A Computer Aided Design Tool for NISQ Logic Synthesis

Albert B Dinkins V
Department of Electrical Engineering
University of South Florida
Tampa, USA
adinkins1@usf.edu

Austin Bristow
Department of Electrical Engineering
University of South Florida
Tampa, USA
bristow@usf.edu

Kwang-Cheng Chen
Department of Electrical Engineering
University of South Florida
Tampa, USA
kwangcheng@usf.edu

***Abstract*—Computed Aided Design (CAD) is essential to synthesize noisy intermediate scale quantum (NISQ) information and communication systems. The prevailing model for NISQ systems is gate-based and uses logical components arranged into a circuit to represent different quantum operations. A CAD Tool for Quantum Logic is proposed to synthesize gate-based NISQ circuits for computing, communication, or sensing by supplying designers with parameterized versions of common quantum algorithms as well as a design space for simulating photonics hardware implementations. Available in a software package called the CADTQL, designers can test their conventional and optical quantum logic configurations and realize more efficient designs.**

***Index Terms*—CAD, Quantum Logic Synthesis, Quantum Optics and Communications, NISQ**

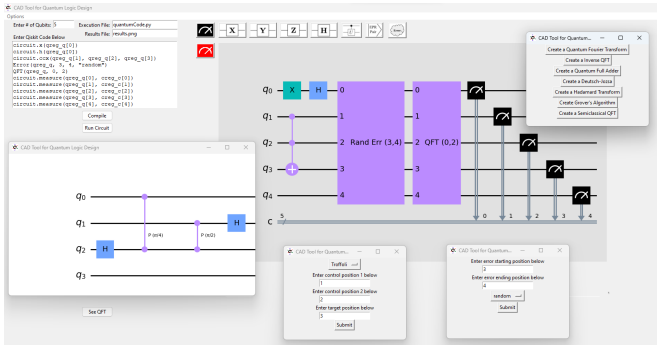

Fig. 1. Sample Frame of The CAD Tool for Quantum Logic - a design space for NISQ quantum logic synthesis

I. Introduction

Quantum logic synthesis plays a critical role in NISQ gate-based quantum information systems for computing, communication, and sensing [1] [2] [9]. The successful construction of physical quantum gates and the preservation of their effects is important to the advancement of integrated quantum solutions such as new algorithms, processors, or operating systems [3]. Quantum logic - a nonphysical representation for quantum computations and algorithms - and subsequent synthesis serve as a guide for scientists seeking highly efficient quantum solutions with applications that span from communications to cryptography or computer architectures [6]. With the help of computer aided design (CAD) technology, we can create a unified space for synthesizing quantum logic with design capabilities fit to specific quantum physical implementations, enabling the eventual construction of application specific quantum devices [4].

This task of developing meaningful quantum algorithms can be achieved through the refinement of current NISQ methodologies [5]. Current methodologies for specific physical quantum implementations, including Strawberry Fields (modeling photonic quantum configurations) [14], the IBM Quantum Composer (modeled after semiconducting configurations) [15], SQUANCH (modeling quantum networks and channels) [16], and QuISP (simulating Quantum Internet) [17], all exist to model individual design approaches to quantum information systems. As current tools are limited to particular implementations, an opportunity arises to provide quantum circuit designers with a more general implementation of optical quantum computing (polarization, spatial, time, etc.) and the ability to compare different physical quantum circuit design choices. To respond to such a challenge, this paper introduces the development of a quantum-assistive software, called the "Computer Aided Design Tool for Quantum Logic" (CADTQL), built to synthesize logical quantum circuitry and simulate the execution of any quantum circuit configuration.

In order to fully describe the CADTQL, we outline (i) the different capabilities of the tool (i.e., simulation and parameterization) and their rationale, (ii) the development approach for these capabilities, (iii) actual methods and challenges during development, (iv) results and computational experiments using the tool, and (v) the importance of the CAD tool to the future of physical quantum circuit-based devices used in computing, communications, and sensing.

II. Designing the CADTQL

There are two main components of the CADTQL - the "Quantum Conventional Logic Design Space" (QCLDS) and the "Photonic Logical Design Space" (PLDS). The development of the QCLDS was conducted with the intention of building a framework for quantum logic in a manner similar to IBM's Quantum Composer such that, when the time for application specific design capabilities to be introduced to the tool (like the PLDS), the techniques and features can be easily replicated and expanded. The PLDS, on the other hand, uses

optical hardware configurations and specific physical inputs like angles of rotation and light wavelength to model quantum circuitry. In quantum computing, it is important to be able to model a system around its physical implementation as each method comes with its own unique obstacles - nonlinear optical configurations struggle to support controlled rotations, superconducting processors require near 0K conditions, etc. [7] [8] [12]. So, we design a more general CAD tool which allows you to simulate quantum circuits dependent on physical parameters. Additionally, having a CAD tool specifically for photonics, as opposed to other implementations, is important for two main reasons: first, communications systems worldwide depend on the transmission of light at different wavelengths, and second, generating/manipulating light is relatively inexpensive and accomplished at room temperature [18] [19] [20]. Through the availability of the PLDS, we aim to impact the design of quantum communications and sensing systems, which already greatly depend on photonics [16] [17].

Together, the QCLDS and PLDS provide interested parties with the opportunity to (i) build and test complicated quantum logic using an effective and designer-friendly interface, (ii) execute parameterized versions of well-known quantum algorithms (developed "in-house" to support a more simplistic and educational method for using quantum algorithms), and (iii) implement custom logical designs for physical quantum circuitry in "design spaces". This paper introduces a version of the CADTQL that lays the groundwork for a holistic CAD tool capable of synthesizing realistic quantum application specific designs that are important to wireless communications implementations (i.e., quantum inner and outer communications receivers, remove quantum sensing devices, etc.) through the ability to integrate conventional logic synthesis with their hardware equivalents across different physical design choices, like photonics.

A. QCLDS Design Philosophies

One of the core design aspects of the QCLDS is the leveraging of the quantum computing devices harbored by IBM through a Python-based software installation of their BasicAER simulator captured in the Qiskit Software Development Kit (SDK). This SDK is capable of simulating quantum computations modeled after physical quantum computers with generally low error rates [10]. To follow the Python basis of Qiskit, the user interface for the tool was also developed in Python using the Tkinter UI library. With a front end written in Python, the ability to easily integrate back-end Qiskit calls with the TKinter UI made for a fluid software development process entirely in Python.

Python's functionality scheme further promotes the ability to parameterize common quantum algorithms through function calls and file write statements. Qiskit already has several parameterized versions of algorithms available, particularly the Deustch-Josza and Grover's algorithms, but their input parameters can become incredibly complex and some are even depreciated, making for a difficult interaction process for designers. The IBM Quantum Composer, which also leverages

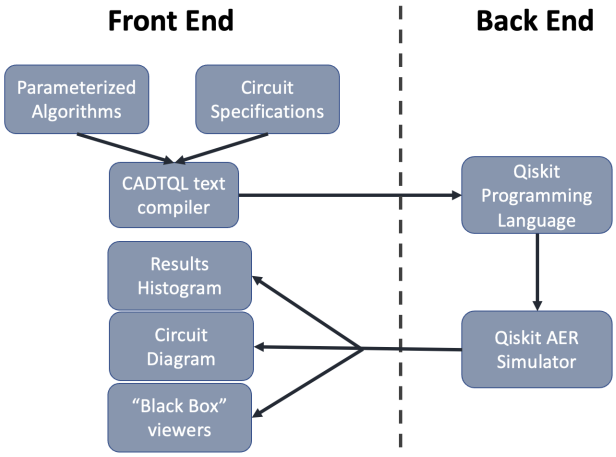

Fig. 2. Design Map for the CADTQL

Qiskit, lacks the ability to manipulate Qiskit code alongside the composer's diagramming. It also does not provide users with the ability to easily integrate the algorithms already offered by Qiskit. If a designer wants to implement an algorithm into the Quantum Composer, they must learn the syntax and also understand all of the parameters, which may or may not be depreciated. As a result, all algorithms developed for the CADTQL - no matter whether the algorithm already existed in Qiskit or not - were redefined from the ground up in an effort to allow seamless integration into circuit design, to promote an algorithm driven design space, and to also build the framework for future design spaces.

B. PLDS Design Philosophies

Wireless quantum communication systems are primarily implemented through optics and the PLDS offers a design space to work directly with physical devices that are vital to quantum optics. Optical implementations of quantum communications systems introduce several non-trivial problems which are often neglected in logic synthesis. First, realistic optical implementations use non-deterministic single photon sources, imperfect optical components, and single photon detectors with $<100\%$ quantum efficiency. This can result in loss or

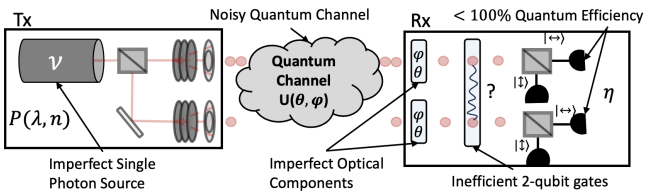

Fig. 3. Challenges in Optical Quantum Communications Systems

small angle error on the Bloch sphere. Second, quantum channels can introduce unitary and non-unitary errors that must be accounted for in order to have an effective communication system. Finally, the practical implementation for 2-qubit optical gates are difficult to implement as photons react poorly

to interactions with other photons and their environment. By integrating physical device parameters into the PLDS, we can more appropriately and exactly address the cause-and-effect of the above problems through iterative designing, simulating these designs, and synthesizing results to come up with a next best implementation.

III. Development of the CADTQL

A. Foundational Components

The core of the application essentially operates as a compiler of designer inputs in order to present (i) a circuit diagram representing the code-based implementation provided by the designer (a diagram-to-code based approach is offered in another section of the application, accessed from the file menu), (ii) a results histogram in the presence of measurement, again, as dictated by the designer, and (iii) the ability to view underlying implementations of parameterized algorithms presented as a "black box" or "oracle". To present these three things, the application relies on designer input to a large white text box, located below entry fields for setting the number of qubits in the circuit being designed, the name of the file the designer wants to save their results to, and the name of the program file that will be sent to IBM's simulator for execution.

B. Developing the QCLDS and Parameterized Algorithms

In the QCLDS's driver file, various python functions are written to capture (i) the algorithm a designer wants to use, (ii) how large the algorithm needs to be, (iii) which qubits the algorithm needs to be applied to, and (iv) any other details that may be important in the construction of the algorithm. For all algorithms parameterized as part of this research, the main method for configuring these algorithms came in the form of Python file write statements that would explicitly define the underlying algorithm for the quantum simulator to process. From a front-end perspective, every algorithm is listed within a central menu panel, allowing users to choose any algorithm they'd like to with the click of a button.

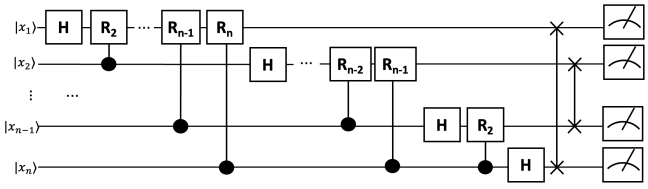

Fig. 4. QFT Circuit Diagram

Two important algorithms parameterized in the QCLDS are the Quantum Fourier Transform Algorithm (QFT) and its inverse (iQFT). These algorithms are subroutines in notable quantum algorithms like Shor's Algorithm and the Quantum Phase Estimation Algorithm. Shor's algorithm, which uses the QFT and Inverse QFT as subroutines in its factorization of prime numbers, shows great promise in its ability to break public-key encryption schemes while Phase Estimation plays an important role in communications and error correction.

When constructing the parameterized Quantum Fourier Transform, we must explicitly define what the circuit should logically accomplish and then implement the correct unitary transformations in terms of their logical Qiskit implementation. Accordingly, the quantum Fourier Transform maps a vector in the complex number space to another vector in the complex number space according to the formula:

$$|x_1 x_2 ... x_n\rangle \xrightarrow{\text{QFT}} \frac{1}{2^{n/2}} \sum_{y=1}^{2^n-1} e^{i2\pi \cdot x \cdot y/2^n} |y_1 y_2 ... y_n\rangle$$

where n is the total number of input qubits, and x = x1, x2, ... xn and y = y1, y2, ... yn are the binary representations of the input and output qubits, respectively [13]. This formula results in the circuit diagram in Figure 4 where the boxes labeled R_k with the qubit connected via the vertical line represents a controlled-phase gate by the matrix:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\pi 2/2^k} \end{bmatrix}$$

Two subroutines were constructed for the parameterized QFT: one function was written to recursively apply the Hadamard gates/controlled rotations (called "qft_rotations()") and another was written for swapping the qubit registers (called "swap_registers()") as is convention when using Qiskit. The recursive algorithm takes a string of n qubits (can be positioned anywhere in the circuit) and, starting at the n^{th} qubit of the subroutine, applies a Hadamard gate to this qubit and n-1 controlled rotations. The n^{th} qubit is the control and the rotations are given by $\pi/2^x$ where x=1...$n-1$. This process recursively decrements through the registers until reaching the final qubit of the algorithm, upon which only a Hadamard gate needs be applied and the circuit is returned. The rotations for the Inverse Quantum Fourier Transform was defined similarly to match the Inverse Quantum Fourier Transform formula.

After combining the rotations subroutine with the swap registers function, the installation of the parameterized QFT and iQFT functions is completed. When any function is called in the code entry box (as seen in Figure 5), the application will send the exact parameterized function definitions to a quantum computer to be simulated. To comprehend the results the quantum computer returns, a series of UI elements (Figure 5) help to fully illustrate what exactly the algorithm is accomplishing and aide in further synthesis.

Beyond the Quantum Fourier Transform, a few other quantum computing solutions exist in the tool. Grover's Algorithm, which provides a quantum solution to the unstructured/unsorted search problem in a quadratic speedup over classical algorithms, essentially searches for the quantum states marked with a negative phase by a quantum oracle and returns the marked state after measuring all qubits post algorithm processing [2]. In terms of parameterizing Grover's Algorithm, there are three components to be constructed: the Hadamard Transform to all n qubits in the algorithm, an

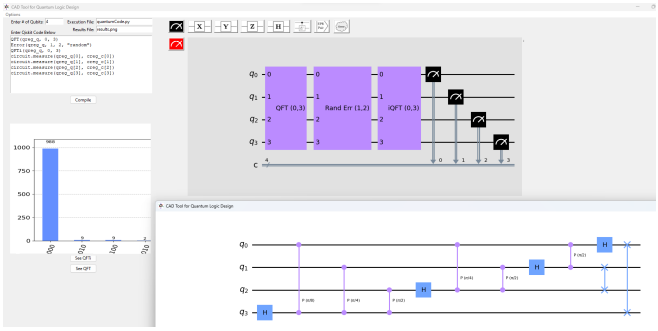

Fig. 5. Creation of the Inverse and Discrete Quantum Fourier Transform in the CADTQL

oracle dictating which state represents the item being searched for, and a diffuser that will cause the circuit's final state to represent the marked state. The oracle's details are left up to the designer to implement as this is how the state to be searched for is initialized. The final component, the diffuser, is built to diffuse a Grover's Algorithm circuit of any size, accomplishing the core parameterization aspect of this algorithm.

Another algorithm included in the tool that is dependent on additional designer input is the Deutsch-Jozsa algorithm. Beyond the automatic application of Hadamard Transform to the first n qubits and the preparation of the ancilla bit, the algorithm also preps a balanced or constant oracle based off user input. Finally, a parameterized quantum full adder algorithm was also included in the system as to serve as the arithmetic module for potential quantum chip designs.

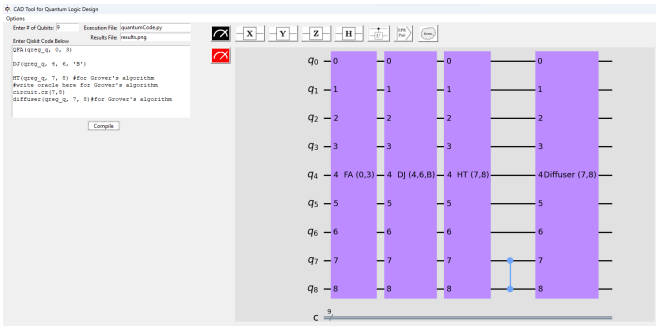

Fig. 6. Sample frame depicting the creation of the parameterized Quantum Full Adder, Grover, and Deutsch-Josza Algorithms all configured alongside each other and applied to different qubits - exemplifying the ability to truly apply algorithms in any arrangement

C. Developing the Photonic Logical Design Space

While parameterization is certainly a foundational aspect of the CADTQL, the ability to construct logical implementations of custom physical circuit configurations and run them against a quantum simulation remains a challenge. We first propose a completely new visualization for designing quantum optical circuits while also providing realistic simulations. The PLDS mimics the the basic structure of IBM's Quantum Composer but for optics: visual representations of the logical components, a drag and drop system for the circuit visuals, and a grid system for configuring the circuit. This design space will permit novel developments in designing application specific quantum devices based in photonics.

A quantum computation first requires the encoding of classical or binary information into qubits as a preprocessing step to any sort of data manipulation [1] [7] [11]. Classical data is often encoded into the physical properties of the qubit and can be manipulated by a series of operations that act on the qubits' physical properties to process the encoded classical information. The PLDS operates on free space photon qubits where horizontally polarized photons are considered to be the quantum logical representation of a binary "0" and vertically polarized photons are considered to be a "1" [7] [8]. The photons used for PLDS computations are delivered from a laser capable of operating at varying wavelengths, subject to the designers choice. This laser, coupled with a laser attenuator and lens, would act as the register initialization for the quantum circuit. From there, the free space light can be processed by quarter-wave plates, half-wave plates, and electro-optic modulators to create and process quantum states, all before measurement in the form of a particle beam splitter and corresponding detectors [7] [8].

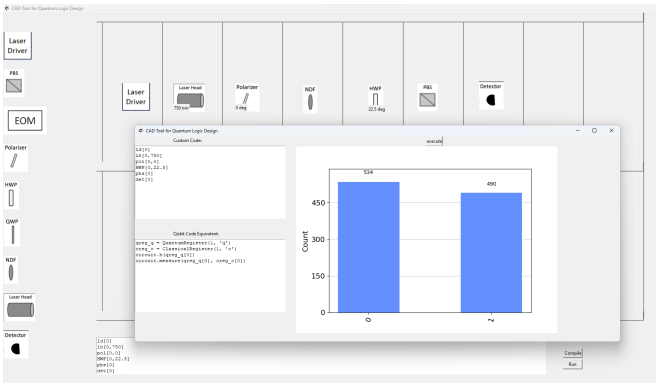

Fig. 7. Circuit layout and code conversion panels specific to the Photonic Logical Design Space

The PLDS is built to support any circuit size from 1x2 to 9x17. Components of the circuit that depend on physical parameters were given data structures to store necessary information related to the circuit piece. For example, lasers can be set to any wavelength, wave plates can be rotated to any degree, laser drivers have a repetition frequency, etc. As photonic gates and their parameters are dragged and dropped into the circuit grid, a matrix representation of the circuit is compiled such that each cell in the matrix corresponds to an OpenQuasm-like language independently developed in this research specifically for photonic set ups. For example, a laser emitting 750 nm light would find its entry in the matrix to be "LH[pos, 750]" where "pos" is the qubit register position initiated by this laser head. After creating similar parameterizations for other components, the system supports

a fully customizable designs for physical photonic quantum circuits.

IV. Using the CADTQL

Having covered the implementation and capabilities of the CADTQL, this section of the paper seeks to demonstrate the application's use process through a series of examples.

A. Conventional Logical Design

First we'll begin with the implementation process for configuring the 3 parameterized algorithms depicted in Figure 6. First the circuit needs to have its registers initialized: this is accomplished by entering a numeric value in the text entry labeled "Enter # of Qubits". Following the order in Figure 6, a designer can configure the quantum fuller adder by applying it to qubits 0-3 (resulting in the addition of qubits 0 and 2 with the result stored in qubit 3 and the carry out in qubit 1). If a designer adds code to apply an X gate to qubit 0 and an identity gate to qubit 2 before the QFA algorithm, they have created a quantum logical representation of a quantum algorithm that computes the addition of "1" and "0". By executing the circuit after applying measurement to qubit 1 for the carry out and qubit 3 for the sum, the designer is presented with a histogram containing the quantum solution for adding the basis quantum state found in qubit 0 to its complement found in qubit 2. Compiling and executing the circuit construction thus far results in the frame captured in Figure 8.

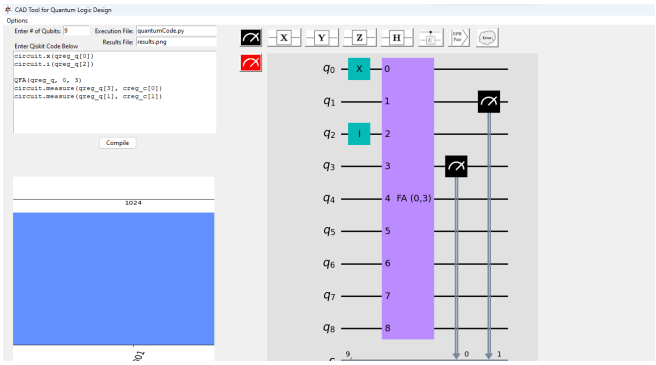

Fig. 8. QFA implementation before adding more algorithms

Following the same steps and prescribing parameters as necessary, we can create an extended circuit that makes use of other algorithms by continuing to append code to the main code entry text box. We can add the Deustch Jozsa to qubits 4-6 and automatically configure a balanced oracle by sending the "B" character to the last argument in the DJ function. Due to the fact that the last qubit of every Deustch Jozsa implementation is an ancilla bit, only qubits 4 and 5 need be measured to obtain the algorithm's result. With Grover's algorithm applied to qubits 7 and 8, we can implement our own custom oracle capable of marking the quantum state 11 (as opposed to 01, 01, or 00). This oracle is realized by a controlled z gate where qubit 7 is the control and qubit 8 is the target. Measurement of qubits 7 and 8 reveals the "marked state". The final cumulative quantum circuit is depicted in Figure 9.

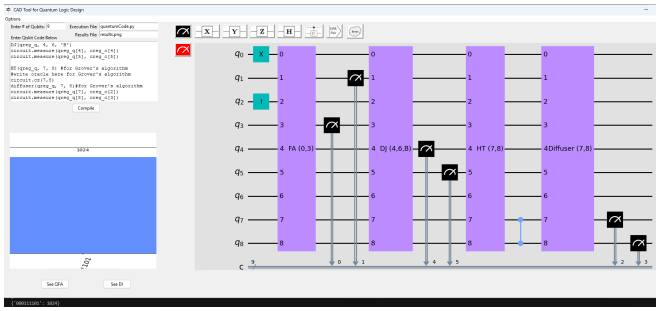

Fig. 9. Result of running the three parameterized algorithms together. Note the classical result in the bottom left of the image is the reversed order of the outcome we expect - this is a qiskit convention

To obtain the histogram, we must send each of the measurements to classical registers that correspond to our classical encoding of information. The results from each of the 6 measurements we conduct (2 for the QFA on qubits 0 and 3, 2 for the DJ on qubits 4 and 5, and, finally, 2 on Grover's Algorithm for qubits 7 and 8) can be distributed to 6 classical bits that construct our results histogram. If we distribute the results across the classical registers in the order presented (q0-c1, q3-c2, q4-c3, q5-c4, q7-c5, q8-c6), we have a classical binary string that represents what the circuit returns. The logic of the circuit is found in the following: the binary addition of 2 bits where one is "0" and the other is "1" results in a sum of "1" and a carry out of "0", making the first two expected outcomes of the circuit to always be "10". The Deutsch-Jozsa algorithm should always return a string of 1s for a balanced function, so the next two expected outcomes are "11". Finally, since we marked the 11 state for Grover's Algorithm, we expect "11" for the final two outcomes. Therefore, for every run of this circuit - for all 1024 executions - we expect a results histogram with 1024 counts of 101111; we find this to be the case as displayed in Figure 9.

B. Photonic Logical Design

The Photonic Logical Design Space, like the QCLDS, is also capable of having its configurations simulated, allowing users to evaluate their designs. The device diagrams used within a particular circuit design are compiled into a list of instructions specific to the implementation involved before being translated into Qiskit to be ran. The circuit in Figure 7 depicts the physical optical implementation of a Hadamard gate and measurement, which can be ran against IBM's simulators to return the physical implementation's expected results.

When a designer opens the PLDS and sets the size of the circuit (see that Figure 7 contains a 2x9 circuit where only the first row is used), they can begin dragging and dropping icons into any space in the grid they'd like. Once done, the designer can hit a "compile" button that will take all the

diagrams in the grid and convert them to custom code (e.g., "HWP[2]" or "POL[3]"). In the future, it is possible that these custom code commands can be used as input to a simulator built specific to the physical device as opposed to leveraging Qiskit's BasicAer Simulator. With this concept implemented, designers would notice different results for 750 nm vs 850 nm light or differences based on attenuation - these are things most logic simulators do not currently have a way to account for.

If a detector ("DET[...]") is found in the grid, the option to "Run" the circuit is presented to the designer. When pressed, a pop-up window illustrates the custom code to qiskit translation while also presenting the designer with the option to execute their newly crafted qiskit code. Upon clicking "execute" the results for running the Qiskit code are presented. The Qiskit code in the "Qiskit Code Equivalent" box is completely customizable and malleable - it even runs the same parameterized algorithms available in the QCLDS. This means we can test the "Error" function (also installed as a button in the QCLDS UI) on the just-implemented physical set up, creating an even more realistic model (see Figure 10 where the "Error" function is added to the "Qiskit Code Equivalent" entry box).

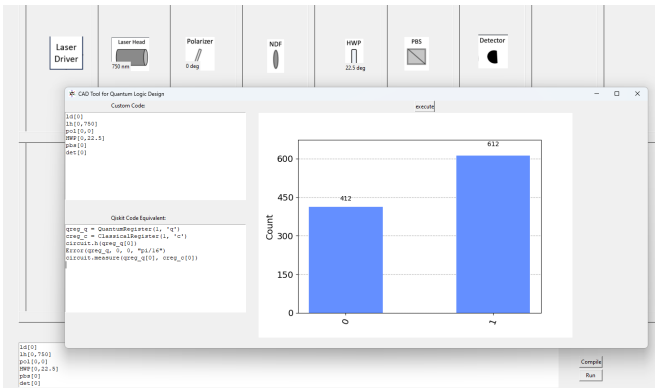

Fig. 10. Manipulating Qiskit to introduce realistic error models for a physical photonic implementation

With a design space that is capable of simulating specific logical configurations of physical devices, computing, communications, and sensing researchers can investigate the results of their physical implementations, test for where error may corrupt the computation, and evaluate how their circuit can be improved all in one centralized software package.

V. Conclusion

This paper has discussed the design philosophy behind, and the capabilities of, a CAD tool for quantum logic synthesis. The CADTQL offers design spaces for synthesizing quantum logic and illustrating the results of quantum computing, communications, and sensing circuits. The tool offers a simple interface for configuring quantum circuitry at the gate level, as gate-based quantum computing is the typical model for NISQ solutions today. The CADTQL serves to further push quantum circuit construction toward more algorithm intensive and application specific designs. Designers can create any circuit configuration they desire, with parameterized versions of famous quantum algorithms made readily available in the form of custom-built function calls. Designers can execute the circuits on IBM supported quantum simulators and view results with ease. We present these features as a means of enabling more complex quantum systems like quantum processors or communications sub-systems.

In subsequent research, we plan to incorporate additional optical encodings into the Photonic Logical Design Space, in addition to other non-optical encoding possibilities in new design spaces. This will enable the generation and simulation of tailored setups for any specific encoding, thereby enhancing the design of all types of physical quantum circuits.

References

[1] P. Kaye, R. Laflamme, and M. Mosca, *An Introduction to Quantum Computing*, Oxford University Press, 2007.

[2] J. D. Hidary, *Quantum Computing: An Applied Approach*, Springer, 2021.

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

[4] S. Y. Kuo, Y. C. Jiang, Y. H. Chou, S. Y. Kuo, and S. Y. Kung, "Quantum Computer-Aided Design Automation," *IEEE Nanotechnology Magazine*,vol. 17, no. 2, pp. 15-25, 2023.

[5] F. Leymann and B. Johanna, "The bitter truth about gate-based quantum algorithms in the NISQ era," *Quantum Science and Technology*,vol. 5, no. 4, 2020.

[6] J. Lee, Y. Chung, J. Kim, and S. Lee, "A Practical Method of Constructing Quantum Combinational Logic Circuits," arXiv:quant-ph/9911053, Nov. 1999.

[7] C. Adami amd N. J. Cerf, "Quantum Computation With Linear Optics," arXiv:quant-ph/9806048, Jun. 1998.

[8] C. Adami, N. J. Cerf, and P. G. Kwiat, "Optical Simulation of Quantum Logic," arXiv:quant-ph/9706022, Jun. 1997.

[9] V. Shende, S. Bullock, and I. Markov. "Synthesis of Quantum Logic Circuits," arXiv:quant-ph/0406176, Apr. 2006.

[10] "Qiskit Documentation," qiskit.org. https:// qiskit.org/documentation. (accessed Sept 15, 2023).

[11] J. Lukens and P. Lougovski, "Frequency-encoded photonic qubits for scalable quantum information processing," arXiv:2210.11830 [quant-ph], Oct. 2022.

[12] T. Roy, Z. Li, E. Kapit, and D. I. Schuster, "Realization of two-qutrit quantum algorithms on a programmable superconducting processor," arXiv:2211.06523 [quant-ph], Nov. 2022.

[13] Y. Shi and E. Waks, "Quantum Fourier transform on photonic qubits using cavity QED," arXiv:2112.00658, Jul. 2022.

[14] N. Killoran, J. Izaac, N. Quesada, V. Bergholm, M. Amy, and C. Weedbrook, "Strawberry Fields: A Software Platform for Photonic Quantum Computing," arXiv:1804.03159v2 [quant-ph], Mar. 2019.

[15] "Composer User Guide," ibm.com. https://learning.quantum-computing.ibm.com/tutorial/composer-user-guide. (accessed Sept 15, 2023).

[16] B. Bartlett, "A Distributed Simulation Framework for Quantum Networks and Channels," arXiv: 1808.07047 [quant-ph], Aug. 2018.

[17] R. Satoh, M. Hajdušek, N. Benchasattabuse, S. Nagayama, K. Teramoto, T. Matsuo, S. A. Metwalli, T. Satoh, S. Suzuki, and R. Van Meter, "QuISP: a Quantum Internet Simulation Package," arXiv:2112.07093 [quant-ph], Dec. 2021.

[18] J.E. Bourassa, R.N. Alexander, M. Vasmer, A. Patil, I. Tzitrin, T. Matsuura, D. Su, B.Q. Baragiola, S. Guha, G. Dauphinais, and K.K. Sabapathy, "Blueprint for a scalable photonic fault-tolerant quantum computer," *Quantum*, vol. 5, p. 392, 2021.

[19] J.W.Z. Lau, K.H. Lim, H. Shrotriya, and L.C. Kwek, "NISQ computing: where are we and where do we go?," *AAPPS Bulletin*, vol. 32, no. 1, p. 27, 2022.

[20] J.E. Bourassa, R.N. Alexander, M. Vasmer, A. Patil, I. Tzitrin, T. Matsuura, D. Su, B.Q. Baragiola, S. Guha, G. Dauphinais, and K.K. Sabapathy, "Blueprint for a scalable photonic fault-tolerant quantum computer," *Quantum*, vol. 5, p. 392, 2021.