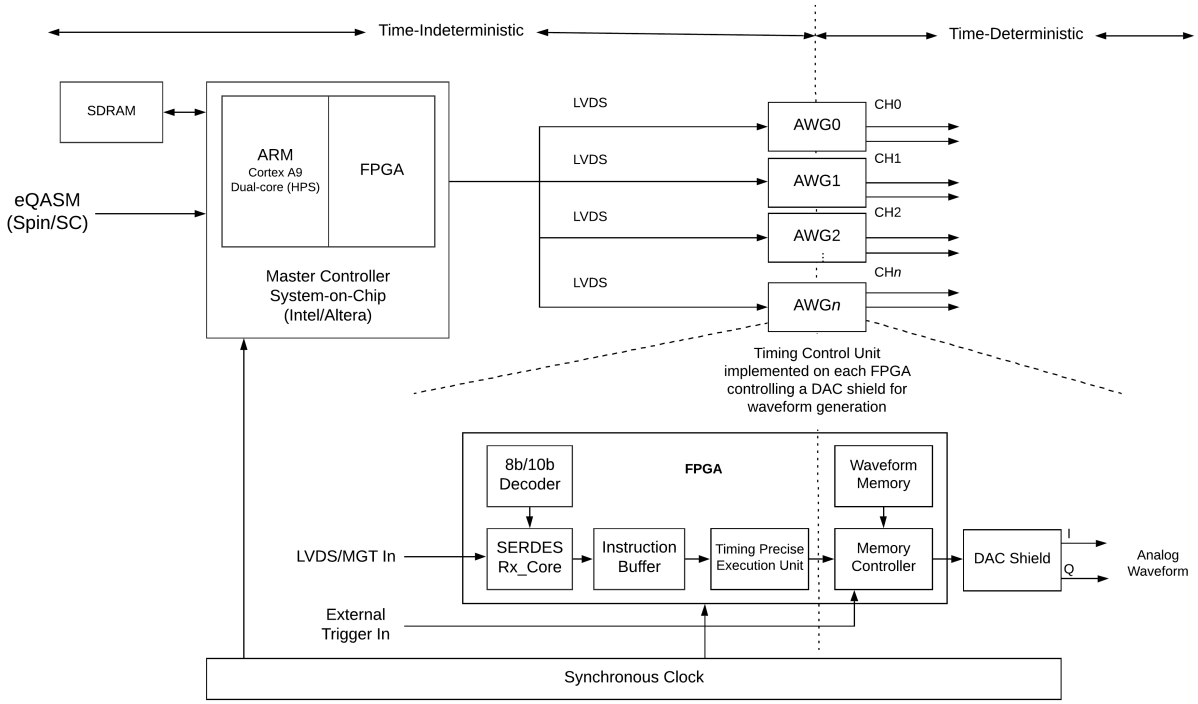

Figure 18: Waveform Generation using a True AWG Architecture

controller through high-speed LVDS links. The SERDES core deserializes the input data stream and stores the instructions in an Instruction FIFO buffer. Using the instruction specifications, the Timing Control Unit issues codeword specifying memory locations in the waveform memory which generates the output waveform. We implemented the HDL design for generating simple sinusoidal waveform with the DAC based AWG. This implementation is given in section 6.1.

The architecture of the Slave FPGA driving a DAC based AWG is shown in figure 18. The slave FPGA buffers the operations in timing queues and executes triggers the memory controller for waveform generation.

5.3 Using DDS based AWG Architecture

Direct digital synthesis (DDS) hardware uses a method of producing an analog waveform (Square, Triangular, and sinusoidal waveform) by generating a digital time-varying signal and using a digital-to-analog converter to get the analog waveform. DDS allows fast switching between output frequencies, fine frequency resolution and allows a broad frequency spectrum for operation, as all internal operations are digital. The DDS are low-power, low-cost and can generate programmable analog output waveform with high resolution and accuracy. It also has low-phase-noise and variable frequency generation capability (<1 Hz up to 400 MHz, on a 1-GHz clock) for waveform generation. We used the AD9910 DDS-DAC based waveform generation which is programmable via a high-speed serial peripheral-interface (SPI) protocol that requires to be developed separately for the DDS-chip. The timing diagrams and instruction definition for designing the SPI interface is available on the AD9910 data-sheet.

The main components of a DDS device are a *phase accumulator*, a means of *phase-to-amplitude conversion* (often a sine look-up table), and a DAC. The DDS internal circuitry is shown in figure 19. In a DDS, the frequency depends on the reference clock and the tuning word.

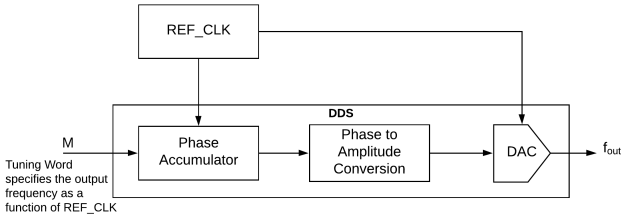

Figure 19: DDS components and signal flow

The phase accumulator is a modulo-M counter. Every time it receives a clock pulse, it increments its stored number. The binary-coded input word (M) determines the magnitude of the incremented value. This word forms the phase step-size between reference-clock updates i.e. effectively sets how many points to skip around the 'phase wheel'. With larger jump size, the phase accumulator overflows faster and completes its equivalent of the sine-wave cycle. The resolution of the phase accumulator(n) (the tuning resolution of the DDS) determines the number of discrete phase points in the phase wheel.

A phase-to-amplitude look-up table is used to convert the phase-accumulator's instantaneous output value into the sine-wave amplitude information that is sent to the (10-bit) DAC. Since the sin wave is symmetrical, the DDS architecture uses only a quarter-cycle of data from the phase accumulator to generate a complete sin wave. The remaining data is generated by reading forward then back through the phase-to-amplitude look-up table.

Because a DDS is digitally programmable, the phase and frequency of a waveform can be easily adjusted without the need to change the external components that would normally need to be changed when using traditional analog-programmed waveform generators. This offers an advantage in using a DDS. Further, DDS chip offers advantage for controlling quantum systems as it allows simple real-time adjustments in frequency to locate resonant frequencies or compensate for temperature drift.

5.3.1 IQ Waveform Generation using DDS-DAC. The chosen DDS solution for waveform generation is the AD9910 1 GSPS, 14-bit, 3.3V CMOS Direct Digital Synthesizer that can generate sinusoidal frequencies up to 400MHz using the 3 control parameters — amplitude, phase and frequency. Note that the REF_CLK frequencies must be 60-1000 MHz (with Clock Multiplier Disabled) and 3.2 - 60 MHz (Clock Multiplier Enabled); and Voltage levels must be 50 - 1000 mV (Single-Ended) and 100 - 2000 mV (LVDS).

We can use two DDS units operating on the same REF_CLK to output the I/Q signals whose phase relationship can then be controlled. To do this, we use both AD9910 with the same REF_CLK and the same Reset that updates both. This setup to do I/Q waveform generation is shown in figure 20. Note that the reset must be asserted after power-up and prior to transferring any data to the DDS. This sets the DDS output to a known phase, which serves as the common reference point that allows synchronization of multiple DDS devices. When new data is sent simultaneously to multiple DDS units, a coherent phase relationship can be maintained, and their relative phase offset can be predictably shifted by means of the phase-offset register.

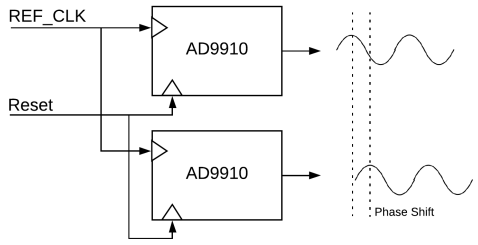

Figure 20: Multiple synchronous DDS for IQ signal generation

In the micro-architecture developed around the Direct Digital Synthesis (DDS) based system, one DDS unit is controlled by a slave FPGA. The firmware for the FPGA is the same as that of DAC based programmable AWG, except an SPI Master interface or Parallel I/O port is used to program the DDS-Slave device. The parameters for programming the DDS through SPI interface is extensive, and requires to be designed from scratch in VHDL/Verilog following the design parameters as provided in the AD9910 data-sheet. The DDS unit can generate control waveform in real-time using input from the Timing Precise Execution Unit (TPEU). Therefore, we can alternately use a simpler instruction set such as given in figure 12 to implement a simpler TPEU with simple registers. Besides this, the setup also uses a very low-noise AD9517 PLL clock synthesizer with an integrated VCO, clock dividers, and 14 outputs. Beside the DDS, we also have multi-channel DAC and ADC connected to the master controller. The DAC facilitates simple DC waveform generation and ADC converts the analog measurement of qubit states to digital data.

5.4 Multi-Device Synchronization

The AD9910 DDS-DAC chip facilitates synchronizing multiple channels for waveform generation. The setup implements multiple slave FPGAs driving multiple DDS units through the SPI-master controller interface. All FPGAs work synchronously with SYN_CLK of the DDS units, while DDS hardware allows synchronizing multiple DDS devices using the clock distribution and delay equilization board AD9517. The DDS chip has a sync generator block that makes one of the DDS unit as the master and remaining as the slaves. The master produces an LVDS SYN_OUT clock ($f_{SYN\_OUT} = f_{SYSCLK}/16$) which synchronizes the remaining boards with the rising or falling edge of $SYSCLK$. The detailed circuit for the setup is shown in figure 21.

6 RESULTS OF WAVEFORM GENERATION FOR SPIN QUBITS

In this section, we present the testing methodology of the sub-components of CC-Spin Micro-architecture. An experimental realization of the CC-Spin control box is shown in Figure 22.

The microarchitecture runs on a Master FPGA controller that issues commands for signal generation to different Waveform generators (DDS/DAC or Modular AWGs). The Master FPGA controller also hosts the Measurement Acquisition Unit that performs filtering, demodulation, integration and thresholding. The Master Controller Microarchitecture firmware is implemented on Enclustra Mercury+ SA2 SoC-FPGA, and the FPGA-controlled DAC AWG is implemented on a Terasic DE10 Nano FPGA with AD9744 4-Bit, 210 MSPS DAC as a part of CBoxV3 hardware. While the complete implementation of the microarchitecture is complex, we implemented specific sub-components of the microarchitecture and obtained their results. Therefore, in the following sections, we have presented results from hardware testing of the two Waveform generation viz. DDS-based and DAC-based, also indicated in the figure. The resource utilization for a simplified quantum pipeline for the general QISA format and TEPU is shown in table 3.

6.1 Implementation of FPGA controlled DAC-based AWG

Digital to Analog Converters (DAC) are devices that convert digital signals to analog signals. The RTL design uses Numerically Controlled Oscillators with a 'Multiplier-based Architecture' to

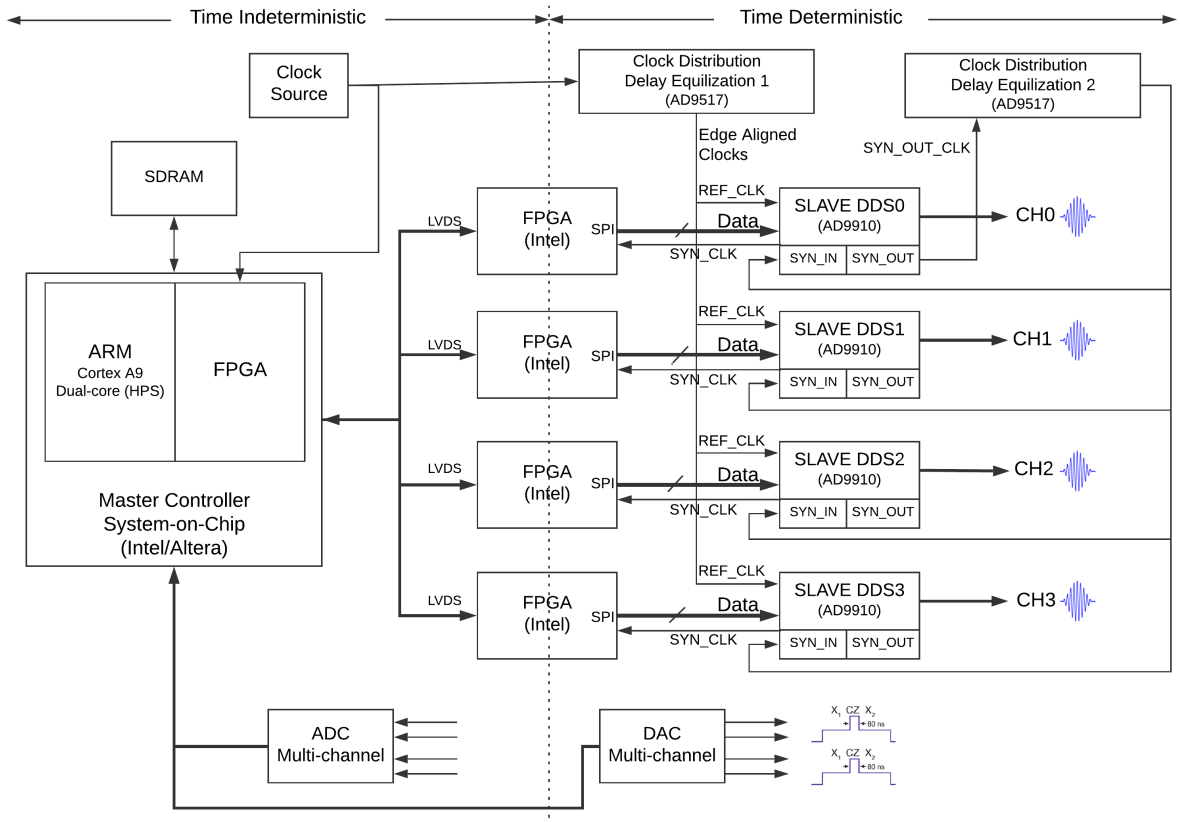

Figure 21: Implementation of Synchronized Waveform generation using AD9910 DDS-DAC

Resource Utilization	
Logic Utilization (in ALMs)	628/41,910 (1%)
Total Block Memory Bits	163,328/5,662,720 (3%)
Total RAM Blocks	18/553(3%)
Total PLLs	1/15(7%)

Table 3: Resource Utilization for Quantum Pipeline and TEPU on FPGA

generate SIN and COS values [6]. The multiplier architecture implements a multiplier circuit in the logic elements of the FPGA and helps reduce memory usage. Alternately, we can also use 'Large ROM Architecture' with requires a larger internal memory but performs best for high speed applications. Our experience shows that large ROM architectures give the highest spectral purity and utilizes least number of logic elements on the FPGA.

Except CORDIC and multi-cycle multiplier-based architectures, all NCO-architectures output a sample every clock cycle. After asserting clock-enable, the oscillator outputs the sinusoidal samples at one sample per clock-cycle rate, following an initial latency of L-clock cycles. The exact value of L varies across architectures and parameters. After the clock enable is asserted, the oscillator outputs the sinusoidal samples at a rate of one sample for every two clock-cycles, following an initial latency of L clock cycles.

6.1.1 Using SignalTap. We used the Quartus SignalTap II Embedded Logic Analyzer software which allows us to capture signals from internal RTL-design nodes from the FPGA in real-time. The tool is very useful to in debugging the design logic running in real-time on the FPGA hardware. The tool is available as a mega-function and can be integrated in the design schematic. We use the .stp file to define the parameters we want to fetch. We setup the parameters by specifying their name and sample depth that we wish to 'tap'. The design is then compiled[7] and is uploaded to the hardware via the USB Blaster Interface. By running analysis on the interface, we can get the data. The real-time data obtained from ADC and DAC implementation is shown in Figure 23.

Furthermore, we can record the data in a Signal Tap-II list file and get the Spectral plot by post-processing in MATLAB. In MATLAB, we take the DAC-input samples from the list file, perform normalization (subtract DC components), perform FFT transform and plot the data. The Normalized Spectral Plot of DAC performance is shown in Figure 25. The main output frequency is at 20MHz and the 1st harmonic is at 40 MHz.

[6]The architecture can be Large ROM-based, Small ROM-based or CORDIC algorithm-based. All are used to generate SIN and COS values.

[7]Using Signal Tap incurs large area consumption in terms of Number of Logic Elements (LEs) occupied

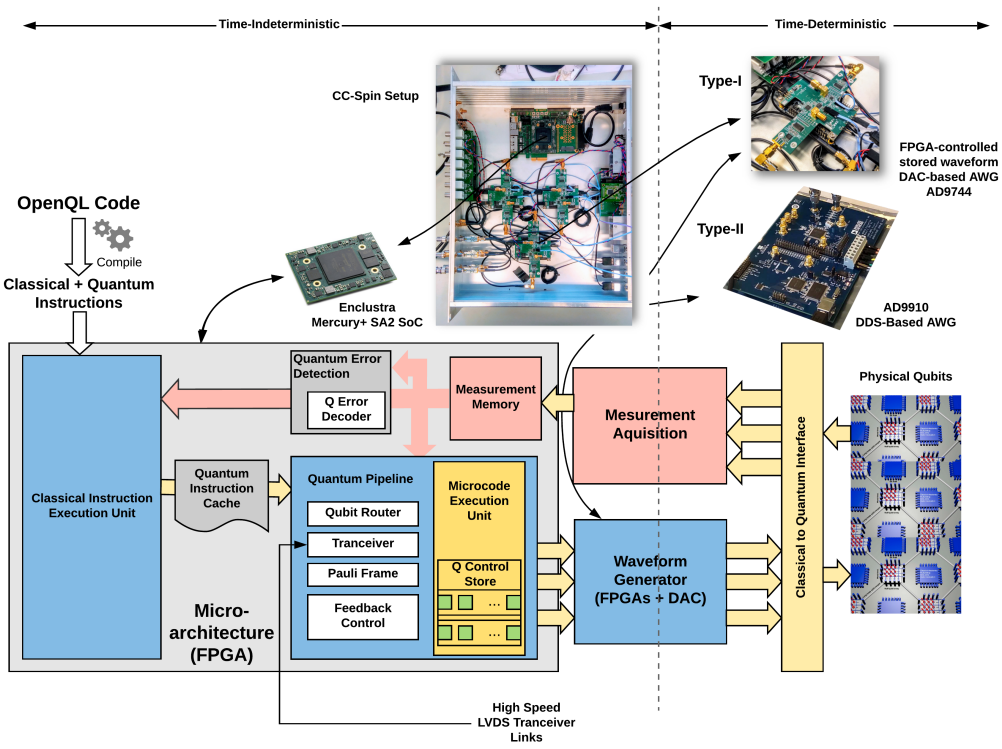

Figure 22: Schematic of Experimental Setup

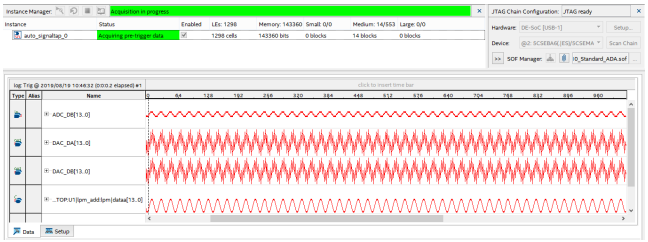

Figure 23: Real-time ADC and DAC waveform obtained through Signal Tap

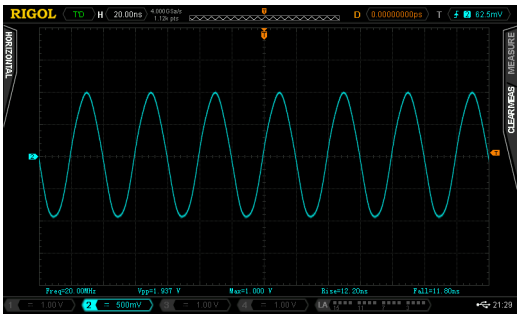

Figure 24: 20MHz SIN wave generated using DAC based AWG

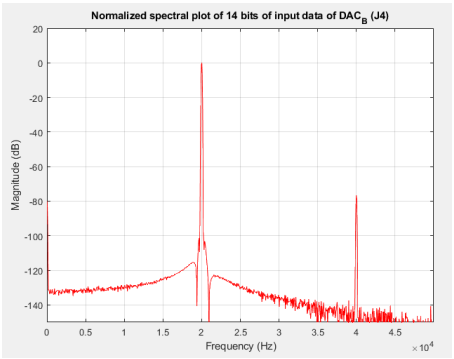

Figure 25: Normalized Spectral Plot of DAC

6.1.2 Hardware Implementation. The RTL-Netlist schematic of the DAC-based Waveform Generator is shown in Figure 26. The NCO generates SIN and COS waves based on the phase_increment value and an lpm_add_sub Megafunction adds the components together to produce a modulated wave. Using this circuit, we are producing a SIN-wave and 90° phase shifted wave (COS wave).

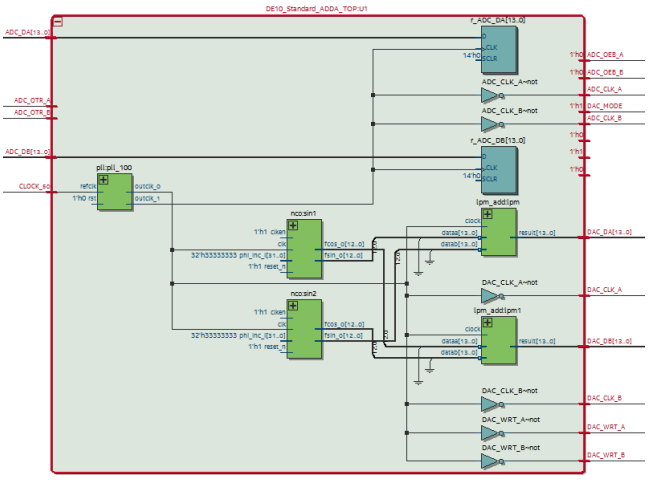

Figure 26: RTL Design of DAC-ADC implementation on Terasic DE10 Standard SoC-FPGA

The analog signal out is shown in figure 27. We further changed the values in the phase_increment to obtain the shape of a Gaussian Waveform (as shown in Figure 28). These can be further modified to obtain DRAG pulses.

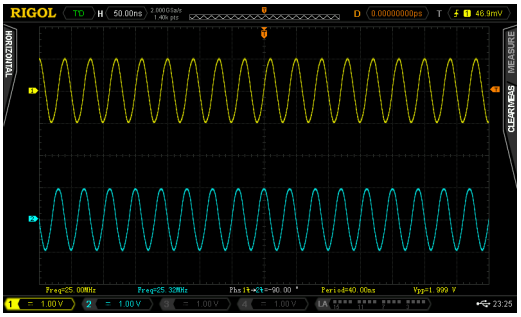

Figure 27: IQ SIN wave 25MHz, V_pp=2V from Terasic Cyclone V with DAC board

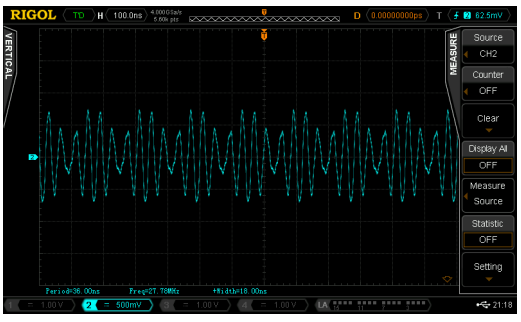

Figure 28: Gaussian waveform from DAC obtained by mixing IQ waveform followed by Analog conversion

6.2 Implementation of DDS-based AWG

In this section, we will discuss the demonstration of AD9910 Direct Digital Synthesis unit using the Evaluation Board. There are two ways to program the DDS board: first, by using the USB interface, and second, by using the parallel port. The vendor, Analog Devices, provides a GUI-based evaluation software to facilitate easy communication through the USB interface. We program the board's internal control registers via either of the interfaces. The DDC-DAC board provides access to three parameters for controlling the DDS — frequency, amplitude and phase. We can tune the frequency using the 32-bit accumulator and at a sample-rate of 1GSPS, the tuning resolution is approximately, 0.23 Hz. The board also allows for fast phase and amplitude switching; and supports sinusoidal waveform outputs at frequencies up to 400 MHz.

We can operate the AD9910 in one of the four modes: Single-tone, RAM modulation, Digital ramp modulation, and Parallel data port modulation.

In single-tone mode, the serial I/O port is used to program the internal registers. The registers provide the signal parameters to the DDS. Independent registers containing the DDS signal parameters are referred as 'profile'. There are 8 profile registers and are independently accessible using the external pin `PROFILE[2:0]`. Change in profile pin state is reflected with the next rising edge of SYNC_CLK and DDS output gets updated with next profile's parameter.

In the RAM-modulation mode, signal parameters are fetched from the internal RAM and played when triggered. The RAM has a depth of 1024 samples and consists of 32-bit words. Using the RAM-mode, we can generate arbitrary time-dependent waveform. The data fetch rate from the RAM is controlled through a programmable timer (value in μs). Further, programming the 8 RAM profile registers allows selecting specific DDS signal parameters that act as the destination for RAM samples.

In digital ramp modulation mode, a Digital Ramp Generator (DRG) provides the signal parameters; and the serial I/O port supplied the parameters for ramp generation. The digitally generated ramp has an output resolution of 32-bits that can be programmed to represent the frequency (32-bits), amplitude (14-bits), or phase (16-bits).

In parallel-data port modulation mode, the DDS signal parameters are driven into the 18-bit parallel port. The 16 MSB bits is the data-word (`D[15:0]`) represented in unsigned binary and 2 LSBs is the 2-bit destination word (`F[1:0]`). The 2 LSBs defines how the data-word is applied to the DDS signal parameters.

6.2.1 Functioning of DDS-Core. The DDS-core generates a SIN/COS-reference signal based on the value of the Control Function Register (CFR1). The frequency control, phase offset, and amplitude control takes the parameters - frequency, amplitude and phase - as input. This is shown in Figure 29

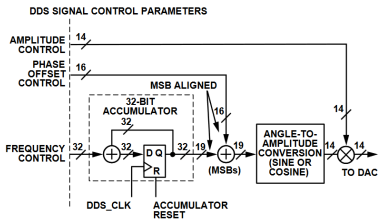

Figure 29: AD9910 DDS Core Functional Block Diagram (AD9910 Datasheet)

The output frequency is a function of the 32-bit *Frequency Tuning Word (FTW)* and System Clock Frequency f_{SYSCLK}. The relationship is given as follows:

$$f_{OUT} = (\frac{FTW}{2^{32}})$$

For calculating FTW of desired f_{out}, we use the following relation[8]:

$$FTW = round(2^{32}(\frac{f_{OUT}}{f_{SYSCLK}}))$$

The relative phase of DDS is controlled by the 16-bit Phase Offset Word (POW). The DDS-core applies this before the Amplitude-to-Angle conversion step. The relative amplitude can also be scaled by the 14-bit Amplitude Scale Factor (ASF). This is done at the last step. The final output of the DDS core is then sent to the 14-bit DAC.

6.2.2 DDS Serial Programming using Slave FPGAs. The AD9910 can be programmed using the Serial Peripheral Interface (SPI) protocol. The SPI-driver is used to control the DDS Evaluation Board for arbitrary waveform generation. The bit representations and instruction used for operating the DDS-DAC can be referred from the AD9910 Data-Sheet.

6.2.3 Implementation. We used the Evaluation Board for simple sinusoidal waveform generation. The evaluation board has two different connectors for power supply, named TB1 and TB2. TB1 supplies power to the digital I/O interface, the digital core and USB circuit, whereas TB2 supplies power to the DAC and the clock_input circuit.

The `clk_in` can be provided in one of the 3 ways: either provide a high frequency input signal through the `SMA` port. The second option allows the user to connect using a lower input reference frequency, enabling the clock multiplier, and connects through J1. The third option allows the user to connect to the on-board crystal oscillator.

The detailed instructions for using the AD9910 Evaluation Board can be accessed from the Evaluation Board User Guide. The obtained experimental waveform in both RAM mode and Single Tone Mode are shown in Figure 30 and 31.

[8]round to nearest integer

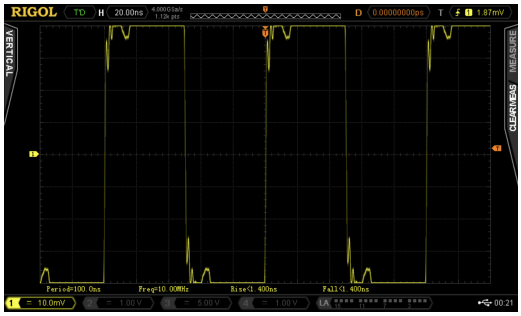

Figure 30: Square wave 10MHz, when operated in RAM mode from AD9910 DDS-DAC

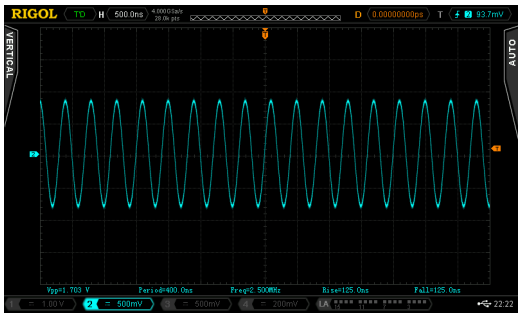

Figure 31: SIN wave 5.0MHz from AD9910 DDS-DAC in single-tone mode.

7 CONCLUSION

In this section, we summarize our research contributions made with this work, present the limitations of the defined architecture and how we can extend the digital design to overcome them by implementing next steps. Our work is intended to explore the design space for integrating the quantum control hardware to the OpenQL compiler tool-chain.

7.1 CC-Spin Micro-architecture Implementation

Micro-architecture development is an iterative process comprising of the following three parts:

(1) **Design Partition:** Break down the idea into easier-to-understand functionality (Divide-and-Conquer).
(2) **Datapath Design:** Identify the components required to design the functionality such as FIFOs, Multiplexers, Adders, Multipliers, ECC, CRC, Encoding/decoding, scrambling/encryption, Internal data-path size (32-/64-/128-bit) etc.
(3) **Control Functions:** Design of one/multiple state-machines. Encoded State-Machines vs One-hot state machine. Full Handshake vs half-handshake in data transfer between clock crossing domains etc.

We approached the challenge with the same philosophy. While designing the QISA and Quantum Pipeline, we focus on the design's

simplicity and modularity. We need the design to be able to meet all the functionality criteria and should be easy to understand, debug, port and extend (scale). To design the micro-architecture layer, we need to understand the requirements for the lower layers of the Full-Stack. We establish the requirement specifications for the Analog-Digital-Interface (the waveform generation layer) and design a QISA format for representing the quantum operations expressed in Quantum Assembly Language. The QISA directly affects the micro-architecture implementation, therefore an efficient QISA design is important.

We have designed a quantum pipeline executing on a master controller FPGA (comprising of the Micro-coded Instruction Decoder and Router) and communicating via high-speed links to the Slave FPGAs (with Timing Precise Execution Unit which is the waveform generation Trigger control). We implemented and tested parts of the pipeline for correct functionality while taking care of hardware errors, such as timing mismatch in parallel lines, Metastability issues, etc. and presented ways to synchronize the Waveform Generation across the distributed waveform generators via SYNC_CLK mechanism. Special care is taken while writing the HDL code for all the components of the quantum pipeline - the pipeline is implemented by setting standard *parameters* for specifying variable elements in the code such as: bus_width, addr_width, memory_depth etc. This allows easy modification of the code for scaling of the same logic to accommodate hardware extensions. Further, all components are described in a *structural logic* which allows easy integration and continuous development on and around the pipeline.

The main contributions of our research are as follows:

(1) We have proposed a QISA design format for specifying quantum and classical instruction execution on hardware. The QISA format can address up to 256 qubits. An extensive pipeline for realizing the QISA and achieve high instruction-issue rate is designed.
(2) We proposed a Multi-stage Instruction Decode pipeline that helps make the micro-architecture design portable across multiple qubit-realization technologies. By updating the contents of the Quantum Control Store, necessary technology dependent constraints can be introduced.
(3) We implemented FPGA-controlled DAC-based and DDS-based waveform generators to generate simple waveform used for one- and two- qubit operations. The flexibility of custom designed hardware to realize quantum operation can significantly bring down the operation costs. Moreover, a fully-integrated system for sequence generation and analog waveform generation using modular devices tailoring 'shaped' control pulses for qubit control, is the first step towards development of standardized unit-cells in quantum control.
(4) A complete control architecture for quantum Control and Read-out of Spin-Qubits is proposed with all the hardware synchronized and achieving timing-precise waveform generation.

7.2 Future Work

The scope of the current research is 'wide' and therefore requires an understanding of quantum operations at all of the layers of the Full-Stack. A simple example of the same is presented in Section 4.5 where based on the type of Quantum Algorithm, by adding support for the algorithm, the micro-architecture layer can highly optimize the system for efficient execution. Similarly, an understanding the waveform generation units is required to make future control infrastructures for qubits, such as real-time waveform synthesizers, Parameterized Waveform Generation, TDM/FDM mechanism for signals etc.

Motivated by the challenges faced and an understanding of the requirements for future quantum control systems, we need to address a number of challenges to CC-Spin design in the future. These are as follows:

(1) The Micro-code Decode Unit needs to be implemented that will allow for multi-stage Instruction Decoding and will help support different qubit-technologies.
(2) Instruction Issue Rate needs to be increased and implementation of SIMQ (Single-Instruction-Multiple-Qubit) addressing is essential to parallelize the operation on the hardware. A dense encoding of QISA instructions is required along with VLIW or Super-scalar Architectures to implement the Quantum Pipeline.
(3) For advanced quantum algorithms (such as, VQE, QLSA, QITE etc.) the Classical Pipeline requires many complex operations often comprising of classical optimization libraries specified in high-level scientific packages. Therefore, implementation of Classical Pipeline on the Hard Processing System is the right way to go.
(4) Taking into consideration the near-term Quantum Algorithms, local memory definitions are required for storing data and on hardware quantum circuit with tunable parameters.
(5) A distributed architecture with ability to address large qubit counts also enables Quantum Error Correction (QEC). Development of complete, parameterized waveform generation units, synchronous with master-controller in a distributed architecture should be the next step in development of efficient hardware for quantum control.

7.3 Addressing Scalability

The current methods of quantum control are apt for few qubit experiments but scaling the current approach to large qubit counts leads to expensive (time and resources) control infrastructure due to the varied qubit properties (such as, resonance frequency) due to process variations in fabrication. And although, qubit fidelities up to 99.9% is achievable, it is the absolute minimum for performing Quantum Error Correction. Further, the non-idealities in signal parameters and noise leads to reduced fidelity of qubit operations. The following solutions can be considered to address this issue:

(1) Pulse shaping is required to obtain better gate and read out fidelities. For example, DRAG (Derivative Removal by Adiabatic Gate) Pulses requires pulse shaping capability.

They are typically <10ns in duration and result in >0.999 fidelity.

(2) Parameterized waveform generation. Modern DDS based waveform generation helps to generate parameterized waveform by specification of parameters such as, frequency, phase and amplitude for real-time signal generation.

(3) Time Division Multiplexing (TDM) can be used to reduce the number of wires running from the controller to the qubit chip. Multiple qubits can addressed by multiplexing the control signals over a single line and using switching logic. The drawback of TDM is that it limits the extent of parallel operation execution.

(4) Frequency Division Multiplexing (FDM) of waveforms on a single line used to address different sub-blocks of different qubit frequencies.

(5) SoC Controlled Architecture for feed-forward quantum operations and Variational Quantum Algorithms. While simple boolean-controlled branch statements can be handled by on-FPGA micro-architecture, complex branching logic (such as, those that require a complete python subroutine execution) statements, used in variational quantum algorithms would require an on-chip processor(such as an SoC-FPGA architecture) to execute these sub-routines with low-latency. Further, Variational Quantum Algorithms also comprise of classical Optimization steps and periodic re-initialization of qubits that can be better achieved using an SoC-FPGA architecture.

ACKNOWLEDGEMENTS

The research leading to these results was completed as a part of master thesis at the Quantum Computer Architecture Laboratory, QuTech at Delft University of Technology, the Netherlands. Amitabh Yadav also acknowledges the support received by Lawrence Berkeley National Lab and by the Quantum Information Science Enabled Discovery (QuantISED) for High Energy Physics program (KA2401032).

REFERENCES

[1] K. Bertels, A. Sarkar, A. A. Mouedenne, T. Hubregtsen, A. Yadav, A. Krol, and I. Ashraf. 2019. Quantum Computer Architecture: Towards Full-Stack Quantum Accelerators. arXiv:quant-ph/1903.09575

[2] Hendrik Bluhm, Sandra Foletti, Izhar Neder, Mark Rudner, Diana Mahalu, Vladimir Umansky, and Amir Yacoby. 2011. Dephasing time of GaAs electron-spin qubits coupled to a nuclear bath exceeding 200 μs. *Nature Physics* 7, 2 (2011), 109.

[3] Andrew W. Cross, Lev S. Bishop, John A. Smolin, and Jay M. Gambetta. 2017. QISKit OPENQASM. *arXiv:1707.03429* (2017).

[4] Xiang Fu, L Riesebos, MA Rol, Jeroen van Straten, J van Someren, Nader Khammassi, Imran Ashraf, RFL Vermeulen, V Newsum, KKL Loh, et al. 2019. eQASM: An executable quantum instruction set architecture. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 224–237.

[5] 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. 2018. eQASM: An Executable Quantum Instruction Set Architecture. *arXiv:1808.02449* (2018).

[6] X Fu, MA Rol, CC Bultink, J van Someren, N Khammassi, I Ashraf, RFL Vermeulen, JC De Sterke, WJ Vlothuizen, RN Schouten, et al. 2017. An experimental microarchitecture for a superconducting quantum processor. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 813–825.

[7] N. Khammassi et al. 2018. OpenQL 1.0: A Quantum Programming Language for Quantum Accelerators,. *QCA Technical Report* (2018).

[8] Nader Khammassi and Imran Ashraf. 2016. *OpenQL Compiler*. https://github.com/QE-Lab/OpenQL

[9] Nader Khammassi, I Ashraf, X Fu, Carmen G Almudever, and Koen Bertels. 2017. QX: A high-performance quantum computer simulation platform. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 464–469.

[10] Nader Khammassi, Imran Ashraf, J v Someren, Razvan Nane, AM Krol, M Adriaan Rol, L Lao, Koen Bertels, and Carmen G Almudever. 2020. OpenQL: A portable quantum programming framework for quantum accelerators. *arXiv preprint arXiv:2005.13283* (2020).

[11] N. Khammassi, G. G. Guerreschi, I. Ashraf, J. W. Hogaboam, C. G. Almudever, and K. Bertels. 2018. cQASM v1.0: Towards a Common Quantum Assembly Language. arXiv:quant-ph/1805.09607

[12] Philip Krantz, Morten Kjaergaard, Fei Yan, Terry P Orlando, Simon Gustavsson, and William D Oliver. 2019. A quantum engineer's guide to superconducting qubits. *Applied Physics Reviews* 6, 2 (2019), 021318.

[13] Daniel Loss and David P DiVincenzo. 1998. Quantum computation with quantum dots. *Physical Review A* 57, 1 (1998), 120.

[14] J Medford, Johannes Beil, JM Taylor, SD Bartlett, AC Doherty, EI Rashba, DP DiVincenzo, H Lu, AC Gossard, and Charles M Marcus. 2013. Self-consistent measurement and state tomography of an exchange-only spin qubit. *Nature nanotechnology* 8, 9 (2013), 654.

[15] Michael A Nielsen and Isaac Chuang. 2002. Quantum computation and quantum information.

[16] Jason R Petta, Alexander Comstock Johnson, Jacob M Taylor, Edward A Laird, Amir Yacoby, Mikhail D Lukin, Charles M Marcus, Micah P Hanson, and Arthur C Gossard. 2005. Coherent manipulation of coupled electron spins in semiconductor quantum dots. *Science* 309, 5744 (2005), 2180–2184.

[17] John Preskill. 2018. Quantum Computing in the NISQ era and beyond. *Arxiv* 1801.00862 (2018), 20.

[18] D. Ristè, S. Poletto, M.-Z. Huang, A. Bruno, V. Vesterinen, O.-P. Saira, and L. DiCarlo. 2015. Detecting bit-flip errors in a logical qubit using stabilizer measurements. *Nature Communications* 6, 1 (Apr 2015). https://doi.org/10.1038/ncomms7983

[19] Aritra Sarkar. 2018. Quantum Algorithms: for pattern-matching in genomic sequences. (2018).

[20] Alexei M Tyryshkin, Shinichi Tojo, John JL Morton, Helge Riemann, Nikolai V Abrosimov, Peter Becker, Hans-Joachim Pohl, Thomas Schenkel, Michael LW Thewalt, Kohei M Itoh, et al. 2012. Electron spin coherence exceeding seconds in high-purity silicon. *Nature materials* 11, 2 (2012), 143.

[21] Jeroen PG van Dijk, Edoardo Charbon, and Fabio Sebastiano. 2019. The electronic interface for quantum processors. *Microprocessors and Microsystems* 66 (2019), 90–101.

[22] TF Watson, SGJ Philips, Erika Kawakami, DR Ward, Pasquale Scarlino, Menno Veldhorst, DE Savage, MG Lagally, Mark Friesen, SN Coppersmith, et al. 2018. A programmable two-qubit quantum processor in silicon. *Nature* 555, 7698 (2018), 633.

[23] X Xue, TF Watson, J Helsen, DR Ward, DE Savage, MG Lagally, SN Coppersmith, MA Eriksson, S Wehner, and LMK Vandersypen. 2019. Benchmarking gate fidelities in a Si/SiGe two-qubit device. *Physical Review X* 9, 2 (2019), 021011.

[24] Amitabh Yadav. 2019. CC-Spin: A Micro-architecture design for scalable control of Spin-Qubit Quantum Processor. (2019).

[25] Jun Yoneda, Kenta Takeda, Tomohiro Otsuka, Takashi Nakajima, Matthieu R Delbecq, Giles Allison, Takumu Honda, Tetsuo Kodera, Shunri Oda, Yusuke Hoshi, et al. 2018. A quantum-dot spin qubit with coherence limited by charge noise and fidelity higher than 99.9%. *Nature nanotechnology* 13, 2 (2018), 102.

[26] Floris A Zwanenburg, Andrew S Dzurak, Andrea Morello, Michelle Y Simmons, Lloyd CL Hollenberg, Gerhard Klimeck, Sven Rogge, Susan N Coppersmith, and Mark A Eriksson. 2013. Silicon quantum electronics. *Reviews of modern physics* 85, 3 (2013), 961.