Improving Qiskit strategies for circuit synthesis using the Reed-Muller spectrum

Raphael B. F. Lima[1], **Luis Antonio B. Kowada**[1], **Fábio G. dos Santos**[1]

[1] Instituto de Computação – Universidade Federal Fluminense (UFF)
Niterói – RJ – Brazil

`rbernardino@pm.me, luis@ic.uff.br, fabiogsrj.2007@gmail.com`

Abstract. *Reversible computing and quantum computing have gained increasing attention due to their potential to overcome some of the fundamental limitations of classical computing, particularly regarding energy efficiency and computational power. In this paper, we compare the synthesis process using the well-known Qiskit framework developed by IBM and an approach based on Reed-Muller spectra. We show that some approaches are efficient for small values and others are better for the general use case. The synthesis results are presented using four strategies: unitary matrices, linear function synthesis, permutation synthesis, and Reed-Muller spectra. Our analysis shows that, on average, our proposal surpasses the Qiskit synthesis algorithms when considering the gate count metric. Additionally, a post-synthesis evaluation is performed using our proposed method, which employs a Reed-Muller–based representation to assess its effectiveness.*

1. Introduction

The emerging fields of reversible computing and quantum computing have gained increasing attention due to their potential to overcome some of the fundamental limitations of classical computing, particularly regarding energy efficiency and computational power. Classical digital computing, based on Boolean logic, typically involves irreversible operations such as AND, OR, and NAND gates. These gates destroy information by compressing many possible input states into a single output, leading to energy dissipation. As a result, traditional computing approaches face a theoretical lower bound on energy efficiency, as described by Landauer's principle in his seminal work [Landauer 1961], which states that any logically irreversible operation that erases one bit of information must dissipate at least $k_B T \ln 2$ joules of heat, where k_B is Boltzmann's constant, and T is the temperature in Kelvin.

To mitigate this, reversible computing proposes computation without information loss, which means that all operations are invertible, and no bits are erased. This eliminates the fundamental source of energy dissipation associated with irreversible processes. In reversible circuits, each output uniquely determines its inputs, allowing the computation to be performed backward and preserving the original information. Classical reversible gates like the Toffoli [Toffoli 1980] and Fredkin [Fredkin and Toffoli 1982] gates demonstrate that universal computation can be achieved reversibly, without erasing bits.

Simultaneously, quantum computing, grounded in the principles of quantum mechanics, offers a computational model that is inherently reversible. In quantum systems, the evolution of the quantum state is described by unitary transformations, meaning that the entire history of the system can be inferred from the present state. This implies that quantum gates, such as the Hadamard, Pauli-X, and controlled-NOT (CNOT) gates, are intrinsically reversible since unitary operations conserve information by the laws of quantum mechanics [Nielsen and Chuang 2000]. As a result, quantum computers naturally align with the objectives of reversible computing. This also means that any quantum computation, at least theoretically, can be decomposed into reversible steps.

The overlap between these fields becomes more apparent in circuit design. While reversible computing primarily focuses on reducing energy dissipation in classical systems by preserving information, quantum computing employs reversible logic as a natural aspect of its operation. Moreover, many quantum algorithms are expressed using networks of reversible gates, highlighting the connection between reversible computing and quantum circuit design. For instance, Shor's factoring algorithm uses reversible logic within its quantum components to achieve exponential speedup over classical factoring methods [Shor 1999]. The Grover's algorithm [Grover 1996] also benefits from improvements in reversible circuits.

Moreover, reversible circuits, such as those made with Toffoli gates, can be used in the classical components of quantum computers to ensure that no information is lost during ancillary classical computations. Thus, research in both areas not only advances the quest for energy-efficient computing but also lays the foundation for the development of quantum technologies, particularly in quantum error correction and fault-tolerant quantum circuits [Gottesman 1998, Knill and Laflamme 1997]. In both paradigms, circuit design is a critical challenge. Quantum circuits require fault-tolerant gate operations due to the fragile nature of quantum states, whereas reversible circuits focus on minimizing the number of ancillary bits (known as "garbage" bits) to ensure the computation is done efficiently and the original information remains preserved [Maslov and Dueck 2003].

2. Background

In the early 1980s, Benioff demonstrated through his work [Benioff 1980, Benioff 1982a, Benioff 1982b] that classical computation could be modeled using reversible circuits. This was very important for the field of quantum computing because quantum computation, which relies on quantum mechanics, evolves according to unitary transformations and is therefore reversible. Thus, Benioff's research helped establish a theoretical foundation for understanding how computation could align with the reversible nature of quantum-mechanical processes.

Building upon Benioff's foundational work, quantum computing has seen significant advancements, leading to the development of sophisticated circuits for implementing novel algorithms. These circuits are constructed from a universal set of quantum gates, among which the controlled-NOT (CNOT) and Toffoli (CCNOT) gates are fundamental. The CNOT gate flips the target qubit if and only if the control qubit is in the state $|1\rangle$, while the Toffoli gate flips the target qubit only when both control qubits are in the state $|1\rangle$. To illustrate the fundamental role of CNOT and Toffoli gates in quantum circuit design, we can examine their application within Shor's algorithm.

Shor's algorithm [Shor 1999] is a quantum algorithm designed to efficiently factor composite integers of the form $N = p \cdot q$, where p and q are prime numbers. This algorithm marked a pivotal advancement in quantum computing, showcasing an exponential speedup over classical factorization methods. Peter Shor's ingenious insight lay in reformulating the factorization problem as an order-finding problem within a finite abelian group. Specifically, the algorithm examines the sequence generated by successive powers of a randomly k chosen number, modulo N: $[x_1 \mod N, x_2 \mod N, \ldots, x_i \mod N]$, where $i < k$. When x_i is coprime [1] to N, this sequence exhibits a periodic pattern. By leveraging the quantum Fourier transform (QFT), Shor's algorithm efficiently determines this period. Using the period and some classical postprocessing, the factors p and q of N can be extracted. The period found is related to the order of x modulo N, which is a divisor of the least common multiple of $(p-1)$ and $(q-1)$, not necessarily equal to it.

The quantum circuit implementation for Shor's algorithm employs Hadamard gates to initialize the input register into a uniform superposition; subsequently, a modular exponentiation circuit generates a sequence that encodes the periodic behavior (sometimes referred to as an oracle), and finally, the quantum Fourier transform (QFT) is applied to extract the period, which directly produces the factors of the input integer.

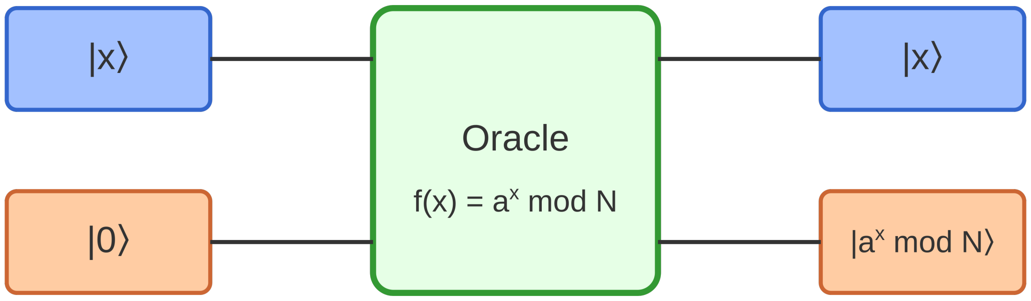


Figure 1. Quantum oracle for the modular exponentiation in the Shor's algorithm.

In Shor's algorithm, the computationally intensive component lies in the order-finding function, specifically the modular exponentiation ($x^a \mod N$) performed by the oracle within it (see Figure 1). This modular exponentiation can be decomposed into a series of modular multiplications, which can be further broken down into modular additions. The exponentiation can be efficiently implemented through a technique called repeated squaring, which significantly reduces the number of multiplication operations required for modular exponentiation.

At the quantum implementation level, this method follows the decomposition approach mentioned above. The key to achieving these computational reductions is the use of control qubits, which is precisely where the Toffoli gate plays an important role. As a reversible three-qubit gate that performs controlled-controlled-NOT operations, the Toffoli gate enables the conditional logic necessary for implementing the modular arithmetic operations while maintaining quantum coherence throughout the computation.

3. Our proposal

Our proposal is to compare the synthesis process using the well-known Qiskit framework developed by IBM [IBM 2017] and an approach based on Reed-Muller spectra [Maslov et al. 2007, Zakablukov 2016]. The Qiskit framework provides implementations of some synthesis algorithms, such as Linear Function Synthesis [Patel et al. 2008],

[1] A number x is coprime to k if their greatest common divisor (GCD) is equal to 1. In our case, x is coprime with N if it is not divisible by p or q.

Permutation Synthesis [Kutin et al. 2007], and the Unitary Transformation Synthesis. The latter synthesis is constructed using the unitary matrix as input and creating an equivalent operator. We then use the findings of [Bernardino and Kowada 2025] to elaborate on our post-synthesis results and test if the synthesis algorithms can be further optimized.

The aforementioned Reed-Muller approach uses the positive-polarity Reed-Muller (PPRM) form that is a canonical representation of Boolean functions and suitable for reversible circuit synthesis [Bandyopadhyay and Rahaman 2014]. In PPRM, a Boolean function is expressed as a sum of products of variables, with each term containing only positive literals (no negations). The benefit of this form lies in its minimality and simplicity, which directly translates to more efficient reversible circuits, both in terms of gate count and depth. To evaluate each approach, we perform a post-synthesis process in each of the generated reversible circuits. The post-synthesis proposed in [Bernardino and Kowada 2025] consists of three techniques:

- Using the rules defined in [Dalcumune et al. 2021] that cancel consecutive duplicated gates, elimination of NOT gates, and merge gates that are similar.
- Replacement of a set of gates for the optimal equivalent circuit if the Hamming distance h within the set is less than 4 ($h < 4$).
- Performing a post-synthesis using Reed-Muller spectra, in the form of Mixed Polarity Reed-Muller (MPRM), when the Hamming distance h is equal or greater than 4 ($h \geq 4$).

In other words, we use the algorithm based on Reed-Muller spectra, a set of rules, and a set of exact circuit templates for $n < 4$ bits. The templates were generated using a simple brute-force technique to test every $2^n!$ possible combination, considering only Generalized Toffoli gates. In total, we need to test $2! + 4! + 8! = 40,346$ cases.

The experiment was realized using an Oracle Cloud machine with the following specifications. The Reed-Muller algorithm was implemented using the programming language Python 3.12, and Qiskit is also written in the same language.

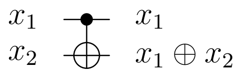
- **OS:** Linux Ubuntu 22.04.4 LTS (kernel: 6.5.0-1027-oracle).
- **CPU:** ARM Neoverse-N1 4x OCPU.
- **RAM:** 24 GB.

In order to evaluate the synthesis results, we generate a fixed number $p = 10$ of random permutations that use n bits. The permutations are generated using the pseudo-random function with the $seed = 10$. Then, we use the strategies described above and synthesize a network of reversible gates. We also perform a post-synthesis afterwards using the synthesized circuits aforementioned.

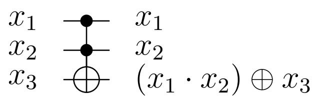
For the sake of clarity, we briefly explain the Generalized Toffoli library in Section 3.1 and the Reed-Muller MPRM algorithm in Section 3.2.

3.1. Generalized Toffoli (GT) library

The Generalized Toffoli library includes NOT gates with multiple controls, which can be either positive or negative. For example, the Controlled NOT (CNOT) gate, which has one control, is shown in Figure 2a. The Toffoli gate, shown in Figure 2b, has two controls. Note that, the Toffoli is also known as C^2NOT gate.



(a) CNOT gate.



(b) Toffoli gate.

Figure 2. Representation of CNOT gate and the Toffoli gate.

The gates can be generalized by adding more m controls to the NOT gate. The Generalized Toffoli (GT) and Multiple Controlled Toffoli (MCT) libraries are similar, although the MCT library does not have negative controls. The representation of a negative control in MCT can be achieved using a NOT gate before the positive control.

For instance, using the generalization example shown in Figure 3, with m controls. The control x_1 is negative and the other controls are positive. In other words, the value of x_{m+1} is inverted when $x_1 = 0$ and $x_i = 1$, for $i \in \{2, ..., m\}$. A generalized Toffoli gate $C^m NOT$ uses $2m - 3$ Toffoli gates for its implementation in a real system, either reversible or quantum [Barenco et al. 1995].

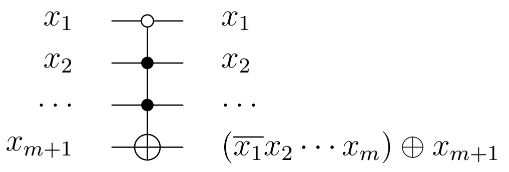


Figure 3. Generalized Toffoli representation using multiple controls, which can be positive or negative. The negative control is shown in the first line of the circuit, where all the other lines have a positive control.

3.2. Mixed Polarity Reed-Muller (MPRM)

The positive polarity Reed-Muller (PPRM) form only uses positive controls, thereby missing many opportunities. In [Bernardino and Kowada 2025], the MPRM is used in the post-synthesis process, yielding excellent results. From this starting point, we use the proposed algorithm to perform the synthesis, with minor adjustments. The polynomials generated through the Reed-Muller expansions in the form of mixed polarity Reed-Muller (MPRM), or Kronecker form, result in smaller circuits, as the polynomial terms start with smaller values, which are then translated into Toffoli gates with fewer controls.

A study conducted by [Abbe et al. 2020] shows the various parameters for the Reed-Muller code, as well as their relationships and impacts for each approach. It is important to note that [Kaufman et al. 2012] demonstrates techniques for constructing the codes, and thus, [Abbe et al. 2015] could manage to improve and prove that such construction has several benefits.

The synthesis process starts with a Boolean function $f(x_1, x_2, ..., x_m)$, which can be written as a sum of EXOR products, also known as ESOP, where $a_i \in 0, 1$:

$$f(x_1, x_2, \ldots, x_m) = a_0 \oplus a_1 x_1 \oplus a_2 x_2 \oplus \cdots \oplus a_m x_1 x_2 \ldots x_m$$

Each term of the polynomial corresponds to an interaction related to the input variables. These terms can, in turn, be mapped to reversible gates, where each gate implements a specific term of the polynomial. Additionally, each term can be interpreted, either partially or entirely, using the idea of complement $x_i = 1 \oplus x_i$.

The MPRM approach, discussed in [Porwik 2002], demonstrates the possibilities of forms in the Reed-Muller spectrum and highlights that using positive polarity Reed-Muller (PPRM) reduces the number of generated expressions. Other methods, such as fixed polarity Reed-Muller (FPRM), are less effective for circuits with many variables [Falkowski and Chang 1995]. Notably, in FPRM, each variable appears with the same polarity across all terms, whereas in MPRM, a variable can appear with both positive and negative polarities. Based on this, [Porwik 2002] suggests using Walsh coefficients to determine the best expression from MPRM expansions.

By definition, we say that the spectrum of a Boolean function is obtained through the multiplication between the output vector of this function and a transformation matrix. The result is the spectral vector, which contains the spectral coefficients of the function. In our case, we will use a transformation matrix K and a vector V that represents the outputs of the Boolean function. The transformation matrix K is defined recursively using Equation 1. That is, each Walsh-Hadamard matrix K_w is generated from a previous matrix K_{w-1}. In a simpler way, we can apply the Kronecker product to matrix K_1 a total of w times in order to obtain K_w. Note that the size of the matrix, as well as the number of operations, is related to the number of variables involved in the function.

$$\begin{aligned} K_w(0) = [1] \qquad & K_w(1) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \\ K_w(n) = & \begin{bmatrix} K_w(n-1) & K_w(n-1) \\ K_w(n-1) & -K_w(n-1) \end{bmatrix} \end{aligned} \tag{1}$$

In this way, we can define our algorithm and give an example using a generic function f that has 3 variables: x_1, x_2, and x_3. Suppose the output generated by this function is $V = [0, 1, 0, 1, 1, 0, 1, 1, 0]$. Based on this information, we will calculate the transformation matrix $K(3)$ and adapt the output V using the following formula: $\{0, 1\} \to \{1, -1\}$.

The next step is to calculate the spectral vector $S = V \cdot K$. Analyzing the spectrum S, we will choose the position that has the highest correlation with the output, which is indicated by the largest value in magnitude. The correlation found using the line of the spectrum helps us determine which variables are involved. If the value of S is negative, we have the case where the correlation is inverse, and if the value is positive, we have a direct correlation. In other words, when the correlation is inverse, we have the complement of all the involved variables, and when there is a direct correlation, we have the variables without complement. For example, in Table 1, only the last row has $S < 0$, which implies an inverse correlation involving the variables x_1, x_2, and x_3.

In Table 1, we can see that the largest value in magnitude is equal to 4. In our approach, we use the first row with the largest value, although any of the values could be used. Therefore, the chosen correlation is x_3. We then use this correlation to simplify the original function, as follows: $d(x) = g(x_3) \oplus f(x_1, x_2, x_3)$. This process repeats recursively using the new function $d(x) = f_i(x_1, x_2, \ldots, x_m)$

Table 1. Example, using a function f, of how to obtain the expressions in the MPRM form, the spectrum S, intermediate functions, and its correlations.

$x_1x_2x_3$	$f(x_1,x_2,x_3)$	V	S	correlation	$g(x_3)$	$d(x)$
000	0	1	0	0	0	0
001	1	-1	4	x_3	1	0
010	0	1	0	x_2	0	0
011	1	-1	4	x_2x_3	1	0
100	0	1	0	x_1	0	0
101	1	-1	4	x_1x_3	1	0
110	1	-1	0	x_1x_2	0	1
111	0	1	-4	$\overline{x_1x_2x_3}$	1	1

until it converges entirely to zero or to one. Each step of the recursion returns a set of variables that must be combined in the ESOP form. In other words, we will have: $f(x_1,x_2,x_3) = f_1(x_1,x_2,x_3) \oplus f_2(x_1,x_2,x_3) \oplus \cdots \oplus f_k(x_1,x_2,x_3)$, where k represents the number of reductions needed to represent the original function.

After another round of the process described above, we will obtain the following expression: $f(x_1,x_2,x_3) = x_3 \oplus x_1x_2$. From Table 1, it is easy to verify that the next step generates x_1x_2, since $d(x)$ is equal to 1 only when $x_1 = x_2 = 1$.

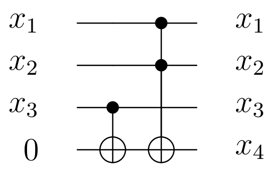


Figure 4. The resulting circuit from the synthesis of the function f using an ancillary bit x_4.

Note that, due to its versatility, this approach is capable of indirectly performing a post-synthesis process. However, keep in mind that optimizations are not performed on the result. In our approach, we use the expression found to generate *ancillas* [2]. In the example above, presented in Table 1, we would place the terms with the target in a new line x_4 — as shown in Figure 4. Each function adds one ancilla in the generated circuit.

4. Synthesis results

Using the Qiskit framework we performed the synthesis process for $n = 2, \cdots, 8$ bits. The synthesis results are shown in Table 2 using the gate count metric. This metric is straightforward and represents the number of gates in the network of reversible gates. The first column is the size of bits to represent a Boolean function. The other columns are, respectively: the number of gates using the unitary synthesis, the linear synthesis, the

[2] By definition, ancilla is an auxiliary bit that exists in the circuit to assist with specific operations, such as simplifying the problem, enabling certain transformations, or facilitating the manipulation of the system without affecting the final outcome.

permutation synthesis, and the Reed-Muller synthesis. The last row of the table shows the average result for a given synthesis algorithm.

Table 2. The synthesis results (in terms of gate count) are presented using four strategies: unitary matrices, linear function synthesis, permutation synthesis, and Reed-Muller spectra. The first column represents the number of input bits. Columns 2-4 display the gate counts using the Qiskit framework, while the last column shows the gate count for our approach.

	Qiskit framework			**ours**
n	Unitary	Linear	Permutation	**Reed-Muller**
2	1.80	5.40	9.60	**3.20**
3	19.00	16.20	42.00	**8.70**
4	99.70	38.10	190.50	**23.00**
5	444.00	82.20	746.40	**57.80**
6	1,868.00	177.00	3,036.60	**144.00**
7	7,660.00	366.30	11,987.10	**347.50**
8	31,020.00	750.30	48,428.10	**785.60**
avg.	5,873.21	205.07	9,205.76	**195.69**

Notice that the unitary approach for synthesis uses the least amount of gates for small values of n, but increases faster than the other approaches. This happens because the unitary matrices and the algorithm do not consider the use of additional ancilla bits.

The linear synthesis approach uses $2^n - 1$ bits and achieves a better gate count. In contrast, the permutation synthesis employs SWAP gates, and when translated into Toffoli gates, the gate count increases on average by a factor of 3, as one SWAP gate is equivalent to three Toffoli gates. The permutation approach requires approximately 2^n bits. The Reed-Muller strategy is preferred for $n > 2$, as it uses $2n$ bits and yields the best results. On average, its performance is better, with a more reasonable growth in gate count and significantly fewer bits compared to the other methods.

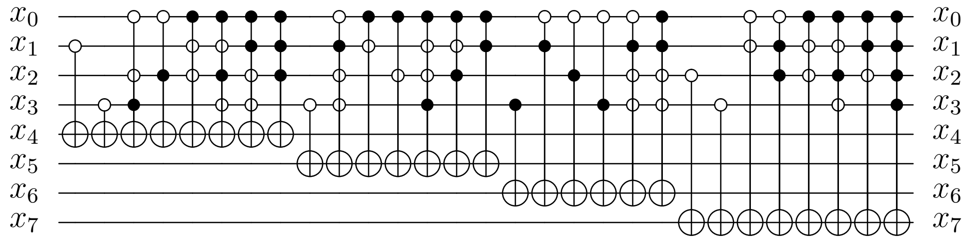
In Figure 5 is noticeable that the Reed-Muller strategy surpasses the current techniques available in the Qiskit framework.

5. Post-synthesis results

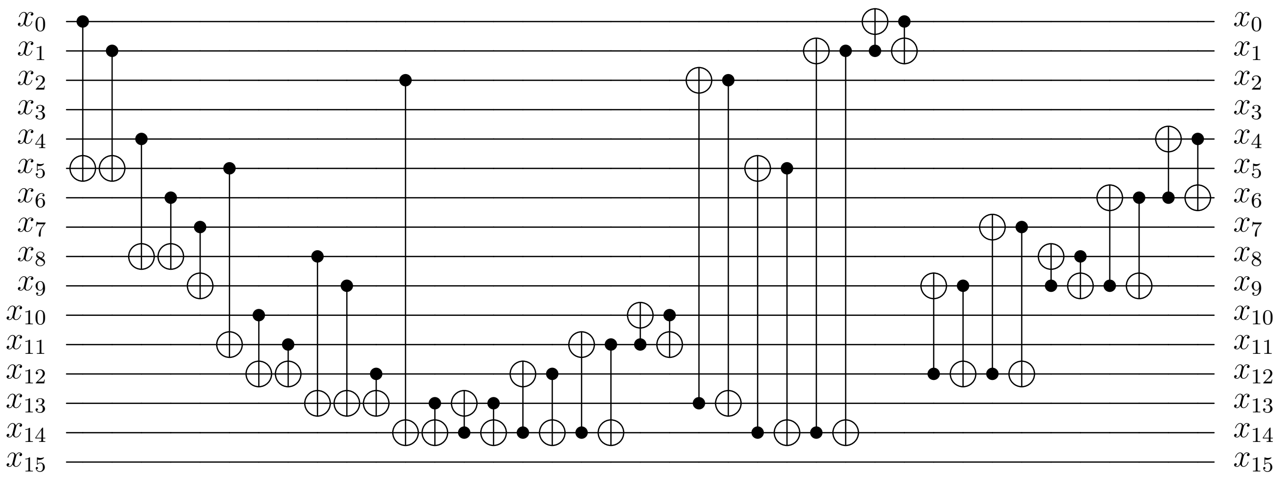
The second part of our proposal is to evaluate which synthesis process gives the best result, considering our post-synthesis approaches. The results after applying the post-synthesis strategies are shown in Table 3. The first column represents the strategy used to evaluate the synthesis, the columns 2-6 show the number of bits used in the input function, and the last column shows the average gate count for the strategy. For columns 2-6, the average number of gates is also shown in the last row.

The strategies used consist of combining each strategy described in Section 3. For the sake of simplicity and readability, we use the following abbreviations:

- OT: Optimal template matching strategy. Outputs the best set of gates for any circuit with a Hamming distance less than 4.
- RM: Reed-Muller spectra based algorithm. Through the PPRM form, finds the minimal spectra and outputs the near-minimal network of reversible circuits.



(a) The resulting circuit obtained using the Reed-Muller synthesis approach.



(b) The resulting circuit obtained using the linear synthesis approach [Patel et al. 2008].

Figure 5. Comparison between (a) our approach using Reed-Muller and (b) the optimal technique in Qiskit when the unitary approach is not applicable. Note that while the unitary method is preferred, it is not always feasible due to the requirement for inputs to be represented as a unitary matrix.

- RU: Rules for canceling and merging gates. Limited to some cases, but is an all around strategy that does not have limitation in terms of bits or number of gates.

For example, when reading in Table 3 RU+RM+OT that means the following strategies were applied, in that order: rules, Reed-Muller, and template matching. Notice that, in most cases, the strategy order matters. The best average result for each strategy is marked in bold.

Some strategies could not be finished for using much memory. That was the case for Reed-Muller. To calculate the Reed-Muller spectra we use a Walsh-Hadamard matrix with dimensions $2^n \times 2^n$, where n is the number of bits. As n increases, the matrix becomes infeasible to create in memory. For example, using 1 byte to represent each bit, $n = 5$ uses 128 Gb for some cases of linear and permutation synthesis. Recall that these approaches generate 2^n bits for an input using n bits. In other words, the dimensions of the Walsh-Hadamard matrix are $2^k \times 2^k$, where $k = 2^n$ and n is the number of input bits.

Thus, Table 3 only shows the average values when all the cases were run successfully. That is, 40 cases for $n < 5$ and 20 cases for $n \geq 5$. The number of cases $p = 10$ was chosen for each strategy, which represents 40 cases in total. Further information can

Table 3. The post-synthesis strategies for each value of n, and its average for each strategy and n value.

	Number of input bits (n)					
Strategy	2	3	4	5	6	avg.
OT	**4.15**	16.28	65.55	73.35	269.35	85.74
OT+RM	4.40	17.08	63.65	50.55	165.75	60.29
OT+RM+RU	4.28	16.55	62.70	47.10	153.00	56.73
OT+RU	**4.15**	**16.00**	64.35	59.85	211.40	71.15
OT+RU+RM	4.40	17.08	63.15	**39.55**	**124.75**	49.79
RM	4.70	17.28	68.28	72.55	274.75	87.51
RM+OT	**4.15**	16.45	62.50	42.80	142.75	53.73
RM+OT+RU	**4.15**	16.08	**62.03**	40.65	129.25	50.43
RM+RU	4.63	17.00	67.85	71.10	270.00	86.12
RM+RU+OT	**4.15**	16.28	62.18	41.85	139.70	52.83
RU	4.63	17.40	71.68	122.20	492.30	141.64
RU+OT	**4.15**	16.13	64.00	64.45	239.45	77.64
RU+OT+RM	4.40	17.08	65.15	61.05	226.25	74.79
RU+RM	4.70	17.28	69.28	81.55	313.75	97.31
RU+RM+OT	**4.15**	16.45	64.00	44.80	153.75	56.63
avg.	4.35	16.69	65.09	60.89	220.41	

be accessed at `https://github.com/raphaelbernardino/rblk`.

As expected, for $n = \{2, 3\}$, the template strategy works best. In some cases, the template strategy is combined with another strategy and yields the same result. For higher values, such as $n = 4$, the RM+RU+OT is the best result. The overall best result is OT+RU+RM, but we think there is no statistical significance given that for $n = \{5, 6\}$ it only analyzes half of the cases.

Notice that OT+RU+RM and RM+RU+OT are the best in 2 out of 5 values, but the RM+RU+OT also achieves the second best result in 3 out of 5. Although the OT+RU+RM for $n = 2$ is third best, $n = 3$ is eighth best, and for $n = 4$ it is fifth best. Thus, the average does not reflect the ideal performance of the strategies.

Based on these preliminary results, we can conclude that the Reed-Muller strategy for synthesis yields promising outcomes. When we examine the post-synthesis strategies, as shown in Table 3, the proposed approach outperforms, on average, the post-synthesis for every value of n and in all cases.

6. Conclusion

The synthesis strategies discussed face challenges related to memory usage, gate count, and scalability as the input size n increases. For large values of n, approaches like Reed-Muller (RM) synthesis become computationally expensive due to the use of large matrices like the Walsh-Hadamard matrix. This leads to high memory requirements, making it difficult to handle cases when $n \geq 5$, limiting the analysis and reducing the statistical significance of results.

In terms of gate count and efficiency, the unitary synthesis approach is the most efficient for small n, using n bits. The permutation synthesis gate count increases rapidly as n grows, making it impractical for larger inputs as it utilizes 2^n bits. Permutation synthesis, which uses SWAP gates, results in a significant increase in gate count when translated into Toffoli gates, as each SWAP gate corresponds to three Toffoli gates. The linear synthesis outputs the best network of reversible circuits, but uses 2^{n-1} bits, making it infeasible for large values of n.

Reed-Muller synthesis, on the other hand, is the most balanced strategy for larger n, using only $2n$ bits and maintaining a reasonable growth in gate count, making it scalable for larger circuits. This strategy is preferred for $n > 2$, as it provides better results with fewer resources compared to the linear and permutation approaches.

For small values of n, the template strategy performs best, while for intermediate values like $n = 4$, combinations such as RM+RU+OT yield better results. For larger values of n, combinations like OT+RU+RM show the best performance, though memory constraints limit the ability to fully analyze all cases for $n = 5$ and $n = 6$. As a result, the choice of synthesis strategy depends heavily on the input size, with Reed-Muller synthesis being the most efficient for larger n.

References

Abbe, E., Shpilka, A., and Wigderson, A. (2015). Reed-muller codes for random erasures and errors. In *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*, pages 297–306.

Abbe, E., Shpilka, A., and Ye, M. (2020). Reed–muller codes: Theory and algorithms. *IEEE Transactions on Information Theory*, 67(6):3251–3277.

Bandyopadhyay, C. and Rahaman, H. (2014). Synthesis of esop-based reversible logic using positive polarity reed-muller form. In *Emerging Trends in Computing and Communication: ETCC 2014, March 22-23, 2014*, pages 363–376. Springer.

Barenco, A., Bennett, C. H., Cleve, R., DiVincenzo, D. P., Margolus, N., Shor, P., Sleator, T., Smolin, J. A., and Weinfurter, H. (1995). Elementary gates for quantum computation. *Physical review A*, 52(5):3457.

Benioff, P. (1980). The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines. *Journal of statistical physics*, 22:563–591.

Benioff, P. (1982a). Quantum mechanical hamiltonian models of turing machines. *Journal of Statistical Physics*, 29:515–546.

Benioff, P. (1982b). Quantum mechanical hamiltonian models of turing machines. *Journal of Statistical Physics*, 29:515–546.

Bernardino, R. and Kowada, L. (2025). Reversible circuit optimization using reed-muller spectrum and rules decomposition. In *2025 IEEE 16th Latin America Symposium on Circuits and Systems (LASCAS)*.

Dalcumune, E., Kowada, L. A. B., Ribeiro, A. C., Figueiredo, C. M. H., and Marquezino, F. L. (2021). A reversible circuit synthesis algorithm with progressive increase of controls in generalized toffoli gates. *JUCS - Journal of Universal Computer Science*, 27(6):544–563.

Falkowski, B. J. and Chang, C.-H. (1995). An exact minimizer of fixed polarity reed-muller expansions. *International journal of electronics*, 79(4):389–409.

Fredkin, E. and Toffoli, T. (1982). Conservative logic. *International Journal of Theoretical Physics*, 21(3):219–253.

Gottesman, D. (1998). The heisenberg representation of quantum computers. *arXiv preprint quant-ph/9807006*.

Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219.

IBM (2017). Qiskit. Available at `https://qiskit.org/`.

Kaufman, T., Lovett, S., and Porat, E. (2012). Weight distribution and list-decoding size of reed–muller codes. *IEEE transactions on information theory*, 58(5):2689–2696.

Knill, E. and Laflamme, R. (1997). Theory of quantum error-correcting codes. *Physical Review A*, 55(2):900.

Kutin, S. A., Moulton, D. P., and Smithline, L. M. (2007). Computation at a distance. *arXiv preprint quant-ph/0701194*.

Landauer, R. (1961). Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3):183–191.

Maslov, D. and Dueck, G. W. (2003). Garbage in reversible design of multiple output functions. In *6th International Symposium on Representations and Methodology of Future Computing Technologies*, pages 162–170.

Maslov, D., Dueck, G. W., and Miller, D. M. (2007). Techniques for the synthesis of reversible toffoli networks. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 12(4):42–es.

Nielsen, M. A. and Chuang, I. L. (2000). *Quantum Computation and Quantum Information*. Cambridge University Press, New York, NY, USA.

Patel, K. N., Markov, I. L., and Hayes, J. P. (2008). Optimal synthesis of linear reversible circuits. *Quantum Inf. Comput.*, 8(3):282–294.

Porwik, P. (2002). Efficient calculation of the reed-muller form by means of the walsh transform. *Int. J. Appl. Math. Comput. Sci*, 12(4):571–579.

Shor, P. W. (1999). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332.

Toffoli, T. (1980). Reversible computing. In de Bakker, J. and van Leeuwen, J., editors, *Automata, Languages and Programming*, page 632, New York. Springer. MIT Technical Memo No. MIT/LCS/TM-151, 1980 (unpublished).

Zakablukov, D. V. (2016). Application of permutation group theory in reversible logic synthesis. In *Reversible Computation: 8th International Conference, RC 2016, Bologna, Italy, July 7-8, 2016, Proceedings 8*, pages 223–238. Springer.