

A System and Method for Quantum Error Correction Circuit Verification

Priority Claim

This application claims priority from Singapore Patent Application No. 10202400302R
5 filed on 1 February 2024.

Technical Field

The present invention relates broadly, but not exclusively, to a system and a method for verifying quantum error correction circuit(s).

Background of the Invention

10 Quantum computing has established itself as one of the leading contemporary technological trends. Quantum computers with over 100 qubits are now freely available and readily accessible, based on a variety of different physical platforms. At the same time, software tools for programming the devices, including applications libraries and developer tools, have lowered the barrier to entry for many potential users of the technology.

15 Useful large-scale quantum computing requires quantum error correction (QEC) and fault tolerance (FT) to combat the proliferation of noise that corrupts the quantum information being processed. In particular, on the software side, there is a need for tools suitable for the design, simulation, analysis, and construction of fault-tolerant quantum circuits.

It is an object of the present invention to provide a system and/or method for addressing the
20 issue of noise that corrupts the quantum information being processed during the design and construction of quantum error correction circuits or which will at least provide the industry with a useful choice. This problem will be referred to as the 'Error Correction Circuit Verification' (ECCV) problem.

Summary of Invention

25 According to a first aspect of the invention, there is a system for verification of a quantum error correction circuit, the system comprising:
a data structure representation configured to receive and/or store a quantum error correction code, an operation performed within the quantum error correction code, and an input circuit,

an output module configured to prepare simulations of the input circuit,
a simulator configured to simulate the input circuit, and
wherein the output module is configured to output at least one Boolean variable indicating
whether the input circuit passes error correction circuit verification (ECCV) conditions,
5 wherein the output module is further configured to provide debugging information when at
least one Boolean variable indicates the input circuit fails at least one of the ECCV
conditions.

10 In an embodiment, the data structure representation comprises validation checks for
verifying the data correctness and/or consistency of at least one of the following: the input
circuit, the quantum error correction code, and/or the operation.

In an embodiment, when the validation checks indicate incorrect data, an error message is
returned through the output module.

15 In an embodiment, the data structure representation comprises a cross-referencing system
for checking at least one of the following: the input circuit, the quantum error, and/or the
operation.

In an embodiment, the data structure representation comprises information to serve as
inputs in the output module, and/or the simulator.

In an embodiment, the data structure representation is further configured to communicate
with the output module, and/or the simulator.

20 In an embodiment, the data structure representation comprises a compressed and/or
modular representation of instructions to execute operations of the quantum error correction
code.

In an embodiment, the output module is configured to interact with the simulator through
the data structure representation.

25 In an embodiment, the system further comprises a code builder module configured to define
the quantum error correction code, and output the quantum error correction code to the data
structure representation.

In an embodiment, the code builder module is configured to allow a user-defined error
correction code as the quantum error correction code.

In an embodiment, the code builder module is configured to verify the quantum error correction code, and/or a set of logical Pauli operators of the quantum error correction code for consistency.

5 In an embodiment, the code builder module is configured to define the quantum error correction code as a stabiliser set comprising elements of N-qubit stabiliser state.

In an embodiment, the code builder module is configured to create a destabiliser set corresponding to the stabiliser set, wherein each element of the destabiliser set forms a pair with a corresponding element of the stabiliser set, each element of the stabiliser anti-commutes with a corresponding paired element of the destabiliser set, and commutes with
10 other elements of the destabiliser set.

In an embodiment, the code builder module is configured to find logical operator sets for the stabiliser set, wherein all elements of the logical operator sets commute with the elements of the stabiliser set, and wherein the logical operator sets comprises a first logical operator set and a second logical operator set, each element of the first logical operator set
15 forms a pair with a corresponding element of the second logical operator set, each element of the first logical operator set anti-commutes with a corresponding paired element of the second logical operator set, and commutes with other elements of the logical operator set.

In an embodiment, the system further comprises a circuit builder module configured to build and/or receive an input circuit, and output the input circuit to the data structure
20 representation.

In an embodiment, the circuit builder module comprises abstractions and convenience functions.

In an embodiment, the circuit builder module is configured to support construction of the input circuit from pre-defined quantum gates, and/or user-defined groupings of quantum.

25 In an embodiment, the system further comprises an operation builder module configured to define the operation, and output the operation to the data structure representation.

In an embodiment, the ECCV conditions comprise: a first condition, wherein the input circuit passes the first condition when an output state of the input circuit is encoded within the quantum error correction code, a second condition, wherein the input circuit passes the
30 second condition when the output state of the input circuit has the same logical operators as the quantum error correction code, and a third condition, wherein the input circuit passes the third condition when the input circuit is extracting syndrome information.

In an embodiment, the input circuit comprises a plurality of sub-circuits, and the output module is configured to output Boolean variables indicating whether at least one sub-circuit of a plurality of sub-circuits passes the first condition, the second condition, and the third condition.

- 5 In an embodiment, the ECCV conditions comprises: a first condition, wherein the input circuit passes the first condition when an output state of the input circuit is encoded within the quantum error correction code, a second condition, wherein when the input circuit contains syndrome measurements, the input circuit passes the second condition when the input circuit is extracting syndrome information, and/or a third condition, wherein the input
10 circuit passes the third condition when its action on each member of a set of input states results in an output state that is consistent with the expected transformation rules.

In an embodiment, the output module is configured to check the ECCV conditions for a logical Clifford operation, wherein the logical Clifford operation comprises a user-supplied set of expected transformation rules, the expected transformation rules describing
15 theoretical transformation of logical Pauli operators by the Clifford operation.

In an embodiment, the data structure representation comprises information to serve as inputs in at least one of the following: a circuit builder module, a code builder module, and/or an operation builder module.

- In an embodiment, the data structure representation is further configured to communicate
20 with at least one of the following: a circuit builder module, a code builder module, and/or an operation builder module.

- According to a second aspect of the invention, there is a method for verifying a quantum error correction circuit, the method comprising:
receiving and/or storing a quantum error correction code, an operation performed within the
25 quantum error correction code, and an input circuit,
simulating the input circuit,
outputting at least one Boolean variable indicating whether the input circuit passes ECCV conditions, and
providing debugging information when at least one Boolean variable indicates the input
30 circuit fails at least one of the ECCV conditions.

Brief Description of the Drawings

Embodiments of the invention will be better understood and readily apparent to one of ordinary skill in the art from the following written description, by way of example only, and in conjunction with the drawings, in which:

Figure 1A shows a system for verification of a quantum error correction circuit according to a first embodiment.

Figure 1B shows a flow chart of the information flow through the modules of the system.

Figure 2 shows an example implementation of the first condition of the ECCV conditions.

Figure 3 shows an example implementation of the second condition of the ECCV conditions.

Figure 4A to 4D shows an example implementation of the third condition of the ECCV conditions.

Figure 5 shows a flow chart of the information between the output module and the simulator.

Figure 6A shows a system for verification of a quantum error correction circuit according to a second embodiment.

Figure 6B shows an example flow chart of the information flow between the modules in the second embodiment.

Figure 7 shows another example flow chart of the information flow between the modules in the second embodiment.

Detailed Description of Embodiments

The ECCV problem may be stated compactly as follows, given:

1. A quantum error correction code Q
2. An operation O that is to be performed within the code Q
3. A quantum circuit or an input circuit C

The following question must be answered: Is the quantum circuit or input circuit C a valid circuit implementation of the operation O for the quantum error correction code Q ? In an embodiment, the quantum circuit is 'correct' or 'valid' if the preceding question is answered in the affirmative. Otherwise, the quantum circuit is 'incorrect' or 'invalid'. The quantum circuit is

sometimes referred to as the input circuit, and for the purposes of this disclosure the two terms should be considered interchangeable.

The formal problem statement above can be phrased more intuitively as follows. The error correction code Q defines the way that logical information is encoded in the quantum system (i.e. in the qubits being used). The operation O is some transformation or operation we wish to perform on the encoded information, for example, a qubit rotation to transform the logical state from zero to one (or, more generally, from some quantum superposition to another), a two-qubit logical CNOT gate, or a measurement. The quantum circuit C is a candidate circuit implementation of the operation O . The goal is to check whether C actually performs O in such a way that is consistent with the error correction code Q .

The error correction code Q and the input circuit C are taken as fixed inputs when solving the ECCV problem. In quantum error correction codes, information is typically encoded in a non-trivial way using many physical qubits to represent a logical qubit. In some QEC codes, the same set of physical qubits may even encode more than one logical qubit. Accessing and manipulating logical information in a way that is consistent with the encoding therefore requires the careful design of quantum circuits to perform the desired operations. Once the practitioner has designed an input circuit C , the input circuit may be verified as correct, or rejected as incorrect: this is the goal of the ECCV problem.

To solve the ECCV problem is to set definitions on the set of input circuits that can fulfil certain computational functions. Because larger circuits can be built upon these components as sub-circuits, it is important to know that these input circuits behave as intended.

There are several reasons why solving the ECCV problem itself is a non-trivial task. In addition to showing that the input circuit performs the desired operation on the logical qubit(s), there is also the added complexity that comes from the fact that the logical qubits are encoded with some QEC code, which means that the input circuit needs to respect the structure of the QEC code, as well as provide functions associated with maintaining the encoded information in the code. Even for relatively small circuits, the verification task is error-prone if performed manually. Computations performed by hand can span multiple pages of working, requiring careful tracking of relevant variables. Manual solutions become impractical for large quantum circuits. The growing number of conditions to be checked means that the time required to find a solution is infeasible. If the input circuit is allowed to take an arbitrary form, the set of rules to be checked to ascertain correctness can become highly complicated, and even difficult to define.

There are two examples where a practitioner may be interested in solving the ECCV problem. Firstly, when a new QEC code Q has been designed, one must design candidate circuits for all

basic logical operations, and subsequently check that they are correct. Secondly, when more resource-efficient circuit implementations of the operation O are sought, for example, if we wish to find a syndrome measurement circuit that uses fewer ancilla qubits than some previously known version, the correctness of the modified candidate circuits consider will need to be verified.

There are practical reasons why the ECCV problem needs to be solvable easily. Firstly, the design, implementation and execution of (large) quantum circuits may be a collaborative effort in the future. Under such circumstances, human errors will inevitably occur somewhere in this process. Solving the ECCV problem on individual circuit components or blocks ensures that these errors can be easily found and corrected. Secondly, other analysis about circuits may rely on the assumption that the circuits are truly valid for what they assert to do.

Currently there is no end-to-end, self-contained software solution for solving the ECCV problem. A user would have to build a system from scratch (with the need to define suitable ECCV conditions), possibly calling upon open-source or custom software for supporting tasks such as stabiliser circuit simulation.

There are a number of academic publications and documents whose titles are suggestive that they may be related to the ECCV problem. However, closer inspection reveals that the problem being addressed in each of them is not in fact the ECCV problem. In the domain of fault-tolerant QEC, similar language is often used to discuss different concepts, generally relating to problems at different levels of abstraction. In other words, the verification that a quantum circuit does what it is intended to do is a question that can be posed on many levels. None of the publications address the ECCV problem as defined in the present disclosure.

System 100

The invention relates primarily to the system and method for verification of a QEC circuit. Figure 1A shows a system 100 for verification of a QEC circuit. The system comprises a data structure representation 101 configured to receive and/or store a QEC code Q , an operation O performed within the QEC code, and an input circuit C . The system 100 further comprises an output module 102 configured to prepare simulations of the input circuit C , and a simulator 103 configured to simulate the input circuit C . The output module 102 is further configured to output at least one Boolean variable indicating whether the input circuit C passes ECCV conditions, wherein the output module 102 is further configured to provide debugging information when at least one Boolean variable indicates the input circuit C fails at least one of the ECCV conditions. The data structure representation 101, output module 102 and the simulator 103 may be referred to as modules in the system. For

the purposes of this disclosure the term module and/or modules may be used to refer to any one or more of the data structure representation 101, output module 102 and/or the simulator 103. The modular design allows custom criteria to be designed and verified for an input circuit C.

- 5 Figure 1B shows an example flow chart of the information passing through the modules of the system. In an embodiment, a method for verifying a QEC circuit is disclosed. The method comprising: receiving and/or storing a QEC code, an operation performed within the QEC code, and an input circuit, simulating the input circuit, outputting at least one Boolean variable indicating whether the input circuit passes ECCV conditions, and
- 10 providing debugging information when at least one Boolean variable indicates the input circuit fails at least one of the ECCV conditions. By way of example, the embodiments described in the disclosure relates primarily to the system. However, it will be appreciated that the disclosure may also be used in a method and/or algorithm.

Data structure representation 101

- 15 The data structure representation 101 comprises information to serve as inputs in at least one of the modules. The information may include the QEC code, the operation, or input circuit. The data structure representation 101 may be further configured to communicate with at least one of the modules.

- The data structure representation 101 comprises validation checks for verifying the data correctness of at least one of the modules. The data structure representation 101 may check that the data is complete, in the sense that the information it contains is always enough to serve as input to other modules. The data structure representation 101 comprises a cross-referencing system for checking at least one of the following: the input circuit, the quantum error, and the operation. This ensures the data within the data structure is safely accessed,
- 20 inserted, and modified by different modules. This allows a modular solution to the system 100. When the validation checks indicate incorrect data, or the data is incomplete an error message may be returned through the output module 102.

- In an embodiment, the data structure representation 101 may be configured to verify commutation of code stabilisers, and/or consistency of the number of qubits, for example,
- 30 data and/or ancilla, in the QEC code Q and/or the input circuit C. When at least one check fails, an error message may output through the output module. The error message may include debugging information.

In an embodiment shown in Figure 7, the data structure representation 101 may be configured to verify at least one of the following: anticommutation of logical operators, anticommutation of stabilisers and/or destabilisers, consistency of expected transformation rules, and/or automated computation of logical and/or destabilisers. When at least one
5 check fails, an error message may output through the output module 102. The error message may include debugging information. When all the checks pass, the information is passed to the output module for verification of ECCV conditions.

The data structure representation 101 may have its own standard application programming interface (API). The communication with the modules may be through the four basic
10 database CRUD (Create, Read, Update, and Delete) operations. The data structure representation 101 may communicate with the modules through CRUD-like operations. These methods are general database-like functionalities used to generate data structures. They may be employed by a user to create complex abstractions to define QEC codes, for example, in the case of the code builder. Another example is to define input circuits in the case of the circuit
15 builder.

The data structure representation 101 may be configured to generate a valid data structure object. In an embodiment, the valid data structure objection is generated using an archetype, an embedding, an operation, and an implementation. These may be specified by the user. The archetype may be templates and/or prototypes of the stabiliser classes used in
20 a QEC code. For example, in a QEC code it is common to see types of stabilisers being repeated across some unit of space. The archetype may define the set of unique stabiliser shapes and types, where the type represents the Pauli operators (e.g. Pauli-X, Pauli-Z) defining the stabiliser. The embedding is a process used to define relationships between archetypes (such as how they overlap) and embed them into some space. This is necessary
25 when constructing a QEC code. Even without having to strictly define a geometry and a topology (e.g. a hexagonal grid), the embedding contains all the information about the relationship between archetypes. In its simplest version, the embedding is a list of stabilisers where each stabiliser is of a given archetype kind. The operation is the data structure containing all necessary information on the logical operation to be performed. The
30 data structure representation 101 needs to embed the explicit set of instructions that should be performed to implement the desired QEC operation O in practice. The operation and/or the implementation may be supplied by the user in any standard quantum programming language, such as QASM or QIR for example.

The data structure representation 101 comprises a compressed and/or modular
35 representation of instructions to execute operations of the QEC code. The data structure

representation 101 may feature a specific data type, herein referred to as the “Q-Loom” in the disclosure. Q-Loom is a compressed and modular representation of the instructions needed to execute operations of the QEC code Q (stabiliser measurements, logical gates) on a device. The device may be a simulated device. The Q-Loom may be designed to be able
5 to write nested operations in a compressed format.

A feature of the Q-Loom is that it may natively support controlled operations, independently of whether the control operation is classical or quantum. This means that loops such as “repeat until success” may be written in an extremely compact way, thus saving memory and computing resources.

10 The Q-Loom, being a data type of the data structure representation 101, may be programmed through the data structure representation 101 internal API. However, abstraction and convenience functions are extremely useful when creating input circuit C at scale.

In an embodiment, the data structure representation 101 may be a circuit root data (CRD) or
15 a circuit root data representation and for the purposes of this disclosure, the terms may be used interchangeably.

Output module 102

In an embodiment, the output module 102 is configured to check the ECCV conditions. In an example embodiment, the operation O is syndrome measurement. The syndrome
20 measurement may be applicable to syndrome measurement circuits. Similar conditions can be defined for other intended operations O, such as logical gates or state preparation for example.

In an embodiment, the output module 102 may be configured to check the ECCV conditions. In another embodiment, the output module 102 may output a Boolean variable
25 indicating whether the input circuit C has passed the ECCV conditions. The output module 102 may give more than one Boolean variable, indicating whether the input circuit passes or fails each of the conditions. The output module 102 may provide debugging information to help the user identify potential issues when the input circuit fails at least one condition of the ECCV conditions.

30 In another embodiment, the output module 102 may output the data structure objects corresponding to at least one of the following: the specified error correction code Q, the input circuit C, and/or the operation O.

In another embodiment, the output module 102 may output the data computed by means of the algorithms disclosed herein, for example, the logical operators of the error correction code Q.

The ECCV problem, as it relates to syndrome measurement circuits, may be stated as follows: for a given QEC code Q, and an input circuit C intended to implement a full set of syndrome measurements for Q, does the input circuit C indeed achieve this correctly?

The following general assumptions may apply to the definition of the ECCV conditions. Firstly, the input circuit C is composed entirely of Clifford operations. This should always be the case for a syndrome measurement circuit for stabiliser codes. Moreover, this condition enables the simulation of large circuits, consisting of many qubits and gates, using the stabiliser formalism. This may allow for checking the ECCV conditions.

Another assumption that may apply to the definition of the ECCV condition is that the input circuit is defined on three registers: a first register, a second register, and a third register. A register refers to a group of bits or qubits being used for a certain purpose. The first register is a set of qubits referred to as the 'data qubits', which are those carrying the quantum information of the desired computation. The second register is a set of qubits called the 'ancilla qubits', which are used to facilitate the extraction of information from the data qubits, and/or determining if an error has occurred in the computation. The ancilla qubits are always initialised in the state $|0\rangle$ prior to application of the syndrome measurement circuit. The third register is a set of classical bits, denoted R. After the application of the input circuit C, the ancilla register is measured, and all measurement results are written to R. Additionally, classical computation can be performed on R, and the results can be written to R.

The output module 102 may be configured to check the ECCV conditions for a logical Clifford operation, wherein the logical Clifford operation comprises a user-supplied set of expected transformation rules, describing how logical Pauli operators should theoretically be transformed by the Clifford operation. For example, in the case of a CNOT operation acting on two qubits labelled 0 and 1, where qubit 0 is the control qubit and qubit 1 is the target qubit, the expected transformation rules would be:

$$X_0 \rightarrow X_0 X_1$$

$$X_1 \rightarrow X_1$$

$$Z_0 \rightarrow Z_0$$

$$Z_1 \rightarrow Z_0 Z_1$$

Moreover, one way to check that the expected transformation rules are in fact correctly implemented by the input circuit C is to verify that they hold true for the set of states

- 5 $|00\dots 0\rangle$
 $|++\dots+\rangle$
 $|+0\dots 0\rangle, |0+0\dots 0\rangle, \dots, |00\dots+\rangle$
 $|0+\dots+\rangle, |+0+\dots+\rangle, \dots, |++\dots 0\rangle$

- 10 Namely, the state where all logical qubits are in the state $|0\rangle$, the state where all logical qubits are in the state $|+\rangle$, the set of all possible states where N-1 logical qubits are in state $|0\rangle$ and one is in state $|+\rangle$, and the set of all possible states where N-1 logical qubits are in state $|+\rangle$ and one qubit is in state $|0\rangle$.

- 15 It is important to emphasise that here we are referring to logical qubit states, so for example $|00\dots 0\rangle$ means that all logical qubits are in the logical state $|0\rangle$.

The set of stabilisers that define the error correction code Q is denoted as $S = \{s_1, s_2, s_3, \dots\}$. Similarly, the set of logical Pauli operators of Q is denoted as $L = \{LX_1, LZ_1, LX_2, LZ_2, \dots\}$, where in general LA_j is a multi-qubit Pauli operator. For example, for the distance-3 repetition code, we have the two logical Pauli operators $LX = XXX$, and $LZ = ZZZ$.

- 20 It is assumed that both S and L have been verified for consistency prior to checking the ECCV conditions.

- 25 The logical qubits are specified to be used in the computation. Some QEC schemes, known as subsystem codes, do not make use of all the possible logical qubits that can be defined by the set L. For example, in principle the 4-qubit code can encode 2 logical qubits. Its stabilisers are $S = \{XXXX, ZZZZ\}$, and the logical operators are $L = \{LX_1 = XXII, LZ_1 = ZIZI, LX_2 = IXIX, LZ_2 = IIZZ\}$. In an alternative embodiment, the information may be encoded only in a logical qubit defined by the operators LX_1 and LZ_1 . The other logical qubit, defined by LX_2 and LZ_2 , is then referred to as a 'gauge qubit'.

- 30 Let M be a subset of L (this may be chosen by the user). Let the number of logical qubits thus chosen be n.

Let the output of C that carries the encoded state for computation be represented as a stabiliser array, with the array denoted by set $G = \{g_1, g_2, g_3, \dots\}$.

In an embodiment, the ECCV conditions comprises a first condition, a second condition, and a third condition. The input circuit C is a valid syndrome measurement circuit for the error correction code Q if the three conditions are satisfied.

5 The input circuit passes the first condition when an output state of the input circuit C is encoded within the QEC code Q. In particular, the first condition is passed if the output state has the same set of stabilisers S as the QEC code Q. Figure 2 shows an example implementation of the first condition.

10 The first condition may be expressed as follows. For each element s in S , there exists a non-empty subset A from G such that the product of all the elements in A is equal to s . The first condition may be checked in practice by taking each s in S , propagating it through the input circuit C, and ensuring that it remains unchanged.

15 This condition may not be specified with respect to a particular input state, because the input state may be encoded in a different error correction code Q' . Q' may even be the trivial code, i.e. no code whatsoever. Because the first condition is general, if C is valid, then the first condition must also be applicable when the input state is already encoded in code S. In that case, by taking an input state stabilised by S, and propagating it through C, the output state will still be stabilised by S. If the output state is not stabilised the circuit is not a valid syndrome measurement for S. The output state may be much larger, of which S is only a sub-system.

20 The input circuit passes the second condition when the output state of the input circuit has the same logical operators as the QEC code. Figure 3 shows an example implementation of the second condition.

25 For each logical qubit q_i , where $1 \leq i \leq n$, if the input state to C for q_i is stabilized by the logical X operator, i.e. by LX_i , then there exists some non-empty subset A from G such that the product of all the elements in A is equal to LX_i . A similar statement holds for the logical operator Z acting on q_i .

The second condition may be checked in practice by taking each logical operator m in M, propagating it through the circuit C, and ensuring that it remains unchanged.

30 In an embodiment, the first condition and the second condition may be combined as follows: for each logical operator m in M, take as input to C the state $S+m$, i.e. the state jointly stabilised by S and m; propagate this state through C to obtain the output state, and then perform a measurement of each s in S, as well as m. All measurement outcomes should be deterministic.

In general, information on a given logical operator may not be contained in only one row of the output stabiliser array. In some instances, it is likely that the logical state must be found as the product (row-sum) of certain rows of the output array. The output code Q need not correspond to the input code Q' . The number of logical qubits need not be the same.

- 5 The input circuit passes the third condition when the input circuit is extracting syndrome information. Syndrome information results from a set of measurements made on a certain set of qubits ('ancillary qubits') in the circuit. The syndrome information may be used to determine if an error has occurred in the circuit. However, it is a challenge to construct circuits which correctly extract syndrome information. A syndrome extraction circuit must
- 10 not destroy the encoding scheme (QEC code) being used, which is what the first and second conditions check for. The first and second conditions are not sufficient to rule out the possibility that C is a trivial circuit, such as the identity operation. Since C is intended to be a syndrome measurement circuit, it should be extracting syndrome information. The third condition is checking that a proposed syndrome extraction circuit does indeed yield
- 15 the necessary information. Some classical function must be able to take the measurement outputs and return the syndrome for the stabilisers of S . Figures 4A to 4D shows an example of the implementation of the third condition.

- For each element s_j in S , there exists some function f_j on R , denoted $f_j(R)$, such that $f_j(R) = \text{sign}(s_j)$. The 'sign' operation here refers to whether the state is a +1 or -1 eigenstate of the
- 20 stabiliser s_j . has the meaning that if the stabiliser s_j is, for example, the operator ZZI , then $\text{sign}(s_j)$ should tell us if the state is the +1 or -1 eigenstate of that operator.

The third condition may be checked by the following procedure:

- Define a new circuit C' , which is derived from the original circuit C , with the removal of the final measurements on qubits of the ancilla register of C . That is,

25 this new circuit C' should contain no measurements.

- Let A be the stabiliser array that represents the measurement operators on each of the qubits in the ancilla register. Array A should be a diagonal array of single-qubit Z operators.
- Propagate the array A backwards through the circuit C' , and call the resulting array

30 A' . Note that A' will now include qubits from both the data and ancilla registers.) For each element s in S , we desire to know $\text{sign}(s)$ with respect to the state A' .

- Augment the matrix A' by adding the arrays S and A as additional rows to A' , calling the result A'_{aug} . By using an appropriate sequence of column swaps, ensure

that the columns that refer to the data qubit register come before the columns that refer to the ancilla qubit register.

- Reduce A'_{aug} into row-echelon form (REF), by means of row-sums. In REF, the number of rows in A'_{aug} that are entirely zeros should be equal to the number of rows of S . If this is not satisfied, then no function on R exists that will give the syndromes.

The procedure above checks whether the syndrome information lies somewhere within the classical register after measurements take place. In an alternative procedure, the algorithm relies on the user promising that the syndrome information will be found at a specific location. Let R_n denote the classical register bit where the user promises that the syndrome bit - $\text{sign}(s_j)$ - will be held after C has been successfully executed. To verify the user's claim, the following steps are carried out:

- Choose any input state that is stabilised by S , i.e. by all s_j in S . After execution of the circuit C , all corresponding syndrome bits in R should carry the value 0.
- For each stabiliser s_j in S , choose any input state that is stabilised by $-s_j$, but stabilised by $+s_k$ for all other stabiliser operators in S . After execution of the circuit C , the syndrome bit in the register R corresponding to s_j should take value 1.
- If the outcome of the two preceding steps are as expected (i.e. as described in the steps), the circuit C satisfies ECCV Condition 3. If not, the circuit C will not satisfy the third condition.

When a state is measured to be the +1 (-1) eigenstate of a given stabiliser, conventionally this is recorded as a 0 (1) in a corresponding bit of the classical register R .

The first condition, the second condition and/or the third condition may be expressible in alternative but equivalent ways, but not limited to any examples disclosed herein. For example, the ECCV conditions may be expressed using stabiliser tableaux equivalence to cover the first condition and the second condition as disclosed herein.

In an embodiment, the input circuit comprises a plurality of sub-circuits. The multiple sections of sub-circuits may be smaller and simpler to describe. These sub-circuits may allow for a 'divide and conquer' approach to checking the ECCV conditions for the input circuit C , by checking each sub-circuit individually. When each such sub-circuit conforms to the structure defined by the general assumptions for the input circuit, and the sub-tasks

that each sub-circuit is supposed to carry out is specified, then ECCV checking can proceed analogously on the individual sub-circuits.

For example, a valid sub-task may be that C can be broken up into the set of subcircuits $\{C_1, C_2, \dots\}$, each of which is intended to measure one or several code stabilisers. Then

5 ECCV Condition 3 can be verified by:

- verifying that each of the subcircuits does extract syndrome information;
- verifying that the union of syndrome information obtained from all subcircuits covers all of the code stabilisers.

10 In another example for a valid sub-task, suppose that C can be broken up into the set of subcircuits $\{C_1, C_2, \dots\}$. Each such subcircuit need not necessarily individually fulfil the full set of Conditions 1, 2, 3. However, on the whole, C needs to satisfy the entire set of Conditions. This leaves greater freedom and flexibility to design potential C . Some simple examples of how C could be designed are described as follows:

- 15 • Let C be constructed from two subcircuits C_1 and C_2 , such that one follows the other. That is, the output of C_1 becomes the input of C_2 . C_1 can violate condition 1, by outputting a state that is stabilised by some code Q_{alt} . However, since C needs to fulfil all Conditions, this implies that C_2 has to be able to take any state stabilized by Q_{alt} , and output a state that is stabilized by Q .
- 20 • Let C be constructed as two subcircuits such that one follows the other. That is, the output of C_1 becomes the input of C_2 . C_1 can violate condition 2, which performs a quantum unitary transformation on the encoded state. However, since C needs to fulfil all Conditions, this implies that C_2 has to be the unitary inverse to C_1 , so that the output state of C_2 becomes the same as the input state of C_1 .

25 The output module 102 is configured to output Boolean variables indicating whether at least one sub-circuit of a plurality of sub-circuits passes the first condition, the second condition, and the third condition. The Boolean variable may indicate whether the sub-circuit passes or fails at least one condition. The output circuit may give more than one Boolean variable, indicating whether each sub-circuit passes or fails each of the conditions. Each sub-circuit may not need to pass all three conditions, as long as all three conditions are
30 passed by the input circuit C overall. At least one sub-circuit may pass more than one condition. The output circuit may provide debugging information to help the user identify potential issues when at least one sub-circuit fails at least one condition of the ECCV conditions.

In another embodiment, the ECCV conditions comprises a first condition, a second condition, and a third condition. The input circuit passes the first condition when an output state of the input circuit is encoded within the QEC code. When the input circuit contains syndrome measurements, the input circuit passes the second condition when the input circuit is extracting syndrome information. When the input circuit does not contain syndrome measurements, the second condition may be skipped or bypassed. The input circuit passes the third condition when its action on each member of a set of input states results in an output state that is consistent with the expected transformation rules.

In an embodiment, the output module 102 may comprise two modules, a first module, and a second module. The first module may be configured to check whether the input circuit passes the ECCV conditions. The second module may be configured to output information to the user, for example the error message when the data is incorrect or the data is incomplete, and/or debugging information.

Simulator 103

The simulator 103 is configured to simulate the input circuit with different inputs. The simulator 103 receives instructions from the output module 102 to prepare simulation initial states. The output module 102 may be configured to interact with the simulator 103 through the data structure representation 101. The simulator 103 then returns the simulation output. The simulation output is sent to the output module 102 to for analysis. Figure 5 shows an example of this procedure when checking the ECCV conditions. Figure 5 may be a standalone procedure or may be a more detailed version of the later steps shown in Figure 1B. The simulator 103 may be a stabiliser simulator for example. The simulator 103 may be a wavefunction simulator, but other quantum circuit simulation methods may be used. An advantage of using the stabiliser formalism is that it can scale to larger circuit sizes, both in terms of the number of qubits and the number of gates. Different methods of stabiliser simulations exist, such as CHP (CNOT-Hadamard-Phase) and Pauli frame. These may be used interchangeably to achieve the same outcome.

In another embodiment, the system 100 further comprises at least one of the following modules: a code builder 104, a circuit builder 105, and/or an operation builder 106. The code builder 104, the circuit builder 105, and the operation builder 106 may be referred to as modules in the system 100. For the purposes of this disclosure the term module and/or modules may be used to refer to any one or more of the code builder 104, the circuit builder 105, and the operation builder 106. These modules will be disclosed below.

Code builder module 104

- Figure 6A shows the system 100 further comprises a code builder module 104. Figure 6B shows the code builder module 104 is configured to define the QEC code, and output the QEC code to the data structure representation 101. The QEC code may be defined in terms of one or more of the following: stabilisers, stabiliser list, textual form, Tanner graph,
5 matrix form, and/or parity check matrix. The code builder module 104 may take at least one of the following as input: logical Pauli operators of the QEC code Q, destabilisers of the QEC code, expected transformation rules for verification of logical Clifford operations. The code builder module 104 is configured to define the QEC code as a stabiliser set comprising elements of N-qubit stabiliser state.
- 10 In an embodiment, the code builder module 104 is configured to allow a user-defined error correction code as a QEC code. For example, the user may make a custom definition of the logical qubit encoding and the proposed code stabilisers. The code builder module 104 may verify that the user defined error correction code is consistent, and represents a valid QEC code. The code builder module 104 is configured to verify the QEC code, and/or a set of
15 logical Pauli operators of the QEC code for consistency.

To prepare a valid input for the system 100, a valid data representation embedding may be specified. The embedding may be specified by the user. The embedding may be specified together with the underlying archetypes.

- 20 The error correction code may be a known a standard code. For example, from scientific literature or from previous research, in which the definition of the archetypes, embedding, and operation is guaranteed to be correct and consistent. The QEC code is defined in terms of code stabilisers, and the circuit builder module comprises instructions to translate code stabilisers to executable instructions.

- 25 In another embodiment, a custom or new QEC code may be defined, the data representation module is equipped with a set of validator tools in order to make sure that archetypes, embedding, and operation are consistent and define a valid QEC code.

- 30 In an embodiment, the code builder module 104 may allow the user to obtain a QEC code by: selecting it from a set of commonly used, predefined codes; selecting it from valid codes created from the user's previous work, which have been saved in a suitable format; constructing a new candidate code, which can furthermore be verified to be valid or invalid, by checking consistency of the defined stabilisers and the logical qubit encoding.

The custom QEC code may be defined by the user. The custom QEC code may be defined by specifying a set of custom stabiliser operators. When a custom QEC code is defined, it

will be necessary to compute the corresponding logical Pauli operators to enable the checking of the ECCV conditions. The logical operators may be provided explicitly by the user. Alternatively, the logical operators may be computed using Algorithm 2 disclosed herein. During the execution of Algorithm 2, Algorithm 1 is also used.

- 5 In an embodiment, novel algorithms are provided to assist the workflow of system 100.

Data structures:

To represent the Pauli string Operators acting on N qubits, row bit-vectors of length $L=2N+1$ may be used, consisting of N bits for X values, N bits for Z values, and one bit for the sign. Each pair of (Z,X) bit-values may represent a single-qubit Pauli operator ($00 \rightarrow I$, $01 \rightarrow Z$, $10 \rightarrow X$, $11 \rightarrow Y$).

10

To represent a stabiliser set describing a N-qubit state, the N stabilisers may be stacked forming an array with N rows and $2N+1$ columns. The stabiliser set is sometimes referred to as the stabiliser array, and for the purposes of this disclosure the two terms should be considered interchangeable.

- 15 Valid stabiliser array operations:

Valid stabiliser array operations may include row-swapping and/or multiplying a row into another one, while respecting Pauli multiplication rule:

$$\sigma_j \sigma_k = \delta_{jk} I + i \epsilon_{jkl} \sigma_l$$

where $\sigma_j = \{X \text{ for } j=1, Y \text{ for } j=2, Z \text{ for } j=3\}$

- 20 Stabiliser Set Canonical form:

Every stabiliser set has a canonical form which is the reduced row-echelon form, which is obtain after implementing stabiliser binary Gaussian elimination on the stabiliser set array. Note that this differs from the standard binary Gaussian elimination, as signs need to be tracked when combining rows.

- 25 Characteristic indices:

If an array is in its canonical form, the column index for every row in which the first occurrence of 1 in the row is encountered may be found. These indices may be referred to as “characteristic indices” of the array.

Algorithm 1. Destabiliser Set Finding

In an embodiment, the code builder module 104 is configured to create a destabiliser set corresponding to the stabiliser set, wherein each element of the destabiliser set forms a pair with a corresponding element of the stabiliser set, each element of the stabiliser anti-commutes with a corresponding paired element of the destabiliser set, and commutes with other elements of the destabiliser set. A union of the stabiliser set and destabiliser set generates Pauli operators of the N-qubit Pauli group.

The destabiliser set $D = \{d_i \mid i=0, 1 \dots N-1\}$ for a set of stabilisers $S = \{s_i \mid i=0, 1 \dots N-1\}$ is a set of multi-qubit Pauli Operators for which the following conditions hold: d_i commutes with d_j for all i, j and d_i anti-commutes with s_i but commutes with all s_j where $j \neq i$. This means that the union of the sets S and D can generate any Pauli operator of the N-qubit Pauli group.

The following method ensures all the required properties of the destabilizer are fulfilled. This can be achieved by making modifications that do not violate previously achieved conditions. The method is as follows:

1. Make an array D of the same dimensions as S , filled with zeros.
2. Populate D such that $S+D$ generates the Pauli group. This may be done by putting S in its canonical form and finding which column indices are not “characteristic”. Populate D with 1’s on the column indices found. D is now a set of single qubit operators.
3. Repair or correct the commutation relations between the elements of D , while making necessary modifications such that $S+D$ still generates the Pauli group. This may be done by finding for every non-commuting pair of elements in D a qubit that has not been indexed by D and using this qubit to repair the commutation relation.
4. Repair or correct the commutation relations between the elements of D and S . This may be done by taking one by one the elements i of S and ensuring that every element d_j , $j > i$ satisfies the conditions by using valid stabiliser operations.
5. The destabilizer array should now be correct.

An example, using a 4-qubit code logical state will now be presented.

In steps 1 and 2, the stabiliser array is composed of:

the code stabilisers [+ZZZZ, +XXXX]

the logical operators [+ZZII, +ZIZI]

The array representation is:

+ZZZZ	0	0	0	0	1	1	1	1	0
+XXXX	1	1	1	1	0	0	0	0	0
+ZZII	0	0	0	0	1	1	0	0	0
+ZIZI	0	0	0	0	1	0	1	0	0

The canonical form of this array is:

+XXXX	1	1	1	1	0	0	0	0	0
+ZIIZ	0	0	0	0	1	0	0	1	0
+IZIZ	0	0	0	0	0	1	0	1	0
+IIZZ	0	0	0	0	0	0	1	1	0

Note that it is equivalent to the above, but it is in its reduced row-echelon form. The characteristic indices are [0,4,5,6]. As such, D can be initialised with ones in [1,2,3,7]:

+IXII	0	1	0	0	0	0	0	0	0
+IIXI	0	0	1	0	0	0	0	0	0
+IIIX	0	0	0	1	0	0	0	0	0
+IIIZ	0	0	0	0	0	0	0	1	0

5

In step 3, D+S generates the full Pauli Group. However, the last two operators of D anti-commute since they both make use of the fourth qubit. To repair this anti-commutation, the first qubit that has not been indexed, will be utilised.

Transforming:

$$+IIIX \rightarrow +PIIX$$

$$+IIIZ \rightarrow +P'IIIZ$$

- With a choice of anti-commuting P, P' will make the two operators commute. For example, for $(P, P') = (Y, X)$ we get: $[+YIIIX, +XIIIZ] = 0$. Note that there are 6 such combinations and one
- 5 of the combinations need to be selected that will allow $S+D$ to still generate the Pauli group after the transformation. In this particular case, it turns out that this happens for $(P, P') = (Z, Y)$. Thus, all operators of D commute with each other and D obtains the form:

+IXII	0	1	0	0	0	0	0	0	0
+IIXI	0	0	1	0	0	0	0	0	0
+ZIIIX	0	0	0	1	1	0	0	0	0
+YIIIZ	1	0	0	0	1	0	0	1	0

- In step 4, D transforms using valid stabilizer operations, such that the commutation relations with S are satisfied. To satisfy the commutation relations with the first stabilizer $+ZZZZ$, the
- 10 first destabilizer is kept as it is since it anti-commutes and then the first destabilizer is multiplied into all other operators that anti-commute so that they are commuting afterwards. Thus D becomes:

+IXII	0	1	0	0	0	0	0	0	0
+IXXI	0	1	1	0	0	0	0	0	0
+ZXIX	0	1	0	1	1	0	0	0	0
+YXIZ	1	1	0	0	1	0	0	1	0

Now, the second stabilizer +XXXX commutes with the second destabilizer +IXXI so we shall multiply this row with one that does not. We will pick +ZXIX and then we shall multiply the result (+ZIXX) into the other operators that do not commute with +XXXX.

After this transformation, D becomes:

+IXII	0	1	0	0	0	0	0	0	0
+ZIXX	0	0	1	1	1	0	0	0	0
+IXXI	0	1	1	0	0	0	0	0	0
+YXIZ	1	1	0	0	1	0	0	1	0

5

Continuing this, D eventually becomes:

+YXXZ	1	1	1	0	1	0	0	1	0
+XXXY	1	1	1	1	0	0	0	1	0
+YIXZ	1	0	1	0	1	0	0	1	0
+YXIZ	1	1	0	0	1	0	0	1	0

Which satisfies all the necessary conditions.

Algorithm 2. Logical Operator Finding

- 10 In an embodiment, an algorithm is provided for finding logical operators. The code builder module 104 is configured to find logical operator sets for the stabiliser set, wherein all elements of the logical operator sets commute with the elements of the stabiliser set. the logical operator sets comprises a first logical operator set and a second logical operator set, wherein the each element of the first logical operator set forms a pair with a corresponding element of the second
- 15 logical operator set, each element of the first logical operator set anti-commutes with a

corresponding paired element of the second logical operator set, and commutes with other elements of the logical operator set.

QEC code Q need not have its Pauli logical operators defined explicitly, as these can be computed by the invention. This may be beneficial when the logical operators are not obvious to the user or when testing syndrome extraction circuits where the logical operators are not a necessary input.

Given the code stabilisers of a code S , a set of logical operators L may be found. Assuming that the code involves N data qubits and contains k stabilisers, then $n=N-k$ logical qubits are encoded. n L_z and n L_x operators are identified such that: L_{z_i} anti-commutes with L_{x_i} , L_{z_j} commutes with L_{x_j} , $j \neq i$, and all elements of L_z and L_x commute with all elements of S .

The method is as follows:

1. Find a stabiliser set A corresponding to a logical state. This may be done by projecting any N -qubit state onto the code subset (e.g. by measuring the stabilisers of the code S), one is left with the state being described by A .
2. Find the logical operators L_a in A . This may be done by removing from A all information contained in the code stabilisers S . This can be done by putting both A and S in their canonical form and multiplying rows of S into A . Multiplication may be done such that all columns of A with characteristic indices of S are filled with zeros.
3. Arbitrarily name L_a as L_z .
4. Stack vertically arrays L_z and S to form array B , an equivalent representation of A .
5. Find the destabiliser array D_B of B .
6. The first n rows of D_B represent L_x .

Thus, for an arbitrary S , the logical operators L_z , L_x are found.

An example, finding the logical operators of the 4 qubit code will now be presented.

In step 1, start with state $|0000\rangle$ and measure the 2 code stabilisers $S=\{+ZZZZ, +XXXX\}$:

$ZZZZ$ will have eigenvalue $+1$

$XXXX$ will have a random eigenvalue. (let's assume -1)

This leaves the state in the stabiliser state: $A = \{+ZZII, +ZIZI, +ZIIZ, -XXXX\}$

In step 2, using the canonical form of A and S, all information that is contained in S may be removed from A. A in canonical form:

-XXXX	1	1	1	1	0	0	0	0	1
+ZIIZ	0	0	0	0	1	0	0	1	0
+IZIZ	0	0	0	0	0	1	0	1	0
+IIZZ	0	0	0	0	0	0	1	1	0

S in canonical form is:

+XXX X	1	1	1	1	0	0	0	0	0
+ZZZZ	0	0	0	0	1	1	1	1	0

5

S has characteristic indices 0 for +XXXX and 4 for +ZZZZ so they may be multiplied into the canonical form of A, wherever there is a 1 in the characteristic index.

This gives us the array B:

-IIII	0	0	0	0	0	0	0	0	1
+IZZI	0	0	0	0	0	1	1	0	0
+IZIZ	0	0	0	0	0	1	0	1	0
+IIZZ	0	0	0	0	0	0	1	1	0

10 Putting B in canonical form gives us:

+IZIZ	0	0	0	0	0	1	0	1	0
+IIZZ	0	0	0	0	0	0	1	1	0
-IIII	0	0	0	0	0	0	0	0	1
+IIII	0	0	0	0	0	0	0	0	0

The last row does not give any information about the stabilizer set since it's an empty row. The last row may be deleted.

- 5 The second to last row, that is -IIII, is the result of one stabilizer (+XXXX) being projected onto its negative value. As a result, this row may also be deleted.

Eventually, this will result in the array L_a, that contains logical operators of the code:

+IZIZ	0	0	0	0	0	1	0	1	0
+IIZZ	0	0	0	0	0	0	1	1	0

In step 3, the operators +IZIZ, +IIZZ are defined to be the Z operators for the first and second logical qubit respectively.

- 10 In step 4, the array Q is defined as:

+IZIZ	0	0	0	0	0	1	0	1	0
+IIZZ	0	0	0	0	0	0	1	1	0
+XXX X	1	1	1	1	0	0	0	0	0
+ZZZZ	0	0	0	0	1	1	1	1	0

In step 5, the destabilizer array of Q is found, which evaluates to:

+YXIZ	1	1	0	0	1	0	0	1	0
+YIXZ	1	0	1	0	1	0	0	1	0
+XXX Y	1	1	1	1	0	0	0	1	0
+YIIZ	1	0	0	0	1	0	0	1	0

In step 6, the corresponding operators of the destabilizer array of Q are identified as the L_x operators. In particular, YXIZ and YIXZ are the X logical operators for the first and second logical qubit respectively. These operators can be redefined/simplified by multiplying them by stabilisers or the Z logical operators.

Circuit builder module 105

Figure 6A shows the system 100 further comprises a circuit builder module 105. Figure 6B shows the circuit builder module 105 may be configured to build and/or receive an input circuit, and output the input circuit to the data structure representation 101. The circuit builder module 105 may allow the user to build a circuit with a certain syntax. The circuit builder module 105 may receive an input circuit in a standard format, such as Qiskit or QASM, and convert the circuit to an internal representation used by system 100.

Figure 7 shows the circuit builder module 105 may be configured to specify the input circuit and the associated metadata, for example, data qubits, ancilla qubits, classical register variables, data-to-ancilla map, ancilla-to-classical register map.

The circuit builder module 105 comprises abstractions and convenience functions. These abstractions and convenience functions are used when creating the input circuit at scale. A convenience function may be used, for example, to rapidly insert a block of standard/common operations into the input circuit. The abstractions may be, for example, sequences of quantum gates that achieve some specific outcome. This, for example, may be a sub-operation. Many such sub-operations may be composed together to build the target operation O. An example, sub-operation may be a lattice surgery operation such as Grow, Shrink, Merge, or Split. These lattice surgery operations are understood in the standard

surface code literature. The operation O may be a logical CNOT gate. Different sequences and/or combinations of Grow, Merge, Shrink, Split may be assembled to achieve the overall effect of a logical CNOT gate.

5 The circuit builder module 105 is configured to support construction of the input circuit from pre-defined quantum gates, and/or user-defined groupings of quantum. A single quantum gate may be a single operation, for example, any of the standard quantum gates that are common to the literature (e.g. CNOT, Hadamard, T-gate, ...). A grouping of quantum gates may be a combination of such gates.

10 To prepare a valid input for the system 100, a valid data structure representation 101 Q-Loom circuit, which corresponds to the input circuit may be specified. The valid data structure representation 101 Q-Loom circuit may be specified by the user. The input circuit C in its Q-Loom format may be obtained from a standard quantum programming language, such as QASM or QIR, and using the circuit builder module 105 to construct the input circuit. The circuit builder module 105 may support programmatic and/or visual
15 construction of the circuit.

Operation builder module 106

Figure 6A shows the system 100 further comprises an operation builder module 106. Figure 6B shows the operation builder module 106 may be configured to define the operation, and output the operation to the data structure representation 101. The intended circuit operation
20 O determines the criteria to be used to verify whether the input circuit indeed fulfils its function. To prepare a valid input for the system 100, a valid data structure operation, which corresponds to the operation O, may be specified. The valid data structure operation may be specified by the user.

The operation builder module 106 is configured to provide an input mechanism for the
25 information or data relating to the operation O.

Operations supported by system 100 comprises at least one of the following fundamental components of QEC: syndrome measurement, logical gate application, logical state preparation, and/or QEC code switching.

30 Currently, there are no clear statements of the ECCV problem in the prior art. By extension, no systematic way of solving the problem exists in the prior art either. The disclosed system is the first end-to-end and self-contained system for solving the ECCV problem.

Three elements of the innovation provided by the disclosed system are:

- The recognition of the ECCV problem as a fundamental step that must be tackled in order to design and discover circuits for fault-tolerant quantum computation
 - The specification of a set of conditions that must be respected by a valid circuit for the intended operation O and error correction scheme Q being employed
- 5 • The design of a coherent system for checking the ECCV conditions.

The disclosed has the potential to: a) empower experts to significantly augment their research output and enable them to make new discoveries, and; (b) lower the barrier to entry for performing research in QEC, allowing researchers with a wider diversity of scientific backgrounds to grow their expertise and make their own discoveries. In the context of the growing quantum computing industry, the disclosed system therefore has clear industrial relevance.

Both the data representation module and the output of the system are taggable and serialisable, and can thus be easily input to a database to facilitate data mining and machine learning activities. Specifically, the tags work as explicit labels which are necessary to leverage well known classical machine learning techniques. One could build a database of circuits that pass / fail the ECCV Conditions. This database could then be analysed by machine learning methods for any patterns in valid circuits. This could then form the basis of the construction of generative models for outputting new circuits.

Unless the context clearly requires otherwise, throughout the description and the claims, the words "comprise", "comprising", and the like, are to be construed in an inclusive sense as opposed to an exclusive or exhaustive sense, that is to say, in the sense of "including, but not limited to".

Reference to any prior art in this specification is not, and should not be taken as, an acknowledgement or any form of suggestion that that prior art forms part of the common general knowledge in the field of endeavour in any country in the world.

The disclosed system and method may also be said broadly to consist in the parts, elements and features referred to or indicated in the specification of the application, individually or collectively, in any or all combinations of two or more of said parts, elements or features.

Where, in the foregoing description reference has been made to integers or components having known equivalents thereof, those integers are herein incorporated as if individually set forth.

It should be noted that various changes and modifications to the presently preferred embodiments described herein will be apparent to those skilled in the art. Such changes and

modifications may be made without departing from the spirit and scope of the disclosed apparatus and systems and without diminishing its attendant advantages. For instance, various components may be repositioned as desired. It is therefore intended that such changes and modifications be included within the scope of the disclosed systems and method. Moreover, not

5 all of the features, aspects and advantages are necessarily required to practice the disclosed systems and method. Accordingly, the scope of the disclosed systems and method is intended to be defined only by the claims that follow.

Claims

1. A system for verification of a quantum error correction circuit, the system comprising:
 - a data structure representation configured to receive and/or store a quantum error correction code, an operation performed within the quantum error correction code, and an input circuit,
 - an output module configured to prepare simulations of the input circuit,
 - a simulator configured to simulate the input circuit, andwherein the output module is configured to output at least one Boolean variable indicating whether the input circuit passes error correction circuit verification (ECCV) conditions, wherein the output module is further configured to provide debugging information when at least one Boolean variable indicates the input circuit fails at least one of the ECCV conditions.
2. The system of claim 1, wherein the data structure representation comprises validation checks for verifying the data correctness and/or consistency of at least one of the following: the input circuit, the quantum error correction code, and/or the operation.
3. The system of claim 2, wherein when the validation checks indicate incorrect data, an error message is returned through the output module.
4. The system of claim 2 or 3, wherein the data structure representation comprises a cross-referencing system for checking at least one of the following: the input circuit, the quantum error, and/or the operation.
5. The system of any one of claims 1-4, wherein the data structure representation comprises information to serve as inputs in the output module, and/or the simulator.
6. The system of any one of claims 1-5, wherein the data structure representation is further configured to communicate with the output module, and/or the simulator.
7. The system of any one of claims 1-6, wherein the data structure representation comprises a compressed and/or modular representation of instructions to execute operations of the quantum error correction code.
8. The system of any one of claims 1-7, wherein the output module is configured to interact with the simulator through the data structure representation.

9. The system of any one of claims 1-8, wherein the system further comprises a code builder module configured to define the quantum error correction code, and output the quantum error correction code to the data structure representation.
10. The system of claim 9, wherein the code builder module is configured to allow a user-defined error correction code as the quantum error correction code.
11. The system of claim 10, wherein the code builder module is configured to verify the quantum error correction code, and/or a set of logical Pauli operators of the quantum error correction code for consistency.
12. The system of any one of claims 9-11, wherein the code builder module is configured to define the quantum error correction code as a stabiliser set comprising elements of N-qubit stabiliser state.
13. The system of claim 12, wherein the code builder module is configured to create a destabiliser set corresponding to the stabiliser set,

wherein each element of the destabiliser set forms a pair with a corresponding element of the stabiliser set, each element of the stabiliser anti-commutes with a corresponding paired element of the destabiliser set, and commutes with other elements of the destabiliser set.
14. The system of claim 12 or 13, wherein the code builder module is configured to find logical operator sets for the stabiliser set,

wherein all elements of the logical operator sets commute with the elements of the stabiliser set, and

wherein the logical operator sets comprises a first logical operator set and a second logical operator set, each element of the first logical operator set forms a pair with a corresponding element of the second logical operator set, each element of the first logical operator set anti-commutes with a corresponding paired element of the second logical operator set, and commutes with other elements of the logical operator set.
15. The system of any one of claims 1-14, wherein the system further comprises a circuit builder module configured to build and/or receive an input circuit, and output the input circuit to the data structure representation.
16. The system of claim 15, wherein the circuit builder module comprises abstractions and convenience functions.

17. The system of claim 15 or 16, wherein the circuit builder module is configured to support construction of the input circuit from pre-defined quantum gates, and/or user-defined groupings of quantum.
18. The system of any one of claims 1-17, wherein the system further comprises an operation builder module configured to define the operation, and output the operation to the data structure representation.
19. The system of any one of claims 1-18, wherein the ECCV conditions comprise:
 - a first condition, wherein the input circuit passes the first condition when an output state of the input circuit is encoded within the quantum error correction code,
 - a second condition, wherein the input circuit passes the second condition when the output state of the input circuit has the same logical operators as the quantum error correction code, and
 - a third condition, wherein the input circuit passes the third condition when the input circuit is extracting syndrome information.
20. The system of claim 19, wherein the input circuit comprises a plurality of sub-circuits, and the output module is configured to output Boolean variables indicating whether at least one sub-circuit of a plurality of sub-circuits passes the first condition, the second condition, and the third condition.
21. The system of any one of claims 1-20, wherein the ECCV conditions comprises:
 - a first condition, wherein the input circuit passes the first condition when an output state of the input circuit is encoded within the quantum error correction code,
 - a second condition, wherein when the input circuit contains syndrome measurements, the input circuit passes the second condition when the input circuit is extracting syndrome information, and/or
 - a third condition, wherein the input circuit passes the third condition when its action on each member of a set of input states results in an output state that is consistent with the expected transformation rules.
22. The system of any one of claims 1-21, wherein the output module is configured to check the ECCV conditions for a logical Clifford operation, wherein the logical Clifford operation comprises a user-supplied set of expected transformation rules, the

expected transformation rules describing theoretical transformation of logical Pauli operators by the Clifford operation.

23. The system of any one of claims 1-22, wherein the data structure representation comprises information to serve as inputs in at least one of the following: a circuit builder module, a code builder module, and/or an operation builder module.
24. The system of any one of claims 1-23, wherein the data structure representation is further configured to communicate with at least one of the following: a circuit builder module, a code builder module, and/or an operation builder module.
25. A method for verifying a quantum error correction circuit, the method comprising:
 - receiving and/or storing a quantum error correction code, an operation performed within the quantum error correction code, and an input circuit,
 - simulating the input circuit,
 - outputting at least one Boolean variable indicating whether the input circuit passes ECCV conditions, and
 - providing debugging information when at least one Boolean variable indicates the input circuit fails at least one of the ECCV conditions.