CHEQ: Towards Enabling Circuit Integrity Checking in Quantum Controllers

Barbora Hrdá
Yale University,
Technical University of Munich,
and Fraunhofer AISEC
Munich, Germany
barbora.hrda@tum.de

Sanjay Deshpande
Yale University
New Haven, CT, USA
sanjay.deshpande@yale.edu

Theodoros Trochatos
Yale University
New Haven, CT, USA
theodoros.trochatos@yale.edu

Jakub Szefer
Northwestern University
Evanston, IL, USA
jakub.szefer@northwestern.edu

Abstract

Rapid advances in quantum computing hardware and software are bringing closer the promise of new discoveries and breakthroughs that these machines will enable. To fully utilize and trust the quantum computers, however, users need to have assurances about the confidentiality and integrity of quantum circuits that they execute on the quantum computers. While existing research has begun to address the issues of quantum circuit confidentiality, for example, through various obfuscation methods, there is lack of quantum computer architecture or hardware designs for ensuring and checking the integrity of quantum circuits. This gap in existing research and design of quantum computers is addressed in this paper. This work outlines the design of CHEQ, a Circuit Hashing Engine for Quantum controllers. This work first presents integrity requirements for quantum circuits, then details the design of CHEQ, along with first set of evaluation results. By providing circuit integrity measurements to users through CHEQ, quantum computing systems can become more resilient to security threats that aim to attack circuit integrity. Combined with other prior work on confidentiality, the new CHEQ integrity assurance in quantum computers can enable complete circuit protection, and thus protection of the future discoveries and breakthroughs generated by quantum computers.

ACM Reference Format:
Barbora Hrdá, Sanjay Deshpande, Theodoros Trochatos, and Jakub Szefer. 2025. CHEQ: Towards Enabling Circuit Integrity Checking in Quantum Controllers. In *Great Lakes Symposium on VLSI 2025 (GLSVLSI '25), June 30-July 2, 2025, New Orleans, LA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3716368.3735296

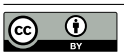
1 Introduction

Quantum computing is expected to solve problems in specific applications like machine learning, cryptography, or chemistry that are currently intractable for classical systems. It leverages the principles of quantum mechanics to process information in fundamentally different ways than classical computing. Quantum algorithms such as Shor's [39] and Grover's [20] demonstrate quantum computing's ability to solve certain problems exponentially faster than classical computers. Quantum algorithms are represented by quantum circuits. These are composed of quantum operations executed through quantum gates, which manipulate qubit states in accordance with the principles of quantum mechanics. These circuits are implemented in a programming environment and transpiled into device-specific instructions for the target Quantum Processing Unit (QPU). Quantum gates are abstract representations of transformations applied to qubits, typically implemented using techniques like microwave pulses or other control methods. The characteristics of these pulses – specifically their shape, intensity, and duration – ultimately determine the resulting state of qubits. These pulses can be modified to indicate errors, falsify results or add noise to the entire system, representing a breach of the circuit's integrity.

In order to utilize quantum computers, users need to send their quantum circuits and inputs to remote third party providers. To ensure the accuracy of a calculation, it is crucial to verify that these pulses have not been altered prior to their execution. As of today, there is only limited research on security mechanisms for quantum computers to ensure the confidentiality of user's inputs or circuits, and hardly any work looking at integrity and quantum computer controllers, posing significant security and privacy concerns. Today's quantum computing users must rely on the operator to process their data correctly and exclusively for its intended purpose. To fully utilize quantum computers, users need assurances about the confidentiality and integrity of quantum circuits they execute on third party quantum computers.

Recent studies have focused primarily on developing protocols that partition calculations to obfuscate or encrypt the original circuit's information. This includes, the areas of research on Blind Quantum Computation [1, 10, 12, 18, 27, 42], Quantum Homomorphic Encryption [3, 19, 26, 44], Circuit Cutting for security [47] or trusted execution environments for quantum computers [45, 46]. In contrast, our approach attempts to integrate integrity protections for quantum circuits to help assure users that their circuits were executed correctly and without modification.

This work introduces and outlines the design of the Circuit Hashing Engine for Quantum Controllers (CHEQ). Immediately



GLSVLSI '25, New Orleans, LA, USA

ACM ISBN 979-8-4007-1496-2/2025/06
https://doi.org/10.1145/3716368.3735296

before execution, a controller unit converts digital instructions into analog signals or pulses. Controllers translate the classical bit-instruction into quantum operations. This requires timing, accuracy, but also integrity in order to perform correct and secure quantum operations. Thus, this work introduces integrity requirements for quantum circuits and details how key components of a quantum controller can be leveraged to utilize a set of hash engines for ensuring the integrity of circuits and pulses. This work utilizes the open source FPGA-based Quantum Instrumentation Control Kit (QICK) [40] controller. QICK is an open source Xilinx RFSoC-based qubit controller [40], running on a RFSoC4x2 board by Xilinx [16]. It provides a digital radio-frequency (RF) board hosting a RFSoC-FPGA, custom firmware and custom software (both open source). It provides a platform for control and readout of different quantum systems, with the flexibility to extend its firmware by adding custom functionality.

QICK's accessibility and modifiability enable the integration of CHEQ's security features on a RF-SoC FPGA board. To identify the optimal placement and targets for CHEQ, this work analyzes various attack paths a malicious actor might exploit during gate processing leading up to pulse execution. Finally, this work analyses CHEQ's overhead and discusses limitations of the design that motivate future work in this direction.

Contribution

This work examines potential security measures for integrity protection within a quantum controller, focusing on

- definition of integrity requirements for quantum circuits across their lifecycle,
- conceptual design of integrity protection through hashing on pulse control level,
- design of a Circuit Hashing Engine for Quantum controllers (CHEQ), and
- presenting first results of the architecture's ability to protect quantum circuits.

2 Background

2.1 Quantum Access via Cloud-based Platforms

Currently, access to quantum computers is provided via cloud-based platforms. These platforms typically integrate quantum hardware, control mechanisms, and software into a unified system. They can be provided either by the manufacturer of the quantum computer itself (e.g., IBM [23], Quantinuum [36] or Rigetti [37]) or by other companies (e.g., AWS provides access to QPUs by different vendors [38]). These usually include a few free-of-charge QPUs that are shared with other users, as well as fee-based access to more advanced QPUs. Control and measurement systems manage qubit operations and transpilation to the targeted QPU. Quantum software often incorporates an SDK tailored to the specific QPU in order to create and run quantum algorithms. The processing across cloud providers can be abstracted to the following steps: Circuits are received, stored, optimized, queued, converted into pulses and sent to a QPU accordingly [21, 33]. Calculation results are stored again and returned to the user. In order to determine what parts of the calculation require protection, it is essential to closely examine the computations performed on a quantum computer.

A calculation that is sent to the cloud provider consists basically of three key components: input, circuit, and output. Based on [21], we are considering all three to be assets. These may include the following:

- data inputs (as there is no memory in current quantum computers, the input values are hard-coded into the circuit, e.g., sensitive medical records),
- quantum circuits (proprietary quantum algorithms, e.g., a company's intellectual property), and
- outputs (results produced by quantum computations, e.g., decrypted RSA keys).

These assets (data inputs and quantum circuits) are sent from a user's local device through a third party cloud provider to a QPU. Cloud providers inherently access the users' assets when receiving circuits for execution. Given the sensitivity, value, and potential impact in case of a security breach, quantum computing assets require robust protection measures, integrated idealy into the quantum computing ecosystem. Due to the current limited availability of QPUs, protection measures should prioritize confidentiality and integrity protection over availability.

2.2 Quantum Controller Architecture

Quantum controllers are the interface between the classical control system and the QPU. They translate classical bit-instruction into quantum operations, are used for calibration, pulse shaping, and qubit control. This requires timing, accuracy, but also integrity in order to perform correct and secure quantum operations. Depending on the targeted QPU, there are different types of quantum controllers, e.g., microwave [5, 43], laser [14, 41], or FPGA-based controllers [40]. Regardless of the targeted physical qubit technology, the controllers' architecture is similar, and typically includes a control unit, a pulse or laser generator, and several digital-to-analog converters (DAC) encoding and firing the actual pulses. Additionally, for the readout of quantum results, controllers also contain several analog-to-digital converters (ADC). In order to create a pulse, controllers generally need the following information:

- targeted DAC channel,
- starting time of the pulse,
- I/Q values for the waveform envelope,
- pulse frequency, phase, duration and gain.

Further, each quantum circuit is composed of multiple pulses on multiple DAC channels. All the pulses and the order of the pulses on a DAC channel, and among DAC channels, is important. Thus, integrity protection measures need to take care of hashing the above pulse information for each pulse, as well as timing information for when different pulses are executed on different DAC channels to capture the whole history of the pulses executed.

The amount of information needed per pulse by different controllers may vary. In this work we use QICK and information is used as a demonstrative example. Controllers are the final point where pulses can be modified before their transmission to the QPU, thus, this is where the integrity protection is added.

3 Threat Model

This work assumes a remote user who sends quantum circuits to an untrusted third party cloud-based QPU provider and wants to

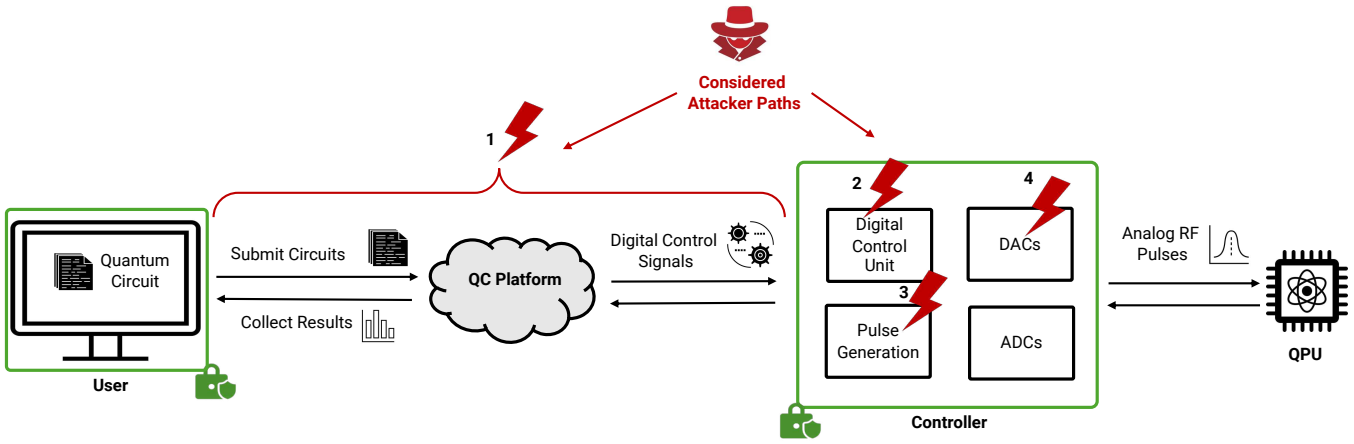


Figure 1: This illustration depicts the abstracted process flow of a circuit from creation at the user's development environment to execution at a third party QPU. As outlined in Section 4.1, the circuit undergoes modifications at various stages. The red lightning symbols highlight potential attack paths (1 - 4) identified in Section 3. As described in Section 4.1, as a first step this work considers the user and the controller to be trusted, symbolized using the green locks. This work focuses on the process leading up to the execution of circuits on a QPU. The handling of results sent back to the user, passing the controller's analog-to-digital converters (ADCs), falls outside the scope of this study and could be addressed in future research.

obtain a correct result from a quantum computation while ensuring their circuit was unmodified before execution. Given existing work that focuses on confidentiality protection, this work focuses only on integrity protection, i.e., detecting modification to the users' circuits. Thus, confidentiality-related attacks, such as side channel or timing attacks, are not considered.

In this threat model, we assume a malicious attacker who wants to alter the users' quantum circuits, in order to disrupt or falsify the calculations, and consequently, the results. In Fig. 1 the red lightning symbols highlight potential attack paths within a typical cloud-based quantum computing system. We assume the attacker cloud try to compromise the network (attack path 1), over which the circuit is send, or parts or the whole controller (attack paths 2, 3 and 4), e.g., by being an insider at the quantum provider side. The attacker's capabilities could include manipulating user's quantum circuits or pulse instructions to interfere with the controller's operations or induce errors (e.g., tampering with control signals or qubit settings).

The initial transfer of user's quantum circuits through the cloud platform to the controller is a first point of attack (attack path 1). An attacker could, for instance, execute a man-in-the-middle attack to intercept, analyze, and manipulate circuit transmissions. This attack path is neither unique to quantum computing nor quantum-specific. Instead, it belongs to the broader category of secure information transmission to remote cloud infrastructures. This attack path is typically addressed using state-of-the-art protection, e.g., network encryption and hashing. Thus, this work focuses on quantum-specific parts of the attacker model (attack paths 2, 3, and 4).

The controller forms a part of the quantum-specific architecture, consisting of the digital control unit, pulse generation, DACs and ADCs. The digital control unit and pulse generation modules store, organize, and ultimately process information required for pulse generation. These modules typically run on classic control computers. In this case, the attacker could take control over the digital control unit and manipulate the circuits or pulses building blocks (attack path 2). The control unit creates information used to generate the pulses in the pulse generation unit, and the attacker could target that unit (attack path 2). Lastly, the DAC takes binary instructions and turns them into analogue pulses. DACs are part of, e.g., classical RF-boards. In this instance, the attacker could try to take over control of the RF-board and manipulate the bit-to-pulse conversion, resulting, e.g., in falsified results (attack path 3).

In this threat model, the controller is trusted. However, in real-world applications, the controller might be attacked or compromised. To detect or prevent a compromised controller, future work could implement protective measures such as embedding a trusted execution environment or incorporating attestation mechanisms into the controller. This work focuses on integrity protection of the user's inputs and quantum circuits. The controller also contains analog-to-digital converters (ADCs) for reading out responses from the qubits. Future efforts may also focus on implementing readout protection.

4 Circuit Hashing Engine for Quantum Controllers (CHEQ)

This section discusses where and how integrity protection for quantum circuits should be added, and then details the design of CHEQ.

4.1 Integrity Protection for Quantum Circuits

Circuit integrity protection must be tailored to and implemented across various stages of the circuit life-cycle, see Fig. 1. In general, every parameter that contributes to the structure of the circuit, or individual pulses, should be hashed. On user's side, the circuit's initial state contains transpiled instructions for a targeted QPU, usually in a quantum assembly language like OpenQASM [13]. The cloud-based platform performs circuit optimization (e.g., reducing

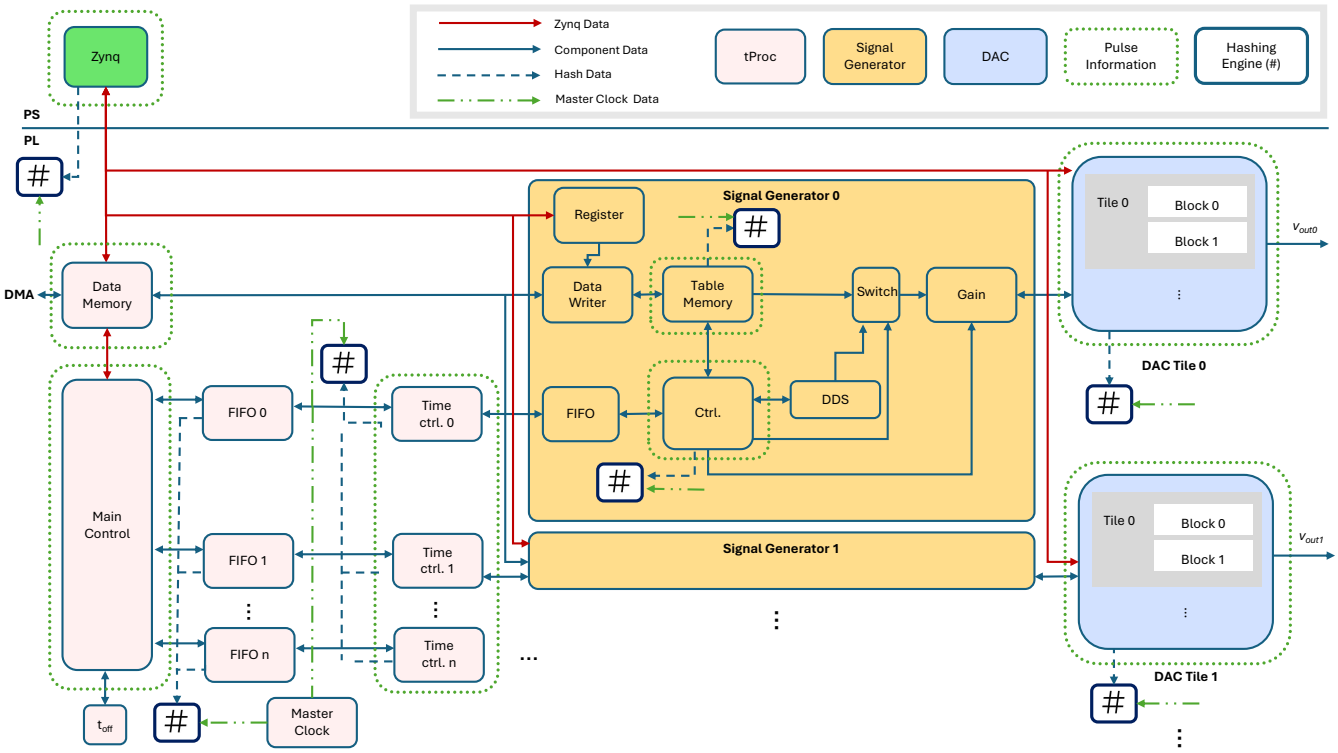


Figure 2: This block diagram of the QICK quantum computer controller is based on [16, 32, 40]. The main QICK components on the programmable logic (PL) side are the timed-processor (tProc), signal generator (SG) and RF blocks (Digital-to-Analog converter (DAC)). Green dotted squares illustrate the locations where pulse information is generated, modified and stored, and thus highlight potential locations for applying integrity protection. Red arrows indicate data that is send directly from Zynq to the three main components. Blue arrows indicate communication within and between components. Dashed arrows symbolize data that is send for hashing to CHEQ engines. Green dashed arrows indicate master clock information send to CHEQ engines. For the sake of clarity, only abbreviated forms of the master clock inputs are shown in this figure. All CHEQ engines buffer the input they receive, push it to their internal SHAKE modules and send a hash back to Zynq. For the sake of clarity, returning the hash values to Zynq is not shown in this figure. At the main control, instructions are dispatched into a time queue and executed by a specified signal generator (SG) channel in the specified order. For the sake of clarity, only one signal generator, including its internal components, is represented; a second SG is shown as a solid block. All SG are constructed similarly. Pulse parameters are passed from the tProc to the SG and queued into the FIFO of each SG channel. The table memory's parameters are not mandatorily passing the tProc and need to be hashed either within the table memory itself or before they are passed to the SG. Integrity protection for the individual pulses should be performed within the specific DAC tiles.

the circuit depth by removing or restructuring gates) and conversion before the circuit instructions are sent to the controller. The controller itself assigns the corresponding pulse parameters to the individual DACs, loads information on envelope forms into its memory and performs the circuit-to-pulse conversion. The generated pulse is then sent to the QPU.

As described in Section 3, this work focuses on the quantum-specific architecture of the attacker model. We identified 3 attack paths within the controller: the Digital Control Unit, the Pulse Generation, and the DACs. In order to protect the circuits' integrity, users' quantum circuit related instructions or data processed by each of the parts needs to be hashed. Because we assume the controller hardware (i.e. CHEQ) can be trusted, we incorporate hashing engines in various parts of the controller.

In future, the trust boundary could be further reduced. For example, the DACs are the final point where the digital information is turned into analog pulses. Security-aware DACs could be created where only data passing through the DACs is hashed. Conversely, assuming the attacks could only happen before the user's quantum circuits reach the controller, all the integrity protection could be in the Digial Control Unit.

4.2 CHEQ Design

For the design of CHEQ, we utilize the SHAKE256 hash module as a new addition to QICK. SHAKE256 is an extendable-output function (XOF) based on the Keccak sponge construction and is standardized by NIST as part of the SHA-3 family [7]. SHAKE256 provides flexibility by allowing output generation of any desired

Table 1: This table presents locations for hashing engines for different hashing levels: from whole circuits to pulse generation.

Hashing Level	Hash Location	Size of Hashed Information	Explanation
Whole circuit	User's Environment, Platform, Controller	3 hash locations × 256 bits	At the user's side the hashing module should be added after the circuit has been transpiled, at the platform's side after the optimization. In the controller hashing should happen in zynq, before the circuit is divided and dispatched to the individual channels.
Pulse Generation Information	Zynq, data memory, main control, time control, SG control and table memory for each SG channel	number of instructions × 6 hash locations × 256 bit	Each pulse consists of different parameters, e.g., 80-bit instructions, 16-bit envelope information, execution time or channel information distributed in QICK across 6 submodules.
Individual Pulses	DAC Tiles and Zynq	number of instructions × 256 bit	Individual pulse instruction should be hashed within their corresponding DAC.

length. It utilizes the Keccak-f permutation function, which operates on a 1600-bit state using a sponge construction. The core operation, Keccak-f[b], consists of multiple transformation rounds. In the SHAKE256 setting, the number of rounds is 24. Each round consists of five layers: θ - diffusion layer, ρ - bit rotation layer, π - lane permutation layer, χ - non-linearity layer and ι - XORing with a round constant. The 1600-bit Keccak-f input state is split into two parts: a rate of 1088 bits and a capacity of 512 bits. The input block size for SHAKE256 is determined by the rate, which is 1088 bits. NIST has standardized SHAKE256 as a recommended XOF for applications requiring hashing-based cryptographic constructions [31]. NIST's PQC standardization effort has recognized SHAKE256 as a core component in some post-quantum cryptographic algorithms due to its security, efficiency, and hardware-friendly design [28].

In our hardware design of SHAKE256, the implementation of Keccak-f round function layers $\{\theta, \rho, \pi, \chi,$ and $\iota\}$ is fully combinatorial, and we operate on the full-width of 1600-bits in parallel. We register the output after each round, and then the output is fed back into the round function for the next iteration.

Fig. 2 highlights QICK's modules targeted for hashing.

In QICK, high-level circuits are created in Jupyter Notebooks or Python. These instructions are passed from the RFSoC Zynq processor (processing system (PS)) to the firmware blocks (programmable logic (PL)). The parameters responsible for pulse generation and readout are distributed across several sub-modules: timed-processor (tProc), signal generator (SG) and RF blocks (DACs and ADCs). Hashing the individual information needed to generate the pulses, enables fine-grained identification of changes within the pulses. In context of this work, the pulse information is found in the submodules shown in Fig. 2.

The block diagram of tProc's current version (64-Bit) is detailed in [32]. Its main control uses standard instructions to operate on registers, push/pop instructions to stack and compute times to execute timed-instructions at specific absolute times. Timed instructions make use of the master clock and offset register T_{off}. When the processor decodes a timed instruction, it computes the absolute time with T_{off} before dispatching it into the timed instruction control queue of that particular SG channel. This also allows the execution of instructions at the same absolute time in different SG channels. Instructions with a time tag are stored in the program memory, before they are dispatched to their respective execution channels. The master clock is a 48-bit counter, which, clocked at the maximum FPGA speed of 500 MHz, a 2 *ns* period, gives roughly 156 hours of counting. It starts counting when the software starts at address 0 and never stops. Each of the tProc's memory location has a 64-bit number which encodes instruction together with the necessary parameters for execution. The key information for pulse creation in the main control and time control queue includes the absolute execution time and the corresponding SG channel, which corresponds to a targeted qubit within the QPU. This information needs to be hashed.

All information required to create pulses is sent to the SG channels via various input channels. The tProc is connected to the SG using a 160-bit interface. The SG creates pulse envelopes according to the user's instructions. A single instruction in QICK creates a waveform that includes the waveform envelope's start address, DDS frequency and phase, pulse duration, output selection, and gain [40]. Each SG channel has its own FIFO and associated time-control. There are three steps in order to create and fire a pulse. Firstly, I and Q values for the waveform envelope (e.g Gaussian) are preloaded as 16-bit words from the Zynq processor directly into the SG table memory section and DDS respectively. This information needs to be hashed. Secondly, initial values are set for all parameters. The SG receives waveform configuration instructions in 80-bit blocks from the tProc. Also, these 80-bit instructions need to be hashed. As instructions are passed to the corresponding SG channels, adding hash engine to all SG channels is necessary. If values change, the channel's registers that need to be updated are overwritten using their page and address information. Assembly instructions are then used to change the value of these registers.

Thirdly, a pulse is fired on the specified channel at the specified time, using whatever values are loaded in the registers. Timed instructions are executed when the control core of the tProc pushes the instruction into the timed queue (FIFO) or when the instruction pops off the timed queue. Then, the DDS block synthesizes the tone for the digital upconversion and the switch determines the corresponding mode. The signal then passes through the gain block before entering the specified DAC tile and block, where the pulse is finally fired. Hashing the resulting pulse itself would require adding

Table 2: This table provides area estimates for our SHAKE256-based CHEQ design and time estimates for generating the hashes. For the evaluation, we used a 100-gate quantum arithmetic circuit. QICK's original size includes 82.029 LUT and 124.948 FF. For hashing levels 'Individual Pulses' and 'Pulse Generation Information' the size of hashed information depends on the number of quantum gates, i.e. instructions, in the quantum circuit. Our Xilinx RFSoC4x2 board currently provides 4 DACs, representing 4 qubits. The hashing levels are explained in Table 1.

Hashing Level	Hash Output Size	Area		†% Increase		Freq.	Timing		
	(bits)	LUTs	FFs	LUTs	FFs	(MHz)	Cycles	Time (*ns*)	Time/Pulse (*ns*)
Whole Circuit	256	4,086	1,621	5%	1%	500	200	400.00	4.00
Full circuit integrity check using one SHAKE256 module									
Individual Pulses	256	4,086	1,621	5%	1%	500	200	400	4.00
Pulse Generation Information	1,536						1,125	2,250.00	22.50
Qubit-wise integrity check (4-qubits) using four SHAKE256 modules									
Individual Pulses	1,024	16,344	6,484	20%	5%	500	50	100	0.25
Pulse Generation Information	6,144						281	562.50	1.41
Instruction-wise integrity check (4-qubits) using four SHAKE256 modules									
Individual Pulses	25,600	16,344	6,484	20%	5%	500	625	1,250.00	3.13
Pulse Generation Information	153,600						3,750	7,500.00	18.75

† % increase in the overall area of the controller when the Hash Engine is interfaced with the QICK quantum controller as shown in Fig. 2.

hashing to Zynq and all DAC blocks. All hash information is returned via Zynq to the user. To sum it up, the following submodules process essential information to safeguard pulse generation: Zynq, Data Memory, Main Control, Time Control, Table Memory, Control (Ctrl.) and the corresponding DAC tiles, see Fig. 2.

4.3 Preliminary Evaluation Results

The required resources for CHEQ are detailed in the following, categorized by the hardware (FPGA) code size increase caused by adding CHEQ engines to QICK and additional data generated through hashing, see also Table 2.

The quantum circuit size, defined by the number of gate operations, is influenced by factors such as the input size or the algorithm's complexity. For example, a quantum adder circuit that adds numbers 50 and 90 has a total of around 100 quantum gates (i.e. instructions), depending on the actual implementation. To ensure integrity protection for the pulse generation information, hashing is applied to each of the 100 instructions, resulting in 100 gates $\times$ 6 hashing locations $\times$ 256 bit hashes. In addition, a dedicated hashing engine is added to each of the six submodules, as shown in Fig. 2. The number of added CHEQ hash engines is independent from the size of the quantum circuit size, but depends on the QICK design.

One approach to reduce the size of the hashing information, while still providing sufficient security, is grouping individual pulses in the SGs and generate a hash for each group. Data packets containing pulse information could be combined into up to 1600-bit blocks. Using our SHAKE256 module, CHEQ could operate on the full-width of 1600-bits in parallel. This would still enable the detection of changes in the generation of pulses, even if the exact data point is not identifiable. This approach also addresses the scalability of circuits.

Further, the overall output could be compressed by hashing all the hash outputs together to generate one 256-bit hash value. The key benefit is that the user receives only 256 bits in return; however, while integrity violations can be detected, their exact location cannot be identified. If, as in Table 2, individual hash outputs are returned, the user can possibly identify which instruction was modified or at what point in the controller process an integrity violation occurred. Alternatively, the user could select the desired level of detail for the collected information, thereby adjusting the amount of hash data generated by CHEQ. Although integrity violations can be detected in all settings, the user's influence lies in determining the granularity with which the source of an error or attack is identified. This means that the user can choose whether to pinpoint an integrity violation to a specific qubit or to a broader group of instructions.

This work uses a small circuit to demonstrate how circuit design and qubit count affect the overall hash sizes. Advances in quantum computing scalability enable the processing of increasingly complex and extensive algorithms. However, a comprehensive assessment of CHEQ's efficiency in larger, or even error corrected, quantum computers, requires further investigation, which lies beyond the scope of this work.

4.4 Choice of Hash Engines

For the hash engine used in our CHEQ design, we consider several widely recognized hash functions in the cryptographic community, including SHA2-256/512 [29], SHA3-256/512, SHAKE128/SHAKE256 [30], and Haraka-256/512 [24].

SHA2-256, which is part of the SHA2 family standardized by NIST, remains a cornerstone of classical security protocols such as TLS. Haraka-256/512, although not standardized, is a lightweight, AES-based hash function specifically designed for high-speed hardware implementations and was featured in SPHINCS+ [22], a NIST-selected post-quantum signature scheme. SHA3-256/512 and SHAKE-128/SHAKE256, based on the Keccak sponge construction and also

standardized by NIST, offer strong security guarantees. Notably, SHAKE128/SHAKE256 allows for variable-length output and is integral to several post-quantum cryptographic schemes.

Overall, all the aforementioned hash functions are relatively FPGA-friendly. However, SHA3/SHAKE256 require more resources to achieve high speeds compared to SHA2 or Haraka hash functions [2, 15]. In our evaluation (provided in Table 2), we assume SHAKE256 to demonstrate the maximum overhead that a hash engine might introduce to the quantum computer controller. Nonetheless, the proposed interface allows for the easy swapping of any hash function based on the available resource constraints.

4.5 Limitations

Currently, this work focuses on integrity protection of the generated input signals, but not on the readout values that are received via ADCs. In the future is seems conceivable to expand CHEQ to cover also output integrity protection. Also, including confidentiality protection to the engine is a shortcoming of this paper and should be addressed in the future. Furthermore, this work focuses on NISQ devices. In the future, error-correcting codes will be used to send instructions back and forth in a loop between the controllers, the error-correcting devices and the QPU. As these are currently neither scalable nor widespread, we do not consider them in this paper. Given these limitations, this paper does not include error corrected quantum computers in its analysis or discussions. Our focus remains on currently implementable and widely accessible quantum computing technologies.

5 Related Work

A number of research projects have considered confidentiality protection for quantum circuits. This section lists some of the main approaches.

Other Controller Architectures. Open-source quantum control systems vary significantly in both architecture and implementation. QubiC [49] uses a distributed setup, where each qubit is managed by its own processor, reducing hardware complexity and simplifying control logic. QICK [40] employs a centralized design, controlling multiple qubits with a single processor, which streamlines software coordination across channels. The two systems also differ in how they handle instructions: QubiC configures all parameters of an RF signal generator at once using a single 128-bit instruction to minimize latency, while QICK adjusts each parameter separately using 80-bit instructions, allowing for greater programming flexibility. This variation illustrates the broad range of design options in quantum control systems, emphasizing the importance of effectively exploring this space to meet strict timing requirements while managing hardware resources efficiently.

Blind Quantum Computing (BQC). BQC protocols allow a client to delegate and perform quantum computations on an untrusted remote server by ensuring the confidentiality of input, gates, and output, while limiting information leakage to e.g. resource requirements or circuit depth [1, 4, 6, 10, 12, 17, 18, 27, 42].

Some BQC protocols incorporate mechanisms to detect unauthorized manipulations and verify results [6, 17]. BQC protocols typically require clients capable of basic quantum operations, such as qubit measurement or qubit state preparation and assume a quantum network for multi-round client-server communication. In comparison, this work does not require any quantum capabilities and, integrates protective mechanism at hardware level within the controller.

Quantum Circuit Cutting. Quantum circuit cutting decomposes quantum circuits into smaller subcircuits by strategically partitioning gates (e.g., splitting multi-qubit gates) or qubit wires [8, 9, 11, 25, 34, 35, 48, 50] and enables the execution on hardware with limited qubits. Quantum circuit cutting also holds promise in the context of quantum computer security, particularly for protecting circuit structure and sensitive computation patterns [47]. By decomposing a quantum circuit into smaller fragments and distributing them across different devices, it becomes harder for an adversary to gain full knowledge about the entire computation, thereby enabling a form of obfuscation. In comparison, this work employs a security-driven methodology, embedding integrity protective mechanisms at the controller hardware level.

Hardware Mitigation Schemes. There has been a recently emerging trend in the community to propose realistic hardware mitigation schemes to protect quantum circuits from untrusted cloud providers in the near-term quantum computers [45, 46]. These schemes assume the dilution refrigerator is the trusted boundary and the sensitive part of information is being processed inside it. While this is a promising yet still developing approach, this kind of defenses comes with its own set of limitations. First, both of these approaches, increase dramatically the circuit depth, leading to fidelity decrease. Second, these methods are valid only under certain assumptions, which might not always work. For instance, the authors only assume an honest-but-curious cloud provider that only eavesdrops the quantum circuit and infers information. Finally, these approaches are tailored specifically for superconducting devices only and further research is required to adjust these methods to other device architectures.

6 Conclusion

This work introduced the Circuit Hashing Engine for Quantum controllers (CHEQ). By providing circuit integrity measurements to users through CHEQ, quantum computing systems can become more resilient to security threats that aim to attack circuit integrity. Combined with other prior work on confidentiality, the new CHEQ integrity assurance in quantum computers can enable complete circuit protection, and thus protection of the future discoveries and breakthroughs generated by quantum computers.

Acknowledgments

The project and research is partially supported by the Bavarian Ministry of Economic Affairs, Regional Development and Energy with funds from the Hightech Agenda Bayern. The research is also part of the Munich Quantum Valley, which is supported by the Bavarian state government with funds from the Hightech Agenda Bayern Plus. This project and research is also partially supported by the United States National Science foundation research grants 2332406 and 2245344, and through grant from TII.

References

[1] Dorit Aharonov, Michael Ben-Or, and Elad Eban. 2008. Interactive Proofs For Quantum Computations. arXiv:0810.5375 [quant-ph] https://arxiv.org/pdf/0810.5375.pdf.

[2] Dorian Amiet, Lukas Leuenberger, Andreas Curiger, and Paul Zbinden. 2020. FPGA-based SPHINCS+ Implementations: Mind the Glitch. In *2020 23rd Euromicro Conference on Digital System Design (DSD)*. 229–237. https://doi.org/10.1109/DSD51259.2020.00046

[3] Frederik Armknecht, Colin Boyd, Christopher Carr, Kristian Gjøsteen, Angela Jäschke, Christian A Reuter, and Martin Strand. 2015. A guide to fully homomorphic encryption. *Cryptology ePrint Archive* (2015).

[4] Pablo Arrighi and Louis Salvail. 2006. Blind Quantum Computation. *International Journal of Quantum Information* 4, 05 (2006), 883–898.

[5] Zenghui Bao, Yan Li, Zhiling Wang, Jiahui Wang, Jize Yang, Haonan Xiong, Yipu Song, Yukai Wu, Hongyi Zhang, and Luming Duan. 2024. A cryogenic on-chip microwave pulse generator for large-scale superconducting quantum computing. *Nature Communications* 15, 1 (2024), 5958.

[6] Stefanie Barz, Joseph F Fitzsimons, Elham Kashefi, and Philip Walther. 2013. Experimental verification of quantum computation. *Nature physics* 9, 11 (2013), 727–731.

[7] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. 2013. Keccak. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 313–314.

[8] Sebastian Brandhofer, Ilia Polian, and Kevin Krsulich. 2023. Optimal partitioning of quantum circuits using gate cuts and wire cuts. *IEEE Transactions on Quantum Engineering* 5 (2023), 1–10.

[9] Lukas Brenner, Christophe Piveteau, and David Sutter. 2023. Optimal wire cutting with classical communication. *arXiv preprint arXiv:2302.03366* (2023).

[10] Anne Broadbent, Joseph Fitzsimons, and Elham Kashefi. 2009. Universal Blind Quantum Computation. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*. 517–526. https://doi.org/10.1109/FOCS.2009.36 https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5438603.

[11] Daniel T Chen, Ethan H Hansen, Xinpeng Li, Aaron Orenstein, Vinooth Kulkarni, Vipin Chaudhary, Qiang Guan, Ji Liu, Yang Zhang, and Shuai Xu. 2023. Online detection of golden circuit cutting points. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, Vol. 1. IEEE, 26–31.

[12] Andrew M Childs. 2001. Secure assisted quantum computation. *arXiv preprint quant-ph/0111046* (2001).

[13] Andrew Cross, Ali Javadi-Abhari, Thomas Alexander, Niel De Beaudrap, Lev S Bishop, Steven Heidel, Colm A Ryan, Prasahnt Sivarajah, John Smolin, Jay M Gambetta, et al. 2022. OpenQASM 3: A broader and deeper quantum assembly language. *ACM Transactions on Quantum Computing* 3, 3 (2022), 1–50.

[14] Shantanu Debnath, Norbert M Linke, Caroline Figgatt, Kevin A Landsman, Kevin Wright, and Christopher Monroe. 2016. Demonstration of a small programmable quantum computer with atomic qubits. *Nature* 536, 7614 (2016), 63–66.

[15] Sanjay Deshpande, Yongseok Lee, Cansu Karakuzu, Jakub Szefer, and Yunheung Paek. 2025. SPHINCSLET: An Area-Efficient Accelerator for the Full SPHINCS+ Digital Signature Algorithm. (2025). https://doi.org/10.1145/3728469

[16] Real Digital. [n. d.]. RFSoC 4x2 - Real Digital. https://www.realdigital.org/hardware/rfsoc-4x2 Accessed: 2025-03-18.

[17] Joseph F Fitzsimons. 2017. Private quantum computation: an introduction to blind quantum computing and related protocols. *npj Quantum Information* 3, 1 (2017), 23.

[18] Joseph F. Fitzsimons and Elham Kashefi. 2017. Unconditionally verifiable blind quantum computation. *Physical Review A* 96, 1 (jul 2017). https://doi.org/10.1103/physreva.96.012303 https://arxiv.org/pdf/1203.5217.pdf.

[19] Caroline Fontaine and Fabien Galand. 2007. A survey of homomorphic encryption for nonspecialists. *EURASIP Journal on Information Security* 2007 (2007), 1–10.

[20] Lov K Grover. 1996. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 212–219.

[21] Barbora Hrdá and Sascha Wessel. 2023. Confidential Quantum Computing. In *Proceedings of the 18th International Conference on Availability, Reliability and Security*. 1–10.

[22] Andreas Hülsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Jean-Philippe Aumasson, Bas Westerbaan, and Ward Beullens. 2020. *SPHINCS$^+$*. Technical Report. National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[23] IBM. [n.d.]. IBM Quantum Platform. https://quantum.ibm.com/ Accessed: 2025-03-18.

[24] Stefan Kölbl, Martin M. Lauridsen, Florian Mendel, and Christian Rechberger. 2016. Haraka v2 - Efficient Short-Input Hashing for Post-Quantum Applications. Cryptology ePrint Archive, Paper 2016/098. https://eprint.iacr.org/2016/098

[25] Angus Lowe, Matija Medvidović, Anthony Hayes, Lee J O'Riordan, Thomas R Bromley, Juan Miguel Arrazola, and Nathan Killoran. 2023. Fast quantum circuit cutting with randomized measurements. *Quantum* 7 (2023), 934.

[26] Urmila Mahadev. 2020. Classical homomorphic encryption for quantum circuits. *SIAM J. Comput.* 0 (2020), FOCS18–189.

[27] Tomoyuki Morimae. 2012. Continuous-Variable Blind Quantum Computation. *Physical Review Letters* 109, 23 (dec 2012). https://doi.org/10.1103/physrevlett.109.230502 https://arxiv.org/pdf/1208.0442.pdf.

[28] National Institute of Standards and Technology. 2024. *Transition to Post-Quantum Cryptography Standards*. NIST Internal Report 8547. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.IR.8547.ipd

[29] National Institute of Standards and Technology (NIST). 2015. Secure Hash Standard (SHS). https://doi.org/10.6028/NIST.FIPS.180-4. FIPS PUB 180-4.

[30] National Institute of Standards and Technology (NIST). 2015. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. https://doi.org/10.6028/NIST.FIPS.202. FIPS PUB 202.

[31] National Institute of Standards and Technology. 2015. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. https://doi.org/10.6028/NIST.FIPS.202 Accessed: 2025-03-24.

[32] Openquantumhardware. [n. d.]. tProcessor-64 and Signal Generator V4. https://github.com/openquantumhardware/qick/blob/main/firmware/tProcessor_64_and_Signal_Generator_V4.pdf Accessed: 2025-03-18.

[33] Matteo Paltenghi and Michael Pradel. 2022. Bugs in Quantum computing platforms: an empirical study. *Proceedings of the ACM on Programming Languages* 6, OOPSLA1 (2022), 1–27.

[34] Tianyi Peng, Aram W. Harrow, Maris Ozols, and Xiaodi Wu. 2020. Simulating Large Quantum Circuits on a Small Quantum Computer. *Physical Review Letters* 125, 15 (oct 2020). https://doi.org/10.1103/physrevlett.125.150504 https://arxiv.org/pdf/1904.00102.pdf.

[35] Michael A Perlin, Zain H Saleem, Martin Suchara, and James C Osborn. 2021. Quantum circuit cutting with maximum-likelihood tomography. *npj Quantum Information* 7, 1 (2021), 64.

[36] Quantinuum. [n.d.]. Quantinuum. https://www.quantinuum.com/ Accessed: 2025-03-18.

[37] Rigetti. [n.d.]. Rigetti Computing. https://www.rigetti.com/ Accessed: 2025-03-18.

[38] Amazton Web Services. [n.d.]. Amazon Braket. https://aws.amazon.com/de/braket/ Accessed: 2025-03-18.

[39] Peter W Shor. 1994. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*. Ieee, 124–134.

[40] Leandro Stefanazzi, Kenneth Treptow, Neal Wilcer, Chris Stoughton, Collin Bradford, Sho Uemura, Silvia Zorzetti, Salvatore Montella, Gustavo Cancelo, Sara Sussman, et al. 2022. The QICK (Quantum Instrumentation Control Kit): Readout and control for qubits and detectors. *Review of Scientific Instruments* 93, 4 (2022).

[41] Thomas Strohm, Karen Wintersperger, Florian Dommert, Daniel Basilewitsch, Georg Reuber, Andrey Hoursanov, Thomas Ehmer, Davide Vodola, and Sebastian Luber. 2024. Ion-Based Quantum Computing Hardware: Performance and End-User Perspective. *arXiv preprint arXiv:2405.11450* (2024).

[42] Takahiro Sueki, Takeshi Koshiba, and Tomoyuki Morimae. 2013. Ancilla-driven universal blind quantum computation. *Physical Review A—Atomic, Molecular, and Optical Physics* 87, 6 (2013), 060301. https://doi.org/10.1103/PhysRevA.87.060301

[43] Naoki Takeuchi, Taiki Yamae, Taro Yamashita, Tsuyoshi Yamamoto, and Nobuyuki Yoshikawa. 2024. Microwave-multiplexed qubit controller using adiabatic superconductor logic. *npj Quantum Information* 10, 1 (2024), 53.

[44] Si-Hui Tan, Joshua A Kettlewell, Yingkai Ouyang, Lin Chen, and Joseph F Fitzsimons. 2016. A quantum approach to homomorphic encryption. *Scientific reports* 6, 1 (2016), 33467.

[45] Theodoros Trochatos, Sanjay Deshpande, Chuanqi Xu, Yao Lu, Yongshan Ding, and Jakub Szefer. 2024. Dynamic Pulse Switching for Protection of Quantum Computation on Untrusted Clouds. In *International Symposium on Hardware Oriented Security and Trust (HOST)*.

[46] Theodoros Trochatos, Chuanqi Xu, Sanjay Deshpande, Yao Lu, Yongshan Ding, and Jakub Szefer. 2023. A quantum computer trusted execution environment. *IEEE Computer Architecture Letters* 22, 2 (2023), 177–180.

[47] George Typaldos, Wei Tang, and Jakub Szefer. 2024. Leveraging Quantum Circuit Cutting for Obfuscation and Intellectual Property Protection. In *International Conference on Quantum Computing and Engineering (QCE)*.

[48] Christian Ufrecht, Laura S Herzog, Daniel D Scherer, Maniraman Periyasamy, Sebastian Rietsch, Axel Plinge, and Christopher Mutschler. 2024. Optimal joint cutting of two-qubit rotation gates. *Physical Review A* 109, 5 (2024), 052440.

[49] Yilun Xu, Gang Huang, Neelay Fruitwala, Abhi Rajagopala, Ravi K. Naik, Kasra Nowrouzi, David I. Santiago, and Irfan Siddiqi. 2023. QubiC 2.0: An Extensible Open-Source Qubit Control System Capable of Mid-Circuit Measurement and Feed-Forward. arXiv:2309.10333 [quant-ph] https://arxiv.org/abs/2309.10333

[50] Songqinghao Yang and Prakash Murali. 2024. Understanding the Scalability of Circuit Cutting Techniques for Practical Quantum Applications. *arXiv preprint arXiv:2411.17756* (2024).