Reversible Quantum Circuit Synthesis

Saumya Shah
Electrical Engineering
IIT Bombay
22B1238

Harshil Singla
Electrical Engineering
IIT Bombay
22B1260

Dattaraj Salunkhe
Electrical Engineering
IIT Bombay
22B1296

Neel Rambhia
Electrical Engineering
IIT Bombay
22B1298

***Abstract*—In the field of quantum computing, one of the fundamental tasks is to synthesize quantum circuits that perform specific logical operations. Quantum circuit synthesis refers to the process of designing a quantum circuit that implements a desired unitary transformation. This paper explores an approach for synthesizing quantum circuits directly from truth tables, where a truth table serves as the input, and the corresponding quantum circuit is generated as the output.**

I. Introduction

Quantum computing represents a paradigm shift in computational theory, promising to revolutionize fields such as cryptography, optimization, and machine learning. At the heart of quantum computing lies the concept of quantum circuits, which are responsible for executing quantum algorithms. Unlike classical circuits, which rely on classical bits, quantum circuits manipulate quantum bits, or qubits, that can exist in superpositions of states.

The synthesis of quantum circuits from classical logic specifications, such as truth tables, is a critical area of study. A truth table provides a complete specification of a Boolean function, mapping every possible input combination to its corresponding output. For many quantum algorithms, the goal is to design quantum circuits that implement a given function as efficiently as possible, while minimizing circuit depth, gate count, and the potential for errors due to noise.

This paper focuses on a method of synthesizing quantum circuits from truth tables. By interpreting the truth table as a classical Boolean function, we explore techniques for transforming this function into a quantum circuit using basic quantum gates. We would like to mention [1], [2], IBM Quantum and Prof. Virendra Singh from whose teachings we took our inspiration for this implementation framework.

Our code and examples can be accessed here: https://github.com/Ingenio17/Quantum-Circuit-Synthesis

II. Methodology

It is essential to make the given Truth Table reversible for Quantum Implementation. The same is discussed in the following subsection.

A. Making the Truth Table Reversible

The process of converting a given truth table to its **Reed-Muller form** involves the following steps:

Step 1: Convert the truth table to its reversible form

1) **Equalize the number of input and output bits:**
   - If the number of input bits (n) and output bits (m) are unequal, add the required number of bits to the side in deficit to make them equal.
   - For example, if $m > n$, add $m - n$ input bits initialized to zeros; if $n > m$, add $n - m$ output bits. The added output bits follow a specific pattern (e.g., 00, 01, 10, 11 for 2 bits) to fill the truth table.
2) **Ensure bijection:**
   - Identify conflicts in the output combinations (i.e., identical output combinations for different input rows).
   - For each conflict, determine the number of additional bits needed to ensure uniqueness. Specifically, add k bits to the output such that $k = \lceil \log_2(\max(\text{conflicts})) \rceil$.
   - Add the same k bits to the input, initializing them to zeros.
3) **Complete the truth table:**
   - Since bits were added, expand the truth table to ensure it remains a bijection. Assign unique output combinations for any new input rows added.

At the end of this step, the truth table represents a reversible function with an equal number of input and output bits, ensuring a one-to-one mapping.

Step 2: Write the Reed-Muller form

1) For each output bit in the reversible truth table:
   - Identify all input combinations where the output bit is 1.
   - Express the output as the XOR of these input combinations.
   - For example, if an output bit is 1 for input combinations $A_1, A_2, \ldots, A_k$, its Reed-Muller form is:

$$\text{Output bit} = A_1 \oplus A_2 \oplus \ldots \oplus A_k$$

By following these steps, the truth table is systematically converted into its Reed-Muller representation.

B. Building Quantum Circuit

We utilize the Algorithm provided in [1] for circuit creation. The core of the algorithm revolves around the PPRM (Positive

Polarity Reed-Muller) expansion of the function to be synthesized. This representation is advantageous because it expresses Boolean functions in a canonical form, facilitating systematic manipulation. The algorithm constructs the reversible circuit by iteratively transforming the PPRM expansion into a sequence of reversible gates.

The idea behind this algorithm is the **cascading of substitutions for the variables** done one at a time. We use trivial gates (Pauli X and Multi Control Toffoli Gates) to make a certain substitution to one of the variables in accordance with the target value we need on that qubit. Once this substitution is made to one of the bits (variables), the target values of each of the variables is calculated by using the updated value. This process is continually repeated until convergence.

For optimizing the above process and to minimize the number of quantum gates required, we utilize priority queues with the priorities chosen from [1]. This leads to faster convergence. Priority is an important feature here since it gives balance between the two extremes of **Breadth First Search** and **Depth First Search** which are the two extreme cases for finding the solution (See Appendix). A proper balance helps find us the required solution optimally.

III. Performance Analysis

The complexity of this framework increases with increase in the number of qubits or the effective number of variables in the Reversible Truth Table. Additionally, an optimal conversion of a Truth Table into its Reversible form is necessary since not every time does the algorithm converge. Sometimes the priority heap is emptied without finding a solution. A remedy for the same would be to try finding alternative reversible functions to represent the same truth table or to try reversing the output or input qubits.

While testing our algorithm on a few examples, we saw a significant increase in computation time and resources when the number of qubits increased from 2 to 3 to 4. Thus, development of better Heuristic-based approaches is essential. The possibility of modifying the expression for calculating Priority can also be explored.

IV. Example Circuits

Shown below are some of the Quantum Circuits built from our code.

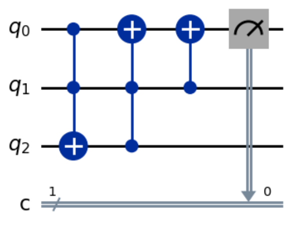

Fig. 1. OR Gate

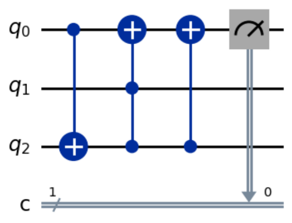

Fig. 2. AND Gate

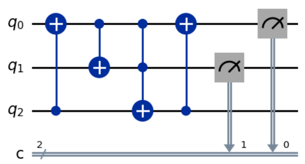

Fig. 3. Half Adder

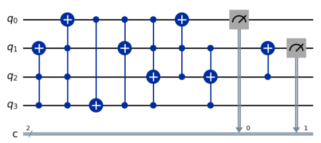

Fig. 4. Full Adder

V. Conclusion and Future Prospects

The synthesis of reversible logic circuits represents a pivotal advancement in the pursuit of energy-efficient computation, particularly in domains like quantum computing, cryptography, and low-power systems. By leveraging the algorithmic framework outlined in this work, we have demonstrated a systematic and scalable approach to reversible circuit synthesis. The use of PPRM-based transformations, coupled with efficient gate synthesis techniques, ensures minimal gate count, reduced circuit depth, and optimal ancillary bit usage.

As quantum technologies and energy-efficient computing gain momentum, the role of efficient reversible circuit synthesis will become increasingly vital. This research offers a robust framework to support these advancements, paving the way for innovative applications and transformative breakthroughs in computation.

References

[1] Gupta et. al., “An Algorithm for Synthesis of Reversible Logic Circuits,” IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems, Vol. 25, No. 11, November 2006.

[2] Miller et. al., “Automated Method for Building CNOT Based Quantum Circuits for Boolean Functions,” Spetmeber 2018.

Appendix

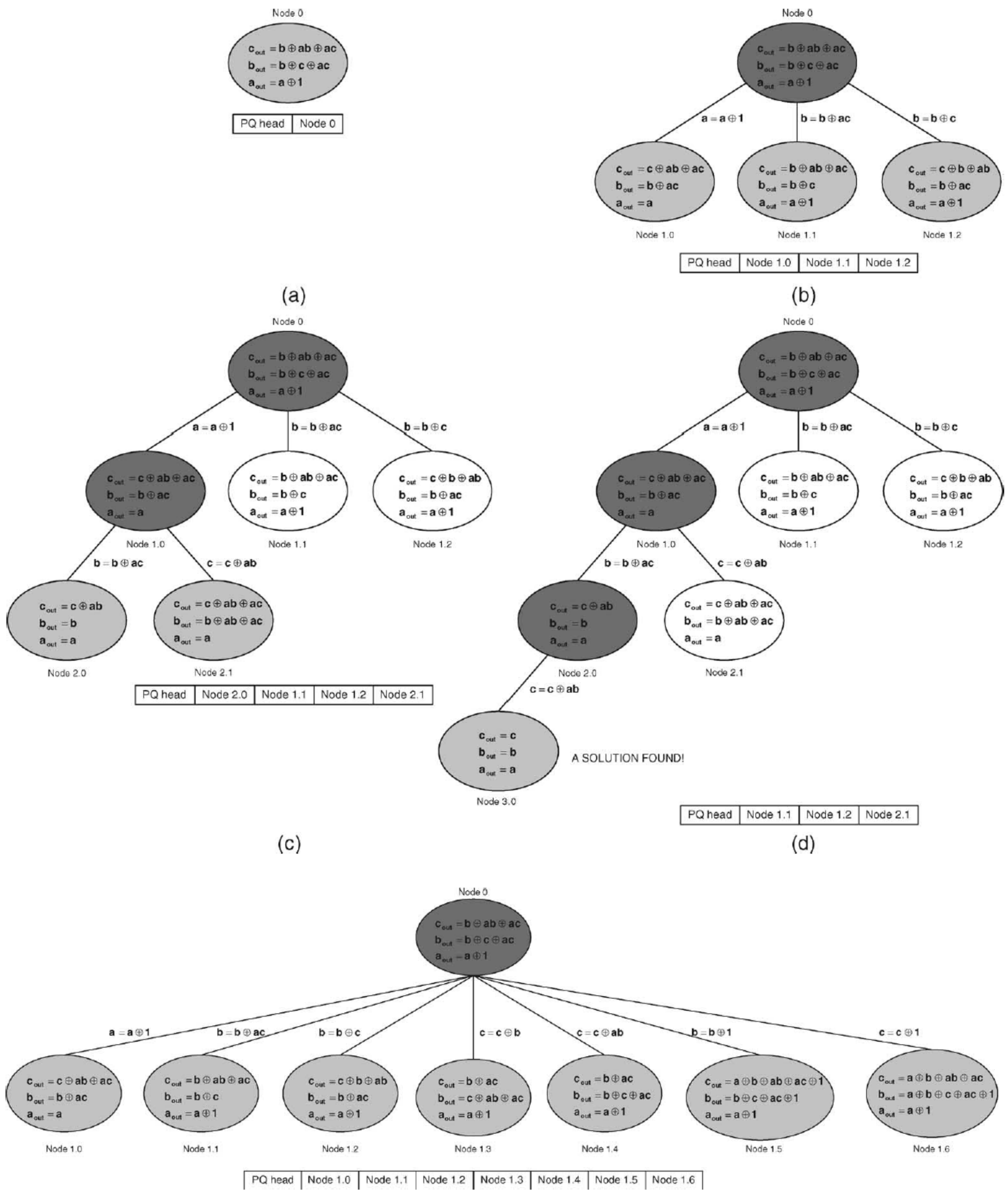
Node 0
$c_{out} = b \oplus ab \oplus ac$
$b_{out} = b \oplus c \oplus ac$
$a_{out} = a \oplus 1$
PQ head | Node 0
(a)
$a = a \oplus 1$
$b = b \oplus ac$
$b = b \oplus c$
Node 1.0
$c_{out} = c \oplus ab \oplus ac$
$b_{out} = b \oplus ac$
$a_{out} = a$
Node 1.1
$c_{out} = b \oplus ab \oplus ac$
$b_{out} = b \oplus c$
$a_{out} = a \oplus 1$
Node 1.2
$c_{out} = c \oplus b \oplus ab$
$b_{out} = b \oplus ac$
$a_{out} = a \oplus 1$
PQ head | Node 1.0 | Node 1.1 | Node 1.2
(b)
$c = c \oplus ab$
Node 2.0
$c_{out} = c \oplus ab$
$b_{out} = b$
$a_{out} = a$
Node 2.1
$c_{out} = c \oplus ab \oplus ac$
$b_{out} = b \oplus ab \oplus ac$
$a_{out} = a$
PQ head | Node 2.0 | Node 1.1 | Node 1.2 | Node 2.1
(c)
Node 3.0
$c_{out} = c$
$b_{out} = b$
$a_{out} = a$
A SOLUTION FOUND!
PQ head | Node 1.1 | Node 1.2 | Node 2.1
(d)
$c = c \oplus b$
$b = b \oplus 1$
$c = c \oplus 1$
Node 1.3
$c_{out} = b \oplus ac$
$b_{out} = c \oplus ab \oplus ac$
$a_{out} = a \oplus 1$
Node 1.4
$c_{out} = b \oplus ac$
$b_{out} = b \oplus c \oplus ac$
$a_{out} = a \oplus 1$
Node 1.5
$c_{out} = a \oplus b \oplus ab \oplus ac \oplus 1$
$b_{out} = b \oplus c \oplus ac \oplus 1$
$a_{out} = a \oplus 1$
Node 1.6
$c_{out} = a \oplus b \oplus ab \oplus ac$
$b_{out} = a \oplus b \oplus c \oplus ac \oplus 1$
$a_{out} = a \oplus 1$
PQ head | Node 1.0 | Node 1.1 | Node 1.2 | Node 1.3 | Node 1.4 | Node 1.5 | Node 1.6

Fig. 5. Depth First Search (Top), Breadth First Search (Bottom). Source: Ref.[1]