QuaCiDe: A General Purpose Quantum Circuit Design and Simulation Interface

Asimakis Kydros
Aristotle University of Thessaloniki
Department of Informatics
Thessaloniki, Greece
asimakis@csd.auth.gr

Konstantinos Prousalis
Aristotle University of Thessaloniki
Department of Informatics
Thessaloniki, Greece
kprousalis@csd.auth.gr

Nikos Konofaos
Aristotle University of Thessaloniki
Department of Informatics
Thessaloniki, Greece
nkonofao@csd.auth.gr

ABSTRACT

A circuit design interface for simulation of quantum circuits is presented. This simple but powerful tool aims to enable and improve experimentation in the field of quantum computing, either for professional or amateur usage. It provides an intuitive, automated, easy-to-use environment for building algorithms using the circuit model of quantum computation, while freeing the experimenter from technical procedures, such as coding or program installations. It provides a wide range of tools to the user, from a universal gate-set, to custom gate definitions, post-selected measurements, complex compositions of sub-circuits in a recursive way, good modularity and nice output results. Moreover, it is combined with popular backend simulators due to its circuit-parsing capabilities.

CCS CONCEPTS

• **Hardware → Quantum computation**; *Physical design (EDA)*; • **Human-centered computing → Visualization design and evaluation methods**.

KEYWORDS

Circuit designer, EDA tool, Quantum circuit, Quantum computation, Quantum simulation

ACM Reference Format:
Asimakis Kydros, Konstantinos Prousalis, and Nikos Konofaos. 2024. QuaCiDe: A General Purpose Quantum Circuit Design and Simulation Interface. In *Recent Advances in Quantum Computing and Technology (ReAQCT '24), June 19–20, 2024, Budapest, Hungary.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3665870.3665874

1 INTRODUCTION

Quantum Computing is a promising and ever-growing field of Sience, that benefits from -and therefore unites- a wide margin of other fields, such as Physics, Chemistry, Engineering and Informatics. Due to the still uncertain physical nature of quantum computers, the classical simulation of quantum algorithms is of great benefit and significance.

Table 1: Features comparison between existing quantum designers and this work.

Features	**Selected Circuit Design Interfaces**							
	JQuantum	QCS	QuCirDET	IBM QC	Quirk	Qiskit	QuEST	QuaCiDe
Live Hosting	✗	✗	✗	✓	✓	✗	✗	✓
Drag and drop	✗	✗	✗	✓	✓	✗	✗	✓
Unlimited qubits	✗	✗	✓	✗	✗	✓	✓	✓
Intermediate measuring	✓	✗	✗	✓	✓	✓	✗	✓
Postselection	✗	✗	✗	✗	✓	✗	✓	✓
Sub-circuits	✗	✗	✗	✓	✓	✓	✗	✓
Output visualization	✗	✓	✓	✓	✓	✓	✗	✓
Circuit stat monitoring	✗	✗	✓	✗	✗	✗	✗	✓
Backend independence	✗	✗	✗	✗	✗	N/A	N/A	✓

A few such simulation tools already exist. While all of them are adept works, each of them include certain quirks: of the first to employ graphical user interfaces, JQuantum [1], QCS [7] and QuCirDET [9] are desktop applications and therefore require installation and system support; Quirk [4] and IBM Quantum Composer [6] are exceptional modern online designers but they limit themselves on the number of qubits; other solutions, such as Qiskit [12] and QuEST [13] are programming libraries and therefore require programming knowledge. These technicalities can cost valuable time to researchers or even become deal-breakers for those who lack the demanded expertise or don't wish to accept the extra requirements.

This Quantum Circuit Designer -abbreviated as *QuaCiDe*, read as *Cassidy*- aims to emancipate the process of simulating quantum experiments from programming and complex installations, by providing a web-hosted, simple to use graphical user interface that allows the creation of arbitrarily complex and scaled circuits using drag-and-drop philosophy. A number of quality-of-life mechanisms are also employed, such as input shortcuts, shift automations, stat counters and more. All these sub tools aim to improve the user experience and allow for faster experimental work and the smoother learning of Quantum Computing to newer audiences.

Table 1 compiles the most prevalent improvements on features among the listed existing designers and this work. It is important that such a designer be a live hosted web service that allows the experimenter to jump in and start work immediately without requiring further effort or knowledge; in the same spirit, allowing the creation of circuits by dragging and dropping gates onto qubits is also a desired quality, one that frees the individual from programming details. An unrestricted amount of qubits, measurements in the middle of the circuit and postselected outcomes are necessary tools for the qualitative analysis of quantum algorithms and ought to be available to the user. A sub-circuit mechanism that allows the creation of custom quantum gates by user defined circuits is a required asset, to allow the user to encapsulate logic according

ReAQCT '24, June 19–20, 2024, Budapest, Hungary
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-1798-7/24/06
https://doi.org/10.1145/3665870.3665874

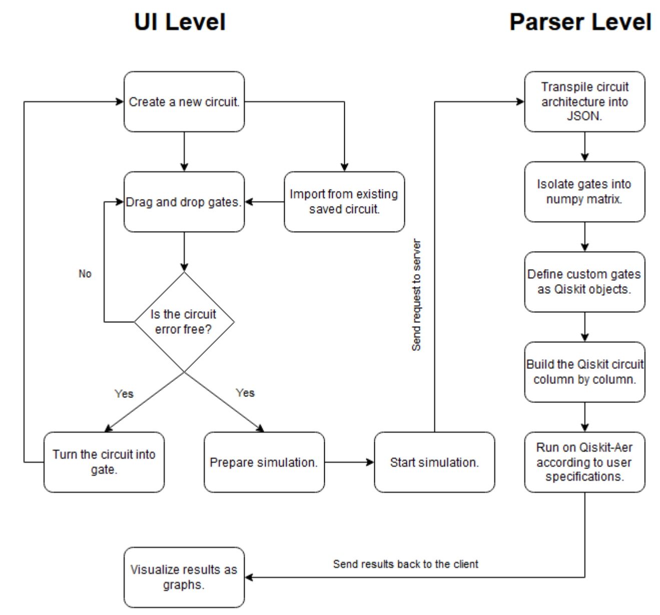

UI Level
Parser Level
Create a new circuit.
Import from existing saved circuit.
Drag and drop gates.
Is the circuit error free?
No
Yes
Yes
Turn the circuit into gate.
Prepare simulation.
Start simulation.
Send request to server
Transpile circuit architecture into JSON.
Isolate gates into numpy matrix.
Define custom gates as Qiskit objects.
Build the Qiskit circuit column by column.
Run on Qiskit-Aer according to user specifications.
Send results back to the client
Visualize results as graphs.

Figure 1: Abstract work flow of a single experiment session.

to their needs. The visualization of the output and the monitoring of circuit resources, such as number of used qubits and gates, are useful additions that elevate the user experience and help them better understand the nature of their work. Finally, keeping the user interface independent from specific computational backends allows it to be easily adaptable to new technologies, but also flexible to each simulator's requirements.

In the following paragraphs, the architecture and capabilites of this tool are described. Section 2 outlines its base architecture, while Sections 3 and 4 discuss the main interface and the parser, respectively. An example demonstration is presented in Section 5, followed by the conclusions in Section 6.

2 ARCHITECTURE

The design philosophy of this software aims to achieve three main goals: be as simple and easy to pick up as possible; allow as much designing freedom to the experienced user as possible; and be able to support arbitrarily complex circuits. As such, implementation details are hidden from the user completely, instead kept on a server, and are invoked only when necessary behind the scenes.

The client side, the main interface, is written in HTML, CSS and Javascript, and strides to help the user and protect them from logical errors. Redundant, empty qubits and steps -columns- are deleted automatically, in order to minimize the circuit and enable clearer thinking. Contradictions in circuit design, that arise from the nature of Quantum Theory -e.g. Hadamard gates on top of collapsed qubits- are detected and communicated immediately to the user, with helpful messages that explain the error, while critical operations are blocked until such errors are resolved.

The source code of the user interface is split between different, semantically significant modules. This facilitates the easier improvement of the code and the acceleration of development, the recycling of code, and also prepares the project for community contribution, in a Free and Open Source format.

As mentioned, the parsing happens on a server, which allows the product to be a light, always available, web accessed service. While it is a goal of the implementation to free the user's system from exhaustive resource demands, the whole procedure can also be ran locally, with the use of a simple server -the development phase makes heavy use of the python library Flask [3], and its application can be easily seen on this work's GitHub page [10]. Fig. 1 gives an abstract depiction of the method employed to map the user specified circuit into an actual Qiskit object [12]. A detailed view is given on Section 5.

3 USER INTERFACE

The user interface of QuaCiDe draws inspiration from many previous works [1, 4, 7, 9, 13], to expand on already established and expected ideas, and to re-imagine inherently flawed others, while also staying true to the notation used in the literature [11].

Fig. 2 shows the main designing environment. On initial load, a single tab is spawned, containing two empty qubits, ready to host gates. On the left-hand side, a toolbox with many useful operations can be found. On the bottom lies a toolbar, projecting extra information about the loaded circuit and offering even more options.

The toolbox hosts all the standard single qubit gates, those being the three Pauli gates, the Hadamard gate, the second and fourth roots of Z -symbolized as S and T respectively, as it is common [8, 11]-, as well as parameterized power versions of the Pauli gates. Measurements, controls and the SWAP gate also exist as single qubit gates, so they can be freely dragged onto the circuit as all the rest, but all of them have caveats to ensure their usage aligns with theory. These details will be explained shortly. All gates can be hovered upon to reveal educational descriptions about them, their effect on the register and their matrix representation.

These gates can be snapped onto the qubit wires of the circuit by letting go of them while hovering each respecting wire. Dragging a gate around spawns a new qubit at the bottom of the circuit, and bottom qubits left unused get deleted automatically. Attempting to place a gate on top of another placed gate will part the circuit at this position to make room for the new gate. Dropping gates outside of the circuit or right-clicking them deletes them automatically. Consequently, empty steps resulted from moving and deleting gates get removed automatically by shifting the circuit to cover the gap. Copies of placed gates can be created instantly, to the right and bottom of the original, by invoking the appropriate key combination. A full list of the available mouse-keyboard commands can be found on [10].

Placing a control on a step conditions it in its entirety and thwarts its execution later unless the control qubit has a satisfying value. By default, a control satisfies on state $|1\rangle$, symbolized by black color, but it can be changed to satisfy on state $|0\rangle$ instead, on command. This change is symbolized by white color. Multiple controls can be stacked on the same step.

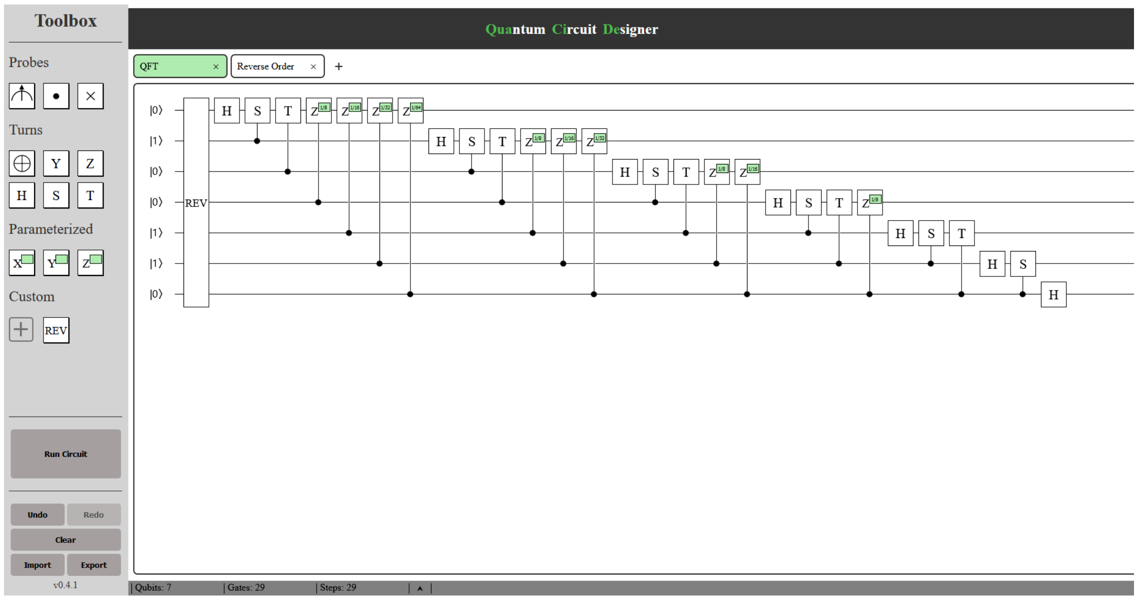


Figure 2: Main interface of QuaCiDe. The implementation of 7-qubit Quantum Fourier Transform is shown.

Measurements permanently collapse the qubit state -symbolized by a double horizontal line- and force the qubit to act as a bit later in the execution phase. Consequently, only controls are allowed to be placed on a collapsed qubit. Such controls become classical. Measurements can be placed in the middle of the circuit to exploit this fact. Measurements can also enforce postselection on the specified qubit on the computational basis, either on $|0\rangle$ or $|1\rangle$.

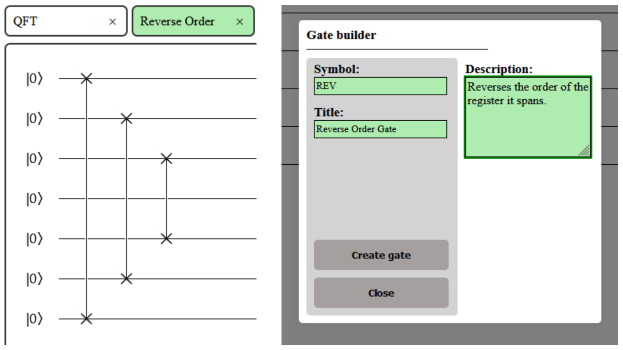
SWAP gates function as single qubit gates, but in reality they are two-qubit gates, and as such, having a single one placed on a step will result in an error. The freedom of gate placement presented in this designer can lead to many contradictions of the established theory. All such cases are assumed to be logical errors, and include non-control gats on collapsed qubits -as previously mentioned-, non-arithmetic inputs on parameterized gates, and one or more than two SWAP gates on the same step. These errors are detected immediately and are shown to the user in red outlines and guiding messages, to help them understand and fix them.

Qubits can be initialized beforehand -as shown in Fig. 2-, and the starting state can be seen in the ket displays on their left side. The state can be shuffled between the six significant points of the Bloch sphere

$$|0\rangle, |1\rangle, |+\rangle, |-\rangle, |+i\rangle, |-i\rangle$$

and qubits can be grouped under colored borders. These borders have no inherent semantic or functional meaning and can be used by the experimenter to sign specific qubits based on significance, or bind them together into sub-registers, as neighboring qubits sporting same colored borders have their borders unified. Every qubit also hosts a self-delete button, that deletes the qubit and all its pinned gates, and an addition button, that spawns a fresh empty qubit above it.

The user can expand a single circuit to their heart's desire, but they can also split work among multiple tabs. Fig. 3 shows that, for the definition of the Reverse Qubit Order gate, a different tab was



Symbol: REV

Title: Reverse Order Gate

Description: Reverses the order of the register it spans.

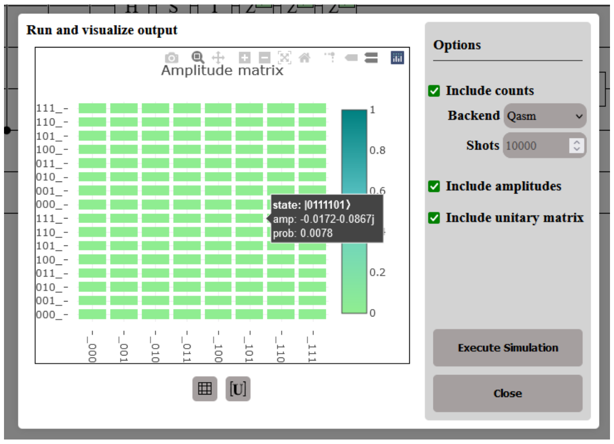
Figure 3: Custom gate creation. Left: the new gate definition on a new tab. Right: the gate creation window.

used. Each tab has its own circuit, statistic and endianness. Tabs can be accessed by their respective link underneath the main title. The bottom toolbar updates to always project the information of the currently active tab. This information consists of the number of used qubits, gates, steps and the declared endianness, which starts at Big Endian by default but can be toggled to Little Endian.

Non-empty, error-free circuits that have no measurements can be turned into custom gates. These gates behave exactly as any other gate and are independent of the circuit that defined them, meaning they can be safely placed anywhere without feat of recursive definitions. Creation happens through the gate builder window, shown in Fig. 3, and uses the currently active tab to define the new gate. This action is blocked if the circuit is error-prone or empty. The experimenter has the option to customize the new gate, by adding a symbol, title and description to it, as notes to its effect. This information is visible on hover. Declared gates can be removed from the toolbox on command at any time.

Apart from gates, the toolbox also supports buttons for many quality-of-life actions. All such actions concern the currently active tab and all of them have keyboard shortcuts for the user's convenience. *Undo* and *Redo* concern gate placement and deletion, state and border change, and qubit creation and deletion. Undo revokes the last taken action, while redo re-applies the last undone action. *Clear* deletes the entire circuit and reverts it back to its default state, with two empty qubits. *Export* saves the loaded circuit, in JSONLines format, on a file that is then downloaded on the user's system. This file can be later used to restore the same circuit on a new tab, through *Import*.

The loaded circuit can be executed on many simulator backends through the *Run* button. The preparation modal of Fig. 4 appears, where the user can finalize the details about the experiment, such as specifying the desired backend to run on, the number of repetitions -or *shots*-, and the type of results to receive. Three kinds of outputs are supported; measured register state counts, theoretical state amplitudes and the total unitary matrix of the circuit. The counts are calculated by the specified backend and are visualized as a histogram of the measured states and their occurences. Note that if no measurements are specified on the circuit, no counts are returned. The amplitudes of each possible state are calculated by Qiskit-Aer's StatevectorSimulator [12], and are returned as a probabilistic heat map. Finally, the unitary matrix of the circuit is calculated by UnitarySimulator [12], and is again returned as a heat map of the square magnitudes for easier visualization. All of these graphs are completely interactable and can be downloaded as images and, in the case of the unitary matrix, as parsable text.



state: |0111101⟩
amp: -0.0172-0.0867j
prob: 0.0078

✓ Include counts
Backend Qasm
Shots 10000
✓ Include amplitudes
✓ Include unitary matrix

Figure 4: Results of running the circuit of Fig. 2. The heat map is based on the theoretical probability of each state. All possible states have the same amplitude. The phase of state $|0111101\rangle$ is shown as an example.

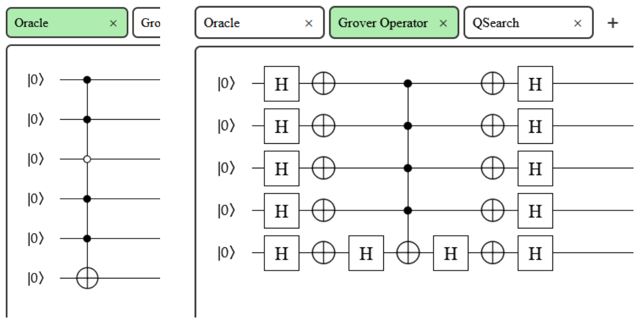


Figure 5: Sub circuits of the Grover search algorithm. Left: the oracle that marks the specified state, here $|11011\rangle$. Right: the Grover diffusion operator.

4 PARSER

Simulating quantum computation classicaly is a tremendously resource-demanding task when multiple qubits are required [8, 11]. Simulation services must isolate the bulk of the actual computation away from the user's system. As previously mentioned, this software employs a server-client schema to translate and run the specified circuits on a remote and readily scalable environment.

As soon as the user sends the request for executing their experiment simulation, their client interface compiles the relevant circuit data into JSONLines format. This payload is then sent to the server and read line-by-line, to reconstruct the described circuit in a way Python and Qiskit understand. The first lines of this payload always describe custom gates found in the final circuit; these gates are included in order of earliest defined to latest, such that no undefined gate can be met inside a circuit. The last line is guaranteed to be the final circuit definition.

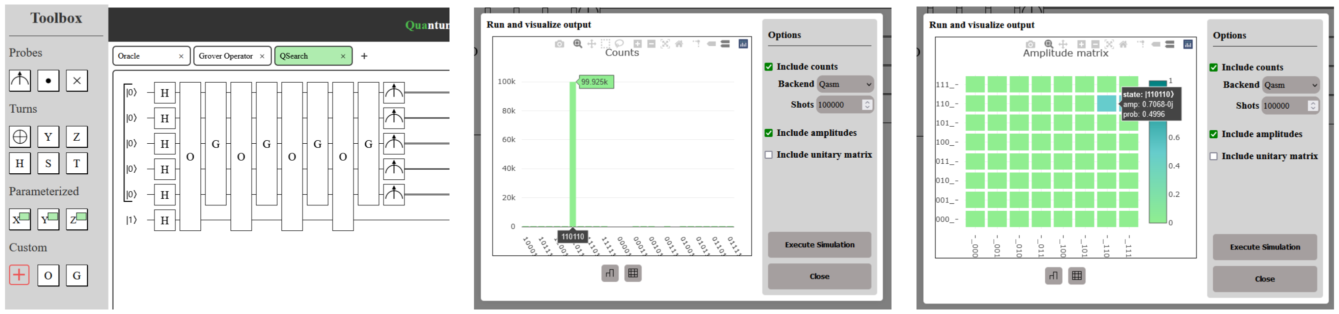
Each JSON line is transmuted into a gate matrix, where the rows represent the gate train of the corresponding qubit. This matrix is then read column by column, and the contents of each column get unified into a single custom Qiskit gate; this is done so any controls existing in the column can affect it in its entirety, as expected. Then, measurements are applied and any specified post selections are noted. The resulting circuit is turned into a gate, if it was an initial line inside the JSON payload, otherwise it is immediately executed.

Execution is split between three sub-experiments, one for each of the three possible result types. They happen in bottom-up order, as shown in Fig. 4, to preserve and recycle resources. Their outputs are filtered according to the specified post selections, and are then returned to the client interface to be visualized.

This entire process has been designed to be wholly independent of the user interface. They are only intertwined in how they send and receive data. Notably, any program that can mimic this data format is a viable parser. This was done for development purposes, for the ease of the parser's improvement, and also for the convenience of the experienced user, who may wish to modify or replace the existing parser to achieve something different.

5 EXAMPLE DEMONSTRATION

As proof of concept, a simulation of Grover's algorithm [5] is presented. Grover's fast searching algorithm is of pivotal importance to quantum computation and signifies the principle of Quantum Supremacy. This example here follows the architecture of [11], scaled to five qubits.



Include counts ✓
Backend Qasm
Shots 100000
Include amplitudes ✓
Include unitary matrix ☐

state: |110110⟩
amp: 0.7068-0j
prob: 0.4996

Figure 6: Left to right: the final structure of Grover's search algorithm for state $|11011\rangle$; Counts results of simulation; Amplitude results of simulation.

From a database of 32 elements, the element at position 11011 needs to be located. The oracle that marks this state is described by

$$f(x) = \begin{cases} 0, & x \neq 11011 \\ 1, & x = 11011 \end{cases}$$

and can be implemented as a circuit using a single multi-controlled X gate, as shown in Fig. 5.

Grover's searching algorithm amplifies the desired state's amplitude at every iteration, until measurement is almost certain to show that state [5]. Fig. 5 shows the implementation of the Grover diffusion operator, which follows the application of the Oracle at every step to amplify the marked state. For a searching problem of five qubits, we need to apply this combination a total of 4 consecutive times, as shown by

$$\frac{\pi}{4}\sqrt{N} - \frac{1}{2} = \frac{\pi}{4}\sqrt{2^5} - \frac{1}{2} \approx 4$$

which describes the amount of iterations needed [8]. The final algorithm structure can be seen on Fig. 6.

This particular experiment was ran on QASM simulator [12] 100000 times. Fig. 6 shows how measurement forms a peak around state $|110110\rangle$[1] while the rest of the possible states occur an insignificant amount of times. From the generated amplitude matrix, one can see that the states $|110110\rangle$ and $|110111\rangle$ have an equal and practically 50-50 chance of appearing, which shows that the experiment worked and the desired state is received[1].

6 CONCLUSIONS AND FUTURE WORK

QuaCiDe covers all the necessary requirements to render itself an adequate tool for quantum circuit design and simulation, and employs many new techniques to send itself even further beyond. Its simplistic and informative nature makes it a great tool for educational purposes, while its powerful foundation enables professional research.

Future work involves overhauling the parser, in order to use decision diagrams for the creation and execution of the circuits, as per the work of [2]. The entire implementation is being tested and fine tuned daily, in order to get the software running on a publicly available web server as soon as possible. The source code is being updated to use TypeScript and jQuery, in order to make the project ready for Open Source publication. A variety of new features are also in development, such as loading circuits written in QASM format and allowing for the execution of variational algorithms.

QuaCiDe is still under development, but it is free to use for everyone and can be obtained by forking the relevant repository [10] on GitHub.

ACKNOWLEDGMENTS

Asimakis Kydros thanks the Aristotle University of Thessaloniki, and especially the Department of Informatics and its professors, for their invaluable support, guidance, resources and teachings throughout his studies and the fruition of this work.

REFERENCES

[1] Andreas de Vries. 2015. JQuantum: A Quantum Computer Simulator. (2015). https://jquantum.sourceforge.net/.

[2] M.A. Thornton D.M. Miller. 2006. QMDD: A Decision Diagram Structure for Reversible and Quantum Circuits. In *36th International Symposium on Multiple-Valued Logic (ISMVL'06)*. IEEE, Singapore, 30–30. https://doi.org/10.1109/ISMVL.2006.35

[3] Flask contributors. 2024. Flask: The Python micro framework for building web applications. (2024). https://flask.palletsprojects.com/en/3.0.x/.

[4] C. Gidney and contributors. 2019. Quirk: a toy for exploring and understanding quantum circuits. (2019). https://github.com/Strilanc/Quirk.

[5] Lov K. Grover. 1996. A fast quantum mechanical algorithm for database search. arXiv:quant-ph/9605043 [quant-ph]

[6] IBM. 2016. IBM Quantum Composer. https://quantum.ibm.com/composer/files/new.

[7] I. G. Karafyllidis. 2005. Quantum Computer Simulator based on the Circuit Model of Quantum Computation. *IEEE Transactions on Circuits and Systems I* 52 (2005), 1590–1596.

[8] I. G. Karafyllidis. 2015. *Quantum Computation*. Kallipos, Open Academic Editions. https://doi.org/10.57713/kallipos-892

[9] Nikos Konofaos Konstantinos Prousalis. 2016. QuCirDET: A design and simulation tool for quantum circuits. In *2016 5th International Conference on Modern Circuits and Systems Technologies (MOCAST)*. IEEE, Thessaloniki, Greece, 1–4. https://doi.org/10.1109/MOCAST.2016.7495162

[10] Asimakis Kydros. 2024. QuaCiDe: An intuitive, user-friendly Quantum Circuit designer. https://github.com/asimakiskydros/QuaCiDe.

[11] I. L. Chuang M. A. Nielsen. 2010. *Quantum Computation and Quantum Information* (10 ed.). Cambridge University Press.

[12] Qiskit contributors. 2023. Qiskit: An Open-source Framework for Quantum Computing. https://doi.org/10.5281/zenodo.2573505

[13] I. Bush et al T. Jones, A. Brown. 2019. QuEST and High Performance Simulation of Quantum Computers. *Scientific Reports* 9 (2019), 10736–. https://doi.org/10.1038/s41598-019-47174-9

[1]Note that the qubit at position 0 is auxillary and not part of the original database. Thus, its value is not taken into consideration and it is never measured. In the counts histogram of Fig. 6, it happens that it defaults to $|0\rangle$. However, the heat map shows both theoretical possibilites. The same information is extracted in both cases.