A Quantum Computational Compiler and Design Tool for Technology-Specific Targets

Kaitlin N. Smith and Mitchell A. Thornton
Quantum Informatics Research Group
Southern Methodist University
Dallas, Texas, USA
{knsmith, mitch}@smu.edu

ABSTRACT

Quantum computing, once just a theoretical field, is quickly advancing as physical quantum technology increases in size, capability, and reliability. In order to fully harness the power of a general quantum computer or an application-specific device, compilers and tools must be developed that optimize specifications and map them to a realization on a specific architecture. In this work, a technique and prototype tool for synthesizing algorithms into a quantum computer is described and evaluated. Most recently reported methods produce technologically-independent reversible cascades comprised of a functionally complete set of operators with no regard to actual technologically-dependent cell libraries or constraints due to a device's pre-configured interconnectivity. In contrast, our prototype tool synthesizes algorithms into technologically-dependent specifications that consist of a set of primitives and connectivity constraints present in the computer architecture. The tool performs optimizations based on actual architectural constraints, and a high-quality technology-dependent synthesized result is achieved through the use of optimizing cost functions derived from real hardware and architecture parameters. Additionally, another important aspect of our tool is the incorporation of internal formal equivalence checking that ensures the initially specified algorithm is functionally equivalent to the optimized, technologically-mapped output. Experimental results are provided that target the IBM Q family of quantum computers.

CCS Concepts

•**Hardware** → **Quantum computation;**

Keywords

Quantum information science, QIS, quantum computing, logic synthesis, compilation

1. INTRODUCTION

The field of quantum computation is of great interest since a quantum computer (QC) can complete tasks such as searching large data sets, accurately simulating quantum behavior, and other applications with significant performance increases as compared to classical approaches. Currently, there are various architectures for qubit representation that are competing to become the standardized quantum computing platform. This variety between machines, even between those using the same underlying technology, causes issues during the design process. For example, several different transmon-based QCs have been developed by IBM. Although these implementations use the same type of transmon qubit technology, differences in architecture create compatibility issues which in many cases prohibit the use of designs originally mapped for one QC to properly target another transmon-based device. This challenge motivates the development of techniques to automatically map and optimize quantum circuits, or programs, to forms that are technologically dependent and physically executable. By coupling this technology mapping and optimizing backend to a QC compiler front-end, a complete quantum logic synthesis tool results.

Special techniques are required to enable a synthesis tool to map a generalized quantum circuit to the limited library of transformations that are available on a specific QC architecture. The tool must reconfigure a circuit meant for one technology so that it can be realized on another device, including the mapping of classical algorithms to quantum machines. This technology mapping scenario motivates the development of an automated quantum logic synthesis tool, or hardware compiler, for technology-dependent targets. Furthermore, the use formal equivalence checking confirms that the generated technology-dependent circuits are logically identical to their original, technology-independent specifications. The prototype compilation tool also incorporates optimizations to boost output circuit performance. Optimization procedures are applied in both the technologically-independent intermediate form and the technologically-dependent final result.

Quantum synthesis methodologies have been developed in the past that generate reversible logic [1, 2], consider layout restrictions of a grid-based quantum structure [3, 4], and map to one- and two-qubit elementary gates [5, 6]. The technology-dependent quantum design automation tool described here differs from these past efforts in two important aspects: 1) the resultant quantum network is mapped to an architecture with technologically-dependent operators and interconnects, and, 2) the tool includes built-in formal equivalence checking to confirm that the generated final circuit is functionally equivalent to its original, technologically-independent form. High-quality quantum circuit synthesis and mapping is achieved through the use of formal verification. The Quantum Multiple-valued Decision Diagram (QMDD), described in [7], is implemented during compilation for efficient circuit equivalence checks.

The prototype tool described here maps, optimizes, and verifies transformations of arbitrary quantum circuits into algorithms that are executable on real QCs. These QCs are characterized by limited connectivity for multi-qubit operations due to intrinsic architectural

ISCA '19, June 22–26, 2019, PHOENIX, AZ, USA
© 2019 ACM. ISBN 978-1-4503-6669-4/19/06…$15.00
DOI: https://doi.org/10.1145/3307650.3322262

constraints of the chosen topology. As a result, we describe our new contribution of a tool that allows end-to-end quantum synthesis and compilation that includes formal verification via the QMDD. The IBM Q family of quantum machines was used as an example target technology as information about this platform is readily available. The tool, however, is modular in the sense that custom transmon-based topologies may also be targeted for synthesis that are outside the IBM family of quantum devices. This paper progresses as follows: Section 2 provides background information on quantum logic, computing, and operations as well as introduces the QMDD data structure. Section 3 describes the IBM quantum machines and tools. The methodology for technology-dependent quantum logic synthesis and compilation is outlined in Section 4. Experimental results are reviewed in Section 5. Section 6 draws conclusions for the current study as well as details future developments and improvements for the quantum synthesis and compilation tool.

2. BACKGROUND

This section includes a brief survey important concepts in quantum information processing (QIP), QC architecture, and quantum function representation, manipulation, and data structures. It is assumed that the reader has a basic understanding of QIP and QC, thus this survey is not exhaustive.

2.1 Quantum Computation

The unit of quantum information is the quantum bit, or qubit. A qubit models information as values such as $|0\rangle$ or $|1\rangle$ which are a set of orthonormal basis states in Dirac notation that represent the two-dimensional column vectors of $|0\rangle = \begin{bmatrix}1 & 0\end{bmatrix}^T$ and $|1\rangle = \begin{bmatrix}0 & 1\end{bmatrix}^T$, respectively. The qubit can theoretically represent an infinite continuum of states while in a superposition of the of basis states, $|0\rangle$ and $|1\rangle$, as shown by

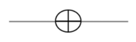
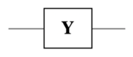
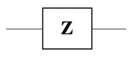
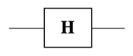
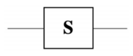
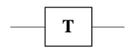
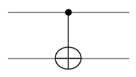
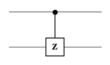
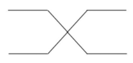
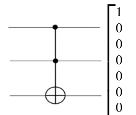
$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{1}$$

In Eqn. 1, α and β are complex-valued probability amplitudes for the basis states in the state vector. During measurement, the probability that $|\psi\rangle = |0\rangle$ is $\alpha^*\alpha = |\alpha|^2$ and the probability that $|\psi\rangle = |1\rangle$ is $\beta^*\beta = |\beta|^2$ where * indicates the complex conjugate. Qubits collapse into a basis state after observation, and their quantum superposition and entanglement properties are lost.

Quantum operators represent specific time evolutions of the quantum state and thus purposely transform qubit state so that meaningful QIP can occur. If a quantum algorithm is modeled as a circuit, quantum operations can be viewed as quantum logic gates. Therefore, the terms of "quantum operator" and "quantum gate" are often used interchangeably. These operators are represented by a unique, unitary transformation matrix, **U**. Common single- and multi-qubit operations are provided in Table 1. It should be noted the Toffoli operation can be generalized, $\mathbf{T}_n$, where the number of controls, $n-1$, is greater than two so that the total number of qubits it transforms, n, is $n > 3$. The Toffoli and generalized Toffoli gate are important operators in many classical to quantum logic synthesis techniques as a result of the method in [1]. These gates, however, are often unsupported by the native gate set of actual quantum machines.

Qubits and quantum operations can be physically realized with a variety of different physical technologies such as with atomic particles, (*i.e.*, electrons, photons, *etc.*), or as superconducting solid-state circuits such as the transmon [8, 9]. Quantum decoherence is a phenomenon that is characteristic to a specific architecture, and some quantum systems are more resilient to decoherence as compared to others. In QIP circuit and QC design, there is a trade-off between average decoherence times and the ability for qubits to interact. Multi-qubit interaction is necessary to enable required operations such as the **CNOT** or controlled-phase, **CZ**. The ability for qubits to couple, however, implies that qubits also have the undesirable capability to interact with the environment, increasing the probability of decoherence. Determining qubit realizations that allow for easy interaction with one another while also resisting environmental coupling is currently an active area of research that accounts for, in part, the lack of a wide consensus on the preferred implementation of a qubit. Many current QC/QIP realization efforts are focused upon the class of superconducting solid-state qubits as they are considered by many to offer the best trade off between qubit interaction versus average time of decoherence.

Table 1: Common Single- and Multi-Qubit Quantum Operators

Operator	Symbol	Transfer Matrix
Pauli-X (NOT)		$\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$
Pauli-Y (Y)		$\begin{bmatrix}0 & -i\\i & 0\end{bmatrix}$
Pauli-Z (Z)		$\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}$
Hadamard (H)		$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}$
Phase (S)		$\begin{bmatrix}1 & 0\\0 & i\end{bmatrix}$
$\pi/8$ **(T)**		$\begin{bmatrix}1 & 0\\0 & e^{i\pi/4}\end{bmatrix}$
CNOT		$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}$
CZ		$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&-1\end{bmatrix}$
SWAP		$\begin{bmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{bmatrix}$
Toffoli		$\begin{bmatrix}1&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&1&0&0&0&0\\0&0&0&0&1&0&0&0\\0&0&0&0&0&1&0&0\\0&0&0&0&0&0&0&1\\0&0&0&0&0&0&1&0\end{bmatrix}$

2.2 Quantum Cost

Quantum device manufacturers continue to pursue devices that are more robust in terms of size and reliability while QIP researchers address the key goal of enabling algorithms to be automatically synthesized to physical devices. As a result, reducing the probability of decoherence during a QIP task becomes highly important. With current implementations, the quantum state will eventually decohere after some time. Additionally, the likelihood of decoherence increases as a set of qubits undergoes more transformations. Therefore, whenever considering quantum cost reduction, the most common method is usually centered around reducing the total number of operations performed on a set of qubits, especially those that are multi-qubit or those that are expensive in terms of fault tolerance, such as the **T** gate (also known as the $\pi/8$ gate and its adjoint, $\mathbf{T}^\dagger$

or $\pi/8^\dagger$) [10].

Arbitrary n-qubit gates, including the three-qubit Toffoli and its generalized forms, are capable of being decomposed into a collection of single qubit gates along with the two-qubit **CNOT** gate [11]. Due to this property, the cost of a quantum circuit is typically a function of the gate counts of a quantum circuit decomposed into one- and two-qubit primitives because more operations on a quantum state introduce increased opportunity for decoherence. Therefore, if an technology-independent circuit contains many complex multi-qubit gates, a high cost will result as the algorithm is transformed into an expanded, technology-dependent form. An example quantum cost function for use in a compiler or synthesis tool is given as

$$q_{cost} = 0.5 \times t + 0.25 \times c + a. \tag{2}$$

In Eqn. 2, t is the count of all **T** and $\mathbf{T}^\dagger$ gates, c is the count of **CNOT** gates, and a is the total gate count, or gate volume, for a circuit. **T** gates are given an additional cost of 0.5 as compared to all other single qubit gates because of the operator's poor fidelity as compared to other single qubit operators in fault tolerant quantum implementations [10]. **CNOT** gates are given a cost that is 0.25 more than single qubit gates, with the exception of **T**, because two-qubit operations for the target qubit in this paper, the transmon, are characterized by a higher error rate as compared to the other single qubit operations [12]. Our cost function was selected based upon what is commonly seen in the literature: fewer gates usually leads to smaller circuits with less probability of decoherence and fewer **T** gates improves reliability and results in more fault-tolerance. A larger quantum cost indicates a higher likelihood of qubit decoherence and decreased fault tolerance. Quantum cost for a design is minimized by the quantum logic design automation tool during the optimization process. The QC compiler described here similarly uses a quantum cost function to decrease decoherence probabilities, however, when actual devices are targeted, the cost function may also incorporate other terms such as area, power, and interconnectivity costs. For this reason, the compiler can use any arbitrary quantum cost function, and it is expected that each particular technologically-dependent quantum cell library will be characterized and annotated with custom cost functions to be used by the synthesis tool. We are experimenting with other metrics, such as qubit and operator fidelity, rather than decoherence times within our cost evaluations. In this work, Eqn. 2 is used for exemplary purposes due to gate fidelity information found in the literature [10, 12]. However, our prototype allows for users to easily modify cost function weights so that optimization parameters can be customized.

2.3 Reversible Logic

Reversible logic, unlike classic Boolean logic, refers to an implementation where input information can theoretically be recovered by executing the algorithm in reverse. When an algorithm is reversible, the original circuit inputs are produced whenever their corresponding outputs are fed into the generating function's inverse. Reversible switching functions are bijective [1]. A necessary condition of this property requires the resultant circuit to have an equal number of input and output ports. In general, the addition of "ancilla" inputs and "garbage" outputs embeds an irreversible function into a reversible form. A goal of the synthesis tool is to minimize the number of added ancilla and garbage ports.

Reversibility is relevant to this work since all QIP and QC processes are necessarily both logically and physically reversible as a consequence of the postulates and axioms of quantum mechanics. This is easily observed due to the fact that state transformations corresponding to QIP operations are modeled as unitary transformations, $\mathbf{U}^{-1} = \mathbf{U}^\dagger$. Therefore, the automated synthesis method described in this paper must produce reversible transformations that are technology mapped. Furthermore, it is desirable to enable a user to specify irreversible algorithms rather than requiring the manual conversion of natively irreversible tasks into a reversible form prior to invoking the tool to produce a technology-dependent specification.

Algorithms exist that embed binary switching functions into reversible functions. This capibility to create reversible logic is important because it allows operations to be specified for a quantum computer without needing to know extensive details of quantum computing. Once the switching functions have been transformed, they may be transformed into QIP circuits, or QC programs, for evaluation on a quantum machine. An example of one of the first reversible logic generators is the work discussed in [1]. The work in [1] was the first such algorithm that allowed for practical mappings to occur over circuits that were of reasonable sizes. All prior algorithms demonstrated exponential complexity in terms of the number of irreversible function variables n. The algorithm in [1] operates over an input switching function that is represented in an exclusive-OR sum of products (ESOP) form and converts it into a reversible Toffoli cascade. Because the input to the mapping algorithm is in the form of a minimized ESOP rather than a truth table or some other exponentially large representation, functions of several hundred variables can be mapped to a reversible form in fractions of a second of runtime. A later version of the ESOP approach is described in [2]. This program includes the ESOP transformation algorithms based on the work first described in [1] as well as a reformulation of the method that utilizes decision diagram (DD) representations. As is well-known, an ordered binary DD can be viewed as a data structure that represents an irreversible function as a set of disjoint cubes (*i.e.*, each path from the initial node to the terminal-1 node specifies a disjoint cube in the support of the function of interest), a method analogous to the ESOP technique in [1] is formulated based on DD paths rather than textually-specified cubes of an ESOP form. However, a disjoint cube list is usually not as compact as a minimized ESOP cube list, but when the disjoint cube list is represented in DD form, additional savings in memory requirements can be present due to the inherent sharing of isomorphic subgraphs within the reduced DD. Existing quantum synthesis efforts use techniques such as ESOP and DD representations to convert classical specifications to reversible logic [2, 6], but since reversible logic is unrealizable on actual QCs, the resulting outputs are technology-independent. For technology dependence, high-level, many-input gates, such as the Toffoli and its generalized forms, must be decomposed into one- and two-qubit primitive operators using transformations that keep the operational limitations of physical devices in mind. Optimization processes are implemented during this portion of the technology mapping process to assist with minimizing the expanded circuit structures. Such quantum logic synthesis that generates technology-dependent specifications from generalized quantum, and therefore reversible, algorithms is the focus of this work.

2.4 Quantum Multiple-valued Decision Diagrams

As previously mentioned, n-qubit quantum logic gates or operators are functionally described using unitary matrices of size $2^n \times 2^n$. The size of the transfer matrix describing an entire quantum circuit thus grows exponentially as the number of qubits in the function increases. Data structures have been developed that allow these matrices to be represented in a compact form. For example, the QMDD represents quantum transfer matrices efficiently in the form of a directed acyclic graph. The QMDD was first introduced in [7] and is further described in [13].

QMDDs are a collection of nodes, or vertices, and directed edges. Non-terminal vertices represent qubits and have four outgoing edges that serve as one of the four quadrants in a quantum transformation matrix. From left to right, the four edges leaving a non-terminal node represent the sub matrices of $\mathbf{U}_{00}$, $\mathbf{U}_{01}$, $\mathbf{U}_{10}$, and $\mathbf{U}_{11}$ for the quantum transformation matrix $\mathbf{U}$. Since redundancy in the graph is removed and each qubit variable only appears once, the QMDD representing a quantum function becomes compact in size. An example of the **CNOT** operation in the form of a QMDD is shown in Fig. 1 where x_0 represents the control qubit and x_1 represents the target qubit. Dashed lines are included in this illustration to make submatrix values more apparent. The variable order is $x_0 \rightarrow x_1$, so x_0 acts as the initial vertex and x_1 vertices appear afterwards along the path for submatrices that are not equal to the constant zero.

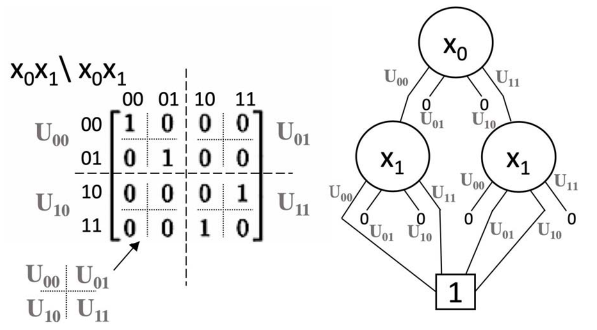


Figure 1: Representation of CNOT Operation as a QMDD.

The QMDD representation of a quantum function is canonical with respect to a fixed variable order due to the reduction rules described in [7]. Even if two circuits with the same transformation matrix are described using different operators, their QMDDs will be equal as long as the variable order used to construct the QMDDs is identical. This concept is used to perform formal verification within the automated quantum logic synthesis tool. An equivalence check between the original technology-independent quantum circuit and the resulting technology-dependent mapped circuit is ensured by requiring the QMDDs to match. If the realizations are indeed identical, the reduction rules for the QMDDs will cause the two designs to share the same graph in memory.

3. IBM Q: A PHYSICAL QC IMPLEMENTATION

IBM has developed multiple QCs and a quantum simulator, and some of these devices are available to the public for experimentation [14]. Because information about the IBM machines is accessible, they were used as the target technology platform during experimentation. Details of the available QCs, both past and current at the time of this work, can be found in Table 2.

IBM implements a type of superconducting semiconductor charge qubit, the transmon, in their devices [12, 15, 16]. The IBM QCs in Table 2, the largest containing 16 qubits, have a variety of available qubit operations that can be used within algorithm realizations. The following gates are included: **NOT** (**X**), **Y**, **Z**, **H**, **S**, $\mathbf{S}^\dagger$, **CNOT** (controlled-**X**), **T**, $\mathbf{T}^\dagger$ phase rotation, amplitude rotation, and measurement.

QCs commonly have a limited range for multi-qubit coupling, and this severely limits the total number of executable multi-qubit operations [3]. The IBM machines demonstrate this constraint

Table 2: IBM Q Device Details (* indicates a retired device) [17, 18, 19, 20, 21]

Name	Release Date	Qubits Supported	Coupling Complexity
ibmqx2 (Yorktown)	Jan. 2017	5	0.3
ibmqx3*	June 2017	16	$0.08\overline{3}$
ibmqx4 (Tenerife)	Sept. 2017	5	0.3
ibmqx5* (Rueschlikon)	Sept. 2017	16	$0.091\overline{6}$
ibmq_16 (Melbourne)	Sept. 2018	14	$0.\overline{098901}$

since the **CNOT** gate is the only available two-qubit gate, and only certain qubits are eligible to act as either a control or a target for this gate. Therefore, **CNOT** placement on the QC is restricted to what is referred to as a coupling map that is set by the layout of the transmon circuits on the QC. The coupling map prevents arbitrary **CNOT** placement. Generalized quantum circuits must be redesigned so that **CNOT** gates are mapped to connected qubits. The coupling maps for the public IBM devices can be represented as dictionaries where **`device`** $= \{a_0 : [b_0], a_1 : [b_1], \ldots, a_{n-1} : [b_{n-1}]\}$. In these dictionaries, the keywords, a_i, are qubits that can act as **CNOT** controls and the paired list, b_i, indicate the qubit(s) that the **CNOT** control can target [17, 18, 19, 20, 21]:

- **`ibmqx2`** = {0:[1,2], 1:[2], 3:[2,4], 4:[2]}
- **`ibmqx3`** ={0:[1], 1: [2], 2:[3], 3:[14], 4:[3,5], 6:[7,11], 7:[10], 8:[7], 9:[8,10], 11:[10], 12:[5,11,13], 13:[4,14], 15:[0,14]}
- **`ibmqx4`** = {1:[0], 2:[0,1], 3:[2,4] 4:[2]}
- **`ibmqx5`**= {1:[0,2], 2:[3], 3:[4,14], 5:[4], 6:[5,7,11], 7:[10], 8:[7], 9:[8,10], 11:[10], 12:[5,11,13], 13:[4,14], 15:[0,2,14]}
- **`ibmq_16`** ={1:[0,2], 2:[3], 4:[3,10], 5:[4,6,9], 6:[8], 7:[8], 9:[8,10], 11:[3,10,12], 12:[2], 13:[1,12]}

The simulator device includes additional gates and qubits that are unrestricted by a coupling map. IBM also has a 20 qubit machine available for commercial use [14].

QCs differ in size and layout, and these variations determine how much a general circuit must be modified in order for it to be realizable on a particular machine. To give designers better insight to the available qubit connections within a machine, we devised a metric referred to as the "coupling complexity." The term complexity was chosen in order to describe the amount of topological interconnects between qubits on a transmon QC. We define coupling complexity as the ratio of the number of allowable **CNOT** couplings found in the coupling map to the total number of two-qubit permutations for a quantum machine. For example, the **`ibmqx2`** machine has 6 couplings on the coupling map and a total of 20 two-qubit permutations:

$$qx2\ coup.\ complex. = \frac{6\ avail.\ couplings}{20\ coupling\ perm.} = 0.3$$

A coupling complexity close to one indicates that a high percentage of a quantum machine's qubits are available for arbitrary two-qubit **CNOT** operations. Simulation devices that assume full connectivity and that thus do not have coupling maps have a coupling complexity equal to one. A coupling complexity close to zero indicates that a low percentage of the qubits can be coupled. As the number of qubits increase in a transmon QC, the coupling complexity decreases because qubits are only able to couple with their connected neighbors for multi-qubit operations. Coupling complexity was calculated for each of the IBM machines where the coupling map was available. This information is found in Table 2.

Circuits, or algorithms, can be loaded onto the IBM QCs and input to the simulator via an available Python API called *Qiskit* (Quantum Information Software Kit) [22]. Circuits can also be created and run using the composer GUI on [14].

4. QUANTUM LOGIC SYNTHESIS METHODOLOGY

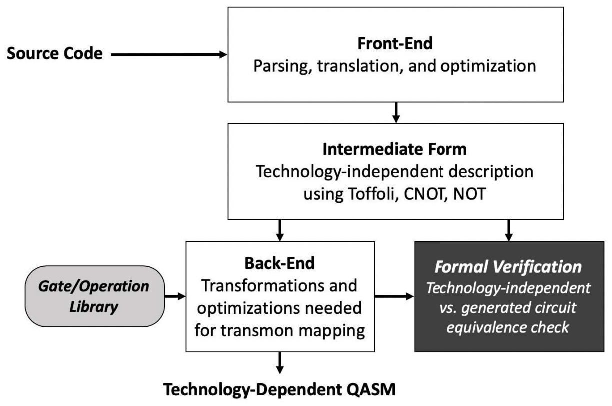

Source Code
Front-End
Parsing, translation, and optimization
Intermediate Form
Technology-independent description using Toffoli, CNOT, NOT
Gate/Operation Library
Back-End
Transformations and optimizations needed for transmon mapping
Formal Verification
Technology-independent vs. generated circuit equivalence check
Technology-Dependent QASM

Figure 2: Synthesis and Compilation Tool Architecture.

A complete flow chart describing the architecture for a quantum logic synthesis and compilation tool is depicted in Fig. 2. This figure illustrates how information is processed through the tool to create the final implementation-specific quantum circuit represented as Quantum Assembly Language, or QASM, code. Initially, the original circuit or algorithm is parsed in as source code. Various file formats are supported for the input specification depending on the type of logic. If the input circuit is in the form of a classical switching function, the front-end will handle the initial parsing and translation of the specification into a reversible cascade of **NOT**, **CNOT**, Toffoli, and generalized Toffoli operators using the algorithm in [1]. The front-end result is a reversible representation of the input circuit that is technology-independent. Reversible code generated by the front-end as well as input circuits already implemented with quantum logic and specified in a quantum instruction language (*i.e.* a .qasm, .qc, or .real file format) are then processed by the back-end of the design tool. The back-end performs transformations and optimizations needed for technology mapping to a specific physical quantum machine. Multi-qubit gate decomposition algorithms, such as those given by Barenco et al. in [11], are representative of some of the transformations implemented in the back-end. However, additional optimizations were devised and implemented to accommodate for device coupling maps that limit qubit connections for two-qubit operations.

A significant drawback of solid-state QCs is that the stationary qubits are limited to certain multi-qubit operations, like those described in coupling maps, due to the layout and properties of the physical device. This limitation, however, extends to other implementations such as optics and trapped ions because qubits must be physically connected, in close proximity, and have the appropriate operational characteristics to realize a multi-qubit operation such as the **CNOT** or other controlled operations in any physical realization. For this reason, a generic reroute algorithm capable of implementing multi-qubit operations on uncoupled qubits for any architecture specified by a coupling map is essential for the technology-dependent quantum logic synthesis tool. Tree data structures assist in finding the shortest **SWAP** route for **CNOT** execution.

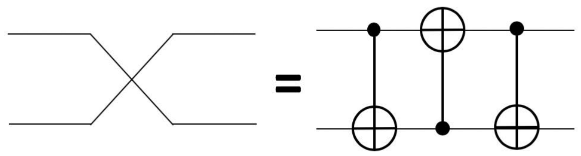


Figure 3: Implementation of SWAP Operation using CNOT.

The connectivity tree reroute algorithm (CTR) was implemented in the quantum synthesis tool to automatically reroute **CNOT** operations that are not supported by a coupling map. In this method, a tree structure based on the coupling map for the selected QC determines the shortest **SWAP** path that the control qubit travels to reposition for a **CNOT** operation. As shown in Fig. 3, a **SWAP** is implemented with **CNOT** gates among physically connected qubits, as indicated by the coupling map, causing the interchange of quantum information. **SWAP** operations continue to move the control qubit until the desired **CNOT** operation can execute on the specified target. After the **CNOT** operation executes, the control qubit traverses the **SWAP** path in reverse to return to its original position in order to preserve the original assignment of qubits in the circuit.

To find the **SWAP** path, CTR builds a tree data structure. The tree root node is the control qubit, and edges leading to other qubit nodes are generated according to the available coupling map configurations. Because of the transformation in Fig. 6, direction of the natively available **CNOT** operation does not matter when building the connectivity tree. In other words, if a qubit $|\psi_0\rangle$ can act as either a control or target in a **CNOT** operation with qubit $|\psi_1\rangle$, the two qubits will be connected with an edge to form a potential **SWAP** path. The tree describes all possible paths that the qubit can take, until the shortest path to a position coupled with the target qubit is found. If a node is reached that is already represented in the tree, the branch is terminated. Pseudocode describing the CTR algorithm is found in Fig. 4.

```
CNOT_w_CTR(control,target,map):
    tree = control
    cnot_execute = False
    while cnot_execute != True:
        tree = build_branches(tree,map)
        if target in tree:
             path = tree[control:target]
             cnot_execute = True
    swap_and_CNOT(control,target,path)
    swap_back(control,target,path)
```

Figure 4: Pseudocode CTR Algorithm.

An example of the CTR algorithm in action can be seen in Fig. 5. In this illustration, the desired operation is a **CNOT** with q5 as the control and q10 as the target on the 16-qubit **ibmqx3** machine. According to the **ibmqx3** coupling map, q5 and q10 cannot natively perform a **CNOT** operation together. After implementing CTR, however, the operation is performed after two swaps of first, q5 with q12 and then second, q12 with q11. Since a connection between q11 and q10 exists on the coupling map, q11 acts as a control for q10. After the desired **CNOT** operation executes, the information of the control qubit swaps back to its original position, q5, on the QC.

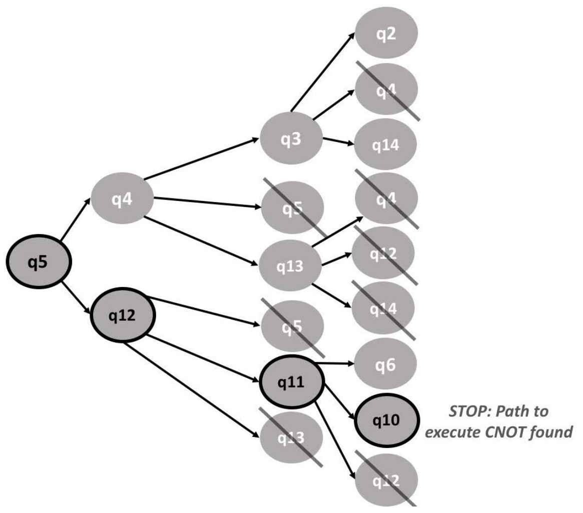


Figure 5: CTR Implementation on the `ibmqx3` Machine for a CNOT with q5 as Control and q10 as Target.

The back-end generates technology-dependent QASM specifications based upon two distinct objectives. The first objective is to produce QASM that conforms to the user-specified target QC's architectural constraints such as a fixed **CNOT** coupling map. The second objective is to determine a mapping that minimizes quantum cost. The quantum cost function is defined in Eqn. 2. A functional prototype for the back-end included in Fig. 2 has been developed, and this design automation tool is the subject of this paper. Results in this work are targeted to the IBM family of QCs as well as machines inspired by the architecture of the IBM QCs, but custom transmon devices with different coupling maps can be added to the tool to provide additional targets during synthesis. The quantum logic synthesis tool implements the following mapping and optimization procedures:

1. **CNOT** operations placed in directions opposite of what is available in coupling map may be reversed [8]. **CNOT** reversal can be seen in Fig. 6.

2. **CNOT** operations on qubits not coupled directly or in reverse on the coupling map are rerouted with CTR.

3. Generalized Toffoli gates are decomposed into Toffoli cascades using [11].

4. Toffoli operations are decomposed into one and two-qubit operators supported by the transmon technology library using transform from [8].

5. Local optimizations based on removing partitions of gates that equal the identity function are implemented recursively until technology library cost function cannot be further reduced.

6. Local optimizations based on removing partitions of gates that can be minimized with an logically identical circuit identity are implemented recursively until technology library cost function cannot be further reduced.

It should be noted that all **SWAP** operations will have a maximum gate count of 7, including four **H** operations and three **CNOT** operations, due to unidirectional transmon **CNOT** operations and the identity pictured in Fig. 6.

After all synthesis and optimization procedures are complete, formal verification is invoked by the compiler. QMDDs are used in equivalence tests to formally verify all technology-dependent compiler outputs. During this process, the original technology-independent specification is compared to the generated technology-dependent specification by building the QMDD data structures. Since the QMDDs share isomorphic subgraphs, the pointers to the original and technology-mapped specification will match if the two designs are functionally identical. The final step of formal verification is critical as it is important that the algorithm's logic is unchanged by the synthesis tool's transformations and optimizations.

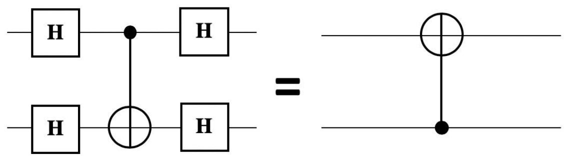


Figure 6: CNOT Orientation Reversal.

5. EXPERIMENTAL RESULTS

Back-end algorithms of the synthesis and compilation tool responsible for mapping and optimizing algorithms for real QC architectures were developed in Python. The tool's purpose is to synthesize technology-dependent algorithms for execution on actual QCs using classical computing methods rather than simulating quantum algorithms. Thus, a personal computer is sufficient for running the quantum compiler with formal verification. Design automation, including formal verification, was performed on a laptop running macOS 10.13.6 with an Intel i5 processor at 2.9 GHz and 8 GB of RAM. Although results in this section include the devices in Section 3 as well as an example 96-qubit IBM-inspired layout, additional architectures can be targeted for synthesis by adding the desired topology coupling map to the device library of the tool.

The first set of benchmarks used during experimentation were obtained from reference [23]. The technology-independent specifications, titled "Optimal Single-target Gates," range from 3 to 6 qubits in size. These important circuits were chosen as benchmarks because they act as essential components for quantum logic synthesis based on lookup-table approaches [6]. Complex reversible and quantum circuits decompose into these functions, and in turn, the single-target gates can be decomposed into one and two qubit operations. These benchmarks were input into the synthesis tool as technology-independent .qc files that contained single qubit operations and **CNOT** gates. When mapped to the simulator, the logic is referred to as technology-independent because is not restricted by the layout of a physical device. The simulator synthesis resulted in technology-independent circuits that match what is featured in [23] with respect to **T** counts and total gate counts because these generated circuits, like the original benchmarks, have no restrictions with respect to where multi-qubit operations are placed. Cost function-based optimization did not reduce the gate counts or the total Eqn. 2 cost of the benchmarks whenever they were mapped to the simulator as the benchmark circuits are already in their most compact and optimal form when qubit connections are unrestricted.

In reality, quantum circuit designers must be careful with gate placement on real QCs due to architectural constraints. These architectural constraints limit what qubit pairs can couple for multi-qubit transformations, and permitted qubit connections for a QC are described in the form of a coupling map. When the single-target gate benchmarks are mapped to real devices, unsupported gate placements must be decomposed or rerouted with **SWAP** operations, as described in Section 4, that cause the circuits to expand. It was noted that QCs with a lower coupling complexity usually required more gates to achieve a technology-dependent mapping. After mapping finishes, the resulting circuit may be optimized using built-in local optimizers. Synthesis results of the "Optimal Single-target Gates" mapped to the IBM devices can be found in Table 3. This table includes both pre- and post-optimization metrics for T-count, total gate count, and cost calculated using Eqn. 2 for each of the generated designs. Technology-independent (i.e. the original, unmapped) metrics for T-count, total gate count, and cost for the benchmarks for unoptimized and optimized mappings were included as well for comparison. A comparison can be made here because these benchmark circuits are already fully optimized in a gate library suitable for IBM that includes one- and two-qubit gates when they are not constrained by any sort of connection restrictions. When these circuits are mapped to a real machine, however, they expand in size because the circuits must be reconfigured to be executable. After original mapping, optimizations help reduce the overall gate counts and cost.

A few observations can be made from analyzing Table 3. First, some designs were not synthesizable, as indicated by N/A, if the target QC was too small for a circuit (i.e. 5 qubit machines cannot support a circuit with n qubits where $n > 5$). Second, technology mapping processes during compilation caused circuits in most cases to expand. This expansion, sometimes of order $\times 10$ in size, was caused by the need to reroute **CNOT** operations to qubits where they could execute. Fortunately, out of the 94 output technology-dependent designs, 74, or approximately 79%, were improved when optimization algorithms based on minimizing the transmon cost function were implemented. Designs that improved in cost post-optimization are emphasized in Table 3. Information about the post-optimization cost function improvement for the technology-dependent "Optimal Single-target Gates" is found in Table 4. The greatest average percent decrease in cost was about 8.5% for the circuits mapped to the **`ibmq_16`** QC. The average post-optimization improvement for all of the technology-dependent forms of the reference [23] benchmarks was approximately 7%.

The second set of benchmarks tested on the automatic quantum logic synthesis tool was a small set of Toffoli cascades from [24]. Toffoli cascade circuits are widely used to describe technology-independent reversible and quantum logic algorithms in the form of **NOT**, **CNOT**, Toffoli, and generalized Toffoli operations. These circuits were used with the tool to demonstrate the Toffoli and general Toffoli decomposition techniques. In their original form, these technology-independent Toffoli cascades include larger, multi-qubit gates that are not supported by the IBM transmon gate library. The benchmarks may seem small because of their gate count, but they are actually complex functions to implement because of the required multi-qubit interaction. Synthesis results of the Toffoli cascade benchmarks mapped to the IBM devices can be found in Table 5. It should be noted that these results do not include a column for technology-independent, unoptimized and optimized mappings as seen in Table 3 because the Toffoli gate is not present in the IBM gate library and is therefore not a technology-ready gate, even when connections between the qubits on the quantum device are disregarded.

In Table 5, circuits that could not synthesize because they were too large for an architecture were once again indicated by N/A. These experiments demonstrated that Toffoli decomposition followed by mapping procedures caused circuits to expand in gate count up to orders of magnitude of $\times 10^2$ their original size. After optimization, 100% of the mapped Toffoli cascades in Table 5 decreased in size. Information detailing the post-optimization improvement of the Toffoli cascade benchmarks is found in Table 6. The greatest average percent decrease in cost was nearly 30% for the circuits mapped to the **`ibmqx3`** machine. The average post-optimization improvement for all of the reference [23] benchmarks was approximately 17.4%.

Most technology-dependent specifications in Table 3 and Table 5 were generated in approximately 10^{-2} seconds, but a few of the larger benchmarks with more multi-qubit gates required a few seconds with none exceeding 5 seconds for synthesis procedures. All outputs were confirmed to be the same function as their original technology-independent description by building the QMDD data structure for each design and testing for equivalence.

The size of quantum architectures is anticipated to increase, so it is important that design tools are able to scale. To test the synthesis and compilation tool, a 96-qubit transmon-based QC architecture was designed and loaded into the tool. The qubits of the machine ranged from q0-q95, and the coupling map, pictured in Fig. 7, was inspired by the **`ibmqx5`** machine.

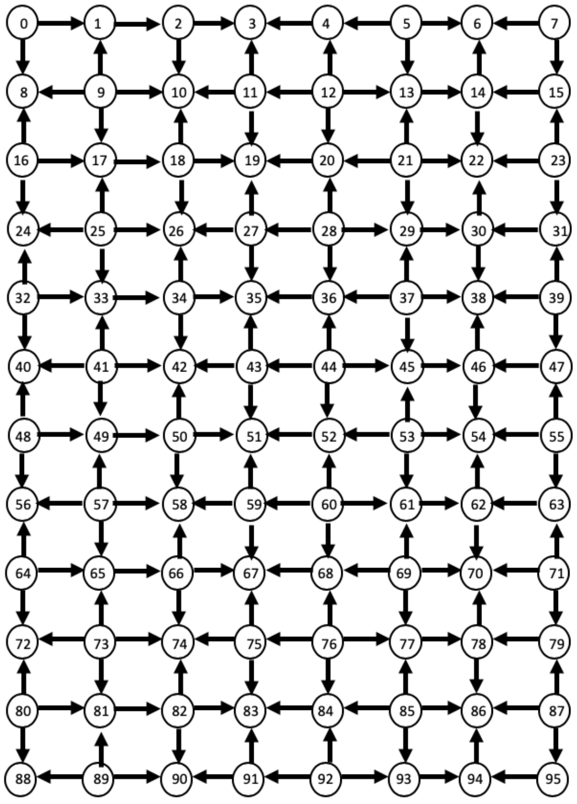


Figure 7: Proposed 96-qubit Machine used for Experimentation.

Benchmarks containing more qubits were needed for the larger

Table 3: Results of Compilation using Benchmarks from [23] Mapped to IBM Devices

Ftn.	# Qubits	T gates	Tech. Ind. (unopt. == opt.)	In form of (unoptimized mapping T-count / gates / cost) (optimized mapping T-count / gates / cost)									
				ibmqx2		ibmqx3		ibmqx4		ibmqx5		ibmq_16	
#1	3	7	7/17/22.25	***7/33/38.25***	***7/29/34.25***	***7/61/69.25***	***7/59/67.25***	***7/29/34.25***	***7/25/30.25***	***7/57/65.25***	***7/55/63.25***	***7/57/65.25***	***7/53/60.75***
#3	3	0	0/3/3.25	0/3/3.25	0/3/3.25	0/17/18.75	0/17/18.75	0/7/7.25	0/7/7.25	0/17/18.75	0/17/18.75	0/17/18.75	0/17/18.75
#01	5	15	15/51/63.75	***15/215/239.75***	***15/213/237.75***	***15/433/480.25***	***15/427/473.75***	***15/207/231.75***	***15/205/229.75***	***15/397/444.25***	***15/389/435.25***	***15/425/472.25***	***15/419/465.75***
#03	4	7	7/20/25.25	7/96/107.25	7/96/107.25	7/116/130.25	7/116/130.25	7/92/103.25	7/92/103.25	7/112/126.25	7/112/126.25	7/112/126.25	7/112/126.25
#07	5	16	16/60/75	***16/272/302***	***16/264/294***	***16/552/612***	***16/536/594.5***	***16/260/290***	***16/252/282***	***16/504/564***	***16/490/548***	***16/532/592***	***16/518/576.5***
#0f	4	0	0/3/3.25	0/21/22.75	0/21/22.75	0/35/38.25	0/35/38.25	0/21/22.75	0/21/22.75	0/35/38.25	0/35/38.25	0/31/34.25	0/31/34.25
#17	4	7	7/43/51.75	***7/215/235.75***	***7/203/223.75***	***7/289/320.25***	***7/271/300.25***	***7/203/223.75***	***7/197/217.75***	***7/273/304.25***	***7/257/286.25***	***7/273/304.25***	***7/257/286.25***
#0001	6	40	40/186/233	N/A		***40/2928/3240.5***	***40/2496/2769***	N/A		***40/2840/3152.5***	***40/2434/2706.5***	***40/2876/3188.5***	***40/2456/2725.5***
#0003	6	15	15/66/83	N/A		***15/900/996.5***	***15/824/913.5***	N/A		***15/880/976.5***	***15/792/880***	***15/904/1000.5***	***15/812/900***
#0007	6	47	47/246/304.25	N/A		***47/3116/3445.75***	***47/2728/3021.75***	N/A		***47/3096/3425.75***	***47/2652/2940.75***	***47/3144/3473.75***	***47/2688/2976.25***
#000f	5	7	7/21/27.5	***7/177/195.5***	***7/153/170***	***7/323/358***	***7/291/323***	***7/153/171.5***	***7/135/152***	***7/303/338***	***7/275/307***	***7/319/354***	***7/287/319***
#0017	6	23	23/129/159	N/A		***23/1851/2050.5***	***23/1683/1864.5***	N/A		***23/1855/2054.5***	***23/1667/1849***	***23/1899/2098.5***	***23/1745/1930***
#001f	6	43	43/194/244.5	N/A		***43/2864/3171***	***43/2474/2744***	N/A		***43/2772/3079***	***43/2434/2707***	***43/2828/3135***	***43/2454/2724***
#003f	6	16	16/73/92.25	N/A		***16/1103/1219.75***	***16/1077/1192.75***	N/A		***16/1075/1191.75***	***16/1051/1167.75***	***16/1115/1231.75***	***16/1087/1203.75***
#007f	6	40	40/189/238.5	N/A		***40/2791/3091***	***40/2593/2876***	N/A		***40/2727/3027***	***40/2503/2780***	***40/2787/3087***	***40/2531/2806.5***
#00ff	5	0	0/3/3.25	0/21/22.75	0/21/22.75	0/45/49.75	0/45/49.75	0/21/22.75	0/21/22.75	0/45/49.75	0/45/49.75	0/49/53.75	0/49/53.75
#0117	6	79	79/401/498	N/A		***79/5167/5718.5***	***79/4631/5132***	N/A		***79/5119/5670.5***	***79/4483/4973***	***79/5159/5710.5***	***79/4471/4956***
#011f	6	27	27/136/169.5	N/A		***27/1886/2086***	***27/1662/1839.5***	N/A		***27/1818/2018***	***27/1628/1807.5***	***27/1842/2042***	***27/1622/1799***
#013f	6	48	48/240/299.5	N/A		***48/3060/3389.5***	***48/2698/2993.5***	N/A		***48/3116/3445.5***	***48/2750/3046.5***	***48/3148/3477.5***	***48/2780/3078.5***
#017f	6	80	80/359/455	N/A		***80/5025/5566.5***	***80/4595/5097***	N/A		***80/4953/5494.5***	***80/4415/4904***	***80/5033/5574.5***	***80/4451/4936.5***
#033f	5	7	7/49/60.75	***7/391/428.25***	***7/299/329.25***	***7/887/978.25***	***7/747/824.25***	***7/347/384.25***	***7/277/307.75***	***7/839/930.25***	***7/691/767.25***	***7/879/970.25***	***7/727/802.75***
#0356	5	12	12/42/54.75	***12/248/274.25***	***12/224/249.25***	***12/444/491.25***	***12/388/430.25***	***12/220/246.25***	***12/208/234.25***	***12/412/459.25***	***12/372/415.25***	***12/416/463.25***	***12/362/404.25***
#0357	6	61	61/266/336.5	N/A		***61/3944/4367***	***61/3418/3788.5***	N/A		***61/3860/4283***	***61/3276/3639.5***	***61/3944/4367***	***61/3296/3653***
#035f	6	23	23/107/135.5	N/A		***23/1327/1469.5***	***23/1193/1322***	N/A		***23/1327/1469.5***	***23/1161/1288***	***23/1343/1485.5***	***23/1191/1319.5***

Table 4: Percent Decrease of [23] Benchmark Cost after Optimization

Funct.	ibmqx2	ibmqx3	ibmqx4	ibmqx5	ibmq_16
#1	10.46	2.89	11.68	3.07	6.90
#3	0.00	0.00	0.00	0.00	0.00
#01	0.83	1.35	0.86	2.03	1.38
#03	0.00	0.00	0.00	0.00	0.00
#07	2.65	2.86	2.76	2.84	2.62
#0f	0.00	0.00	0.00	0.00	0.00
#17	5.09	6.25	2.68	5.92	5.92
#0001	N/A	14.55	N/A	14.15	14.52
#0003	N/A	8.33	N/A	9.88	10.04
#0007	N/A	12.31	N/A	14.16	14.32
#000f	13.04	9.78	11.37	9.17	9.89
#0017	N/A	9.07	N/A	10.00	8.03
#001f	N/A	13.47	N/A	12.08	13.11
#003f	N/A	2.21	N/A	2.01	2.27
#007f	N/A	6.96	N/A	8.16	9.09
#00ff	0.00	0.00	0.00	0.00	0.00
#0117	N/A	10.26	N/A	12.30	13.21
#011f	N/A	11.82	N/A	10.43	11.90
#013f	N/A	11.68	N/A	11.58	11.47
#017f	N/A	8.43	N/A	10.75	11.44
#033f	23.12	15.74	19.91	17.52	17.26
#0356	9.12	12.42	4.87	9.58	12.74
#0357	N/A	13.25	N/A	15.02	16.35
#035f	N/A	10.04	N/A	12.35	11.17
Average	***5.85***	***7.65***	***4.92***	***8.04***	***8.48***

machine. Previous experimentation found that Toffoli circuits decomposed into large designs. Additionally, those that included generalized Toffoli gates with more controls had a greater final gate volume. With this in mind, benchmarks with the gates $\mathbf{T}_6$, $\mathbf{T}_7$, $\mathbf{T}_8$, $\mathbf{T}_9$, and $\mathbf{T}_{10}$ were designed for implementation on the 96-qubit architecture. Each circuit contained a cascade of four gates of each type, and they were placed on the QC in such a manner that the gates shared at least a single qubit with another. Information about the contents of the third set of benchmarks can be found in Table 7. In this table, the controls and target for each $\mathbf{T}_n$ gate in the cascade are described.

All of the Table 7 benchmarks were mapped to the 96-qubit example QC of Fig. 7. Although each circuit originally included four gates, the designs greatly increased in volume to accommodate to the Fig. 7 coupling map as well as the one- and two-qubit transmon gate library. Data concerning pre- and post-optimization T-counts, gate counts, and cost is included in Table 8. A column for technology-independent data for unoptimized and optimized mapping was also omitted from this table for the same reasons as with Table 5. The purpose of Table 8 is to not only demonstrate the generalized Toffoli decomposition capabilities of our tool, but to also demonstrate that the tool is scalable. In Table 8, optimization drastically improved the overall cost on the larger machine. On average, the large Toffoli cascade benchmarks improved in cost by 39.5%. Most of the resulting Table 7 technology-dependent circuits took under a second to generate, with the largest taking approximately 6.5 seconds. All of the output designs were verified for accuracy using the QMDD equivalence test.

6. CONCLUSION

In this work, a formally-verified, technology-dependent quantum logic synthesis tool for design automation that successfully maps and optimizes quantum circuits to the IBM Q architectures is presented. Whenever a technology-independent quantum circuit is input into the tool, a QASM file that maps the circuit to a specific physical quantum implementation is generated. To validate the quality of the generated circuits, formal verification is completed using QMDDs. Currently, the synthesis tool can decompose Toffoli gates, reroute arbitrary **CNOT** connections, and optimize quantum circuits for execution on any IBM QC or user-input layout. Algorithms are flexible so that quantum logic can be synthesized for any architecture using **NOT**, **Y**, **Z**, **H**, **S**, $\mathbf{S}^\dagger$, **T**, $\mathbf{T}^\dagger$ and **CNOT** whenever new coupling maps are provided. Due to a modular design and formally verified outputs, a generalized QC compiler and QIP circuit synthesis tool has been developed.

Table 5: Results of Compilation using Benchmarks from [24] Mapped to IBM Devices

Ftn.	# Qubits	Largest Gate	Gate Count	*In form of (unoptimized mapping T-count / gates / cost) (optimized mapping T-count / gates / cost)*									
				ibmqx2		ibmqx3		ibmqx4		ibmqx5		ibmq_16	
3_17_14	3	toffoli	6	***14/142/160.25***	***14/138/156.25***	***14/142/160.25***	***14/138/156.25***	***14/114/132.25***	***14/112/130.25***	***14/142/160.25***	***14/138/156.25***	***14/142/160.25***	***14/138/156.25***
fred6	3	toffoli	3	***21/164/188***	***21/162/186***	***21/148/172***	***21/138/162.0***	***21/164/188***	***21/154/178***	***21/148/172***	***21/138/162***	***21/148/172***	***21/138/162***
4_49_17	4	toffoli	12	***35/357/402.75***	***35/299/340.25***	***35/567/635.25***	***35/515/578.25***	***35/325/370.75***	***35/271/312.25***	***35/535/603.25***	***35/471/533.25***	***35/567/635.25***	***35/515/578.25***
4gt12-v0_88	5	T5	5	N/A		***70/2882/3211.5***	***70/1348/1515.5***	N/A		***70/2382/2657.5***	***70/1372/1542.5***	***70/2218/2472.5***	***70/1554/1741***
4gt13-v1_93	5	T4	4	***28/377/423***	***28/365/411***	***28/2665/2960***	***28/587/660***	***28/357/403***	***28/345/391***	***28/2181/2422***	***28/631/710***	***28/1685/1869***	***28/863/961.5***

Table 6: Percent Decrease of [24] Benchmark Cost after Optimization

Funct.	ibmqx2	ibmqx3	ibmqx4	ibmqx5	ibmq_16
3_17_14	2.50	2.50	1.51	2.50	2.50
fred6	1.06	5.81	5.32	5.81	5.81
4_49_17	15.52	8.97	15.78	11.60	8.97
4gt12-v0_88	N/A	52.81	N/A	41.96	29.59
4gt13-v1_93	2.84	77.70	2.98	70.69	48.56
Average	***5.48***	***29.56***	***6.40***	***26.51***	***19.08***

Table 7: 96-qubit QC Benchmark Details

Name	Gates	Controls	Target
T6_b	1: T6	q1, q2, q3, q4, q5	q25
	2: T6	q21, q22, q23, q24, q25	q45
	3: T6	q41, q42, q43, q44, q45	q65
	4: T6	q61, q62, q63, q64, q65	q85
T7_b	1: T7	q1, q2, q3, q4, q5, q6	q25
	2: T7	q21, q22, q23, q24, q25, q26	q45
	3: T7	q41, q42, q43, q44, q45, q46	q65
	4: T7	q61, q62, q63, q64, q65, q66	q85
T8_b	1: T8	q1, q2, q3, q4, q5, q6, q7	q25
	2: T8	q21, q22, q23, q24, q25, q26, q27	q45
	3: T8	q41, q42, q43, q44, q45, q46, q47	q65
	4: T8	q61, q62, q63, q64, q65, q66, q67	q85
T9_b	1: T9	q1, q2, q3, q4, q5, q6, q7, q8	q25
	2: T9	q21, q22, q23, q24, q25, q26, q27, q28	q45
	3: T9	q41, q42, q43, q44, q45, q46, q47, q48	q65
	4: T9	q61, q62, q63, q64, q65, q66, q67, q68	q85
T10_b	1: T10	q1, q2, q3, q4, q5, q6, q7, q8, q9	q25
	2: T10	q21, q22, q23, q24, q25, q26, q27, q28, q29	q45
	3: T10	q41, q42, q43, q44, q45, q46, q47, q48, q49	q65
	4: T10	q61, q62, q63, q64, q65, q66, q67, q68, q69	q85

Table 8: 96-qubit QC Benchmark Compilation Results

Name	Unoptimized (T-count / gates / cost)	Optimized (T-count / gates / cost)	Percent Cost Decrease
T6_b	336/17312/19268	336/10156/11359	41.05
T7_b	448/20112/22400	448/12234/13694	38.87
T8_b	560/21264/23728	560/13134/14746	37.85
T9_b	672/17696/19784	672/11544/13002	34.28
T10_b	784/17792/19960	784/9518/10846	45.66
Average			***39.54***

The described synthesis tool shows great promise for simplifying the quantum algorithm and circuit design process whenever real quantum computer architectures are targeted for use. Moving forward, additional decompositions for other controlled gates will be included in the tool. More optimizations to further reduce a circuit's quantum cost, especially those that aim to minimize cost by finding ideal qubit placement on a QC, will also be added. Experimentation here focused on the IBM family of QCs, but in future work, the compiler will be expanded to target other quantum technology platforms. The tool already supports the addition of coupling maps so that new devices can be targeted, and the cost function can be modified to reflect operational constraints. Since the tool has been designed in a modular manner, new technology libraries for non-IBM platforms can be added so that they may be flagged for use during synthesis. As long as each library contains an associated cost function, whether that be linear as the example in Eqn. 2 or nonlinear, existing optimizations based on minimizing the cost through the removal of gate sequences that form the identity function can be ported and implemented. In the future, we hope to target all transmon-based technology platforms as well as quantum photonic structures with formally-verified synthesis.

7. REFERENCES

[1] K. Fazel, M. A. Thornton, and J. Rice, "Esop-based toffoli gate cascade generation," in *Communications, Computers and Signal Processing, 2007. PacRim 2007. IEEE Pacific Rim Conference on*, pp. 206–209, IEEE, 2007.

[2] M. Soeken, S. Frehse, R. Wille, and R. Drechsler, "Revkit: A toolkit for reversible circuit design.," *Multiple-Valued Logic and Soft Computing*, vol. 18, no. 1, pp. 55–65, 2012.

[3] M. Saeedi, R. Wille, and R. Drechsler, "Synthesis of quantum circuits for linear nearest neighbor architectures," *Quantum Information Processing*, vol. 10, no. 3, pp. 355–377, 2011.

[4] C.-C. Lin, S. Sur-Kolay, and N. K. Jha, "Paqcs: Physical design-aware fault-tolerant quantum circuit synthesis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 7, pp. 1221–1234, 2015.

[5] N. Abdessaied, M. Amy, M. Soeken, and R. Drechsler, "Technology mapping of reversible circuits to clifford+ t quantum circuits," in *Multiple-Valued Logic (ISMVL), 2016 IEEE 46th International Symposium on*, pp. 150–155, IEEE, 2016.

[6] M. Soeken, M. Roetteler, N. Wiebe, and G. De Micheli, "Lut-based hierarchical reversible logic synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[7] D. M. Miller and M. A. Thornton, "Qmdd: A decision diagram structure for reversible and quantum circuits," in *Multiple-Valued Logic, 2006. ISMVL 2006. 36th International Symposium on*, pp. 30–30, IEEE, 2006.

[8] M. A. Nielsen and I. Chuang, "Quantum computation and quantum information," 2010.

[9] J. Koch, M. Y. Terri, J. Gambetta, A. A. Houck, D. Schuster, J. Majer, A. Blais, M. H. Devoret, S. M. Girvin, and R. J. Schoelkopf, "Charge-insensitive qubit design derived from the cooper pair box," *Physical Review A*, vol. 76, no. 4, p. 042319, 2007.

[10] M. Amy, D. Maslov, and M. Mosca, "Polynomial-time t-depth

optimization of clifford+ t circuits via matroid partitioning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 10, pp. 1476–1489, 2014.

[11] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," *Physical review A*, vol. 52, no. 5, p. 3457, 1995.

[12] J. M. Chow, J. M. Gambetta, E. Magesan, D. W. Abraham, A. W. Cross, B. Johnson, N. A. Masluk, C. A. Ryan, J. A. Smolin, S. J. Srinivasan, *et al.*, "Implementing a strand of a scalable fault-tolerant quantum computing fabric," *Nature communications*, vol. 5, p. 4015, 2014.

[13] P. Niemann, R. Wille, D. M. Miller, M. A. Thornton, and R. Drechsler, "Qmdds: Efficient quantum function representation and manipulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 1, pp. 86–99, 2016.

[14] "IBM Q." http://research.ibm.com/quantum/.

[15] M. Takita, A. W. Cross, A. Córcoles, J. M. Chow, and J. M. Gambetta, "Experimental demonstration of fault-tolerant state preparation with superconducting qubits," *Physical Review Letters*, vol. 119, no. 18, p. 180501, 2017.

[16] A. D. Córcoles, E. Magesan, S. J. Srinivasan, A. W. Cross, M. Steffen, J. M. Gambetta, and J. M. Chow, "Demonstration of a quantum error detection code using a square lattice of four superconducting qubits," *Nature Communications*, vol. 6, p. 6979, 2015.

[17] IBM Q team, "IBM Q 5 Yorktown backend specification V1.1.0." https://ibm.biz/qiskit-yorktown, 2018. Accessed: Dec. 2018.

[18] IBM Q team, "IBM Q 5 Tenerife backend specification V1.3.0." https://ibm.biz/qiskit-tenerife, 2018. Accessed: Dec. 2018.

[19] IBM Q team, "IBM Q 16 Rueschlikon backend specification V1.1.0." https://ibm.biz/qiskit-rueschlikon, 2018. Accessed: Dec. 2018.

[20] IBM Q team, "IBM Q 16 ibmqx3 backend specification V1.0.0." https://ibm.biz/qiskit-rueschlikon, 2018. Accessed: Dec. 2018.

[21] IBM Q team, "IBM Q 16 Melbourne backend specification V1.1.0." https://ibm.biz/qiskit-melbourne, 2018. Accessed: Dec. 2018.

[22] https://github.com/QISKit.

[23] "Reversible logic synthesis and quantum computing benchmarks." http://quantumlib.stationq.com/, 2017.

[24] http://www.revlib.org/.