Quantum Ternary Circuit Synthesis Using Projection Operations

Sudhinden Bikash Mandal, Amlan Chakrabarti, Susmita Sur-Kolay

1 A. K. Choudhury School of Information Technology, University of Calcutta, India
2 Advanced Computing & Microelectronics Unit, Indian Statistical Institute, India

Basic logic gates and their operations in ternary quantum domain are involved in the synthesis of ternary quantum circuits. Only a few works define ternary algebra for ternary quantum logic realization. In this paper, a ternary logic function is expressed in terms of projection operations including a new one. A method to realize new multi-qutrit ternary gates in terms of generalized ternary gates and projection operations is also presented. We also introduced ten simplification rules for reducing ancilla qutrits and gate levels. Our method yields lower gate cost and fewer gate levels and ancilla qutrits than that obtained by earlier methods for the ternary benchmark circuits. The $n$ qutrit ternary sum function is synthesized without any ancilla qutrit by our proposed methodology.

Key words: Gate level logic synthesis, Ternary quantum circuits, Projection operations

1 INTRODUCTION

Multi-valued quantum computing is gaining importance in the field of quantum information theory and quantum cryptography as it can represent an $n$-dimensional quantum system, defined by the basis states $|0>$, $|1>$, ...,
The unit of information is called a qudit. A qudit exists as a linear superposition of states, and is characterized by a wave function $\psi$. Multi-valued quantum algebra comprises the rules for a set of basic logic operations that can be performed on qudits. While in [3] the structure of a multi-valued logic gate is proposed which can be experimentally feasible with a linear ion trap scheme for quantum computing, this approach produces large dimensional circuits. A universal architecture for multi-valued reversible logic is given in [4], but quantum realization of the circuits thus obtained is not apparent. The universality of $n$-qudit gates is presented in [5], but no algorithms for synthesis were given. Al-Rabedi proposed in [6] the minimization technique for multi-valued quantum galois field sum of product.

In this paper we concentrate on the quantum ternary functions. A ternary quantum system exists in linear superposition of three basis states, labeled $|0\rangle$, $|1\rangle$ and $|2\rangle$. All the operations on a qutrit are developed in a 3-dimensional Hilbert space under the field GF(3) [7]. A two qutrit vector can be represented as $|\psi\rangle = \sum_{i=0}^{8} |C_i| |2^{i}| = 1$. In general, a $n$ qutrit state can be represented as a superposition of $3^n$ basis states. A quantum register of size $n$ qutrits can hold $3^n$ values simultaneously, whereas the $n$-qubit register in binary quantum domain can hold $2^n$ values. For the same size of memory, a Quantum Fourier Transform (QFT) with ternary qutrits improves the approximation and increases the state space by a factor of $(3/2)^n$ [8]. This is very useful in cryptography.

The realization of ternary operations requires a set of universal gates with which a quantum circuit for a given ternary quantum function can be built. One set of universal gates [9] comprise the Ternary Swap, the Ternary NOT and the Ternary Toffoli gate. With these universal gates, any arbitrary quantum circuit can be realized without ancilla bits, but this approach is not optimal in terms of quantum cost [9]. Further, the synthesis method in [10] uses quantum multiplexers and the method of iterative deepening depth first search is employed to minimize the gate cost. Another synthesis technique was proposed in [11] for garbage free GF(3) based reversible or quantum logic circuit from its truth values, using only Muthukrishnan-Stroud (M-S) [3] and shift gates [12]. But this technique did not provide any simplification rule to reduce the gate count. In paper [13] the ternary quantum logic was expressed in terms of Ternary Galois Field Sum of Product, and 16 Ternary Galois Field expansions were also proposed. But this paper did not provide any gate level implementation of ternary benchmark circuit.

The paper here is the extended version of [14], where our motivation
is to synthesis a given quantum ternary benchmark [13] function specified as ternary minterms, using projection operations. Further we realize the gate level implementation of these benchmark [13] circuits using Generalized Ternary Gates [8] and newly defined permutative ternary quantum C^2-NOT gates. A few simplification rules are enumerated for reducing the number of ancilla qutrits and the levels of gates in the resulting quantum circuit.

The preliminary concepts of ternary algebra with a new projection operation appear in Section 2. In Section 3, the characteristics of basic ternary logic gates and the Generalized Ternary Gate (GTG) [12] along with the introduction of a few new gates are discussed. The proposed synthesis methodology along with a set of simplification rules is presented in Section 4. Synthesis of quantum ternary benchmark circuits by this method, and the respective quantum gate cost with ancilla qutrit count are given in Section 5. Concluding remarks appear in Section 6.

2 TERNARY ALGEBRA

We first define the basic operations over the set \{0, 1, 2\}.

2.1 Ternary AND, OR, NOT

The operations of AND, OR, and NOT [15] for ternary variables are:

\[
\text{AND}(a, b) = \begin{cases} 
  a & \text{if } a \leq b; \\
  b & \text{otherwise.}
\end{cases}
\]

\[
\text{OR}(a, b) = \begin{cases} 
  a & \text{if } a \geq b; \\
  b & \text{otherwise.}
\end{cases}
\]

\[
\text{NOT}(a) = a + 1.
\]

Here ‘+’ denotes the modulo 3 addition. It is easy to see that the ternary AND operation is distributive over the OR operation and vice-versa [15].

2.2 Ternary Projection Operations \(L_i\) and \(J_i\)

Our main goal is to define a given ternary function in terms of the minterms corresponding to its 1 or 2 values. The \(3^n\) minterms for a function \(f\) with \(n\) ternary variables are denoted by \(m_0\) to \(m_{3^n-1}\). In the sum of products form of minterms, the minterm \(m_0 = \prod_i NOT(x_i)\) where \(x = \{0, 1, 2\}\) and \(\prod\) denotes the ternary AND; its value is 1, 2, or 0 depending on whether for all \(i\) \(x_i = 0, 1, 2\) respectively.

As the outcomes of a ternary function may be 0, 1 or 2, we cannot express a ternary function \(f\) which results in either 1 or 2 for a given set of inputs, by using the above minterms \(m_i\). So, we present six projection operations, grouped into two types \(L_i\) and \(J_i\), where \(i = \{0, 1, 2\}\). While the \(J_i\) type operation was defined earlier [15] as
TABLE 1

<table>
<thead>
<tr>
<th>( a )</th>
<th>( L_0(a) )</th>
<th>( L_1(a) )</th>
<th>( L_2(a) )</th>
<th>( J_0(a) )</th>
<th>( J_1(a) )</th>
<th>( J_2(a) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

TABLE 1
Truth tables of ternary projection operations \( L_i \) and \( J_i \)

\[ J_i(a) = \begin{cases} 2 & \text{if } a = i; \\ 0 & \text{otherwise.} \end{cases} \]

the \( L_i \) types are newly defined here as

\[ L_i(a) = \begin{cases} 1 & \text{if } a = i; \\ 0 & \text{otherwise.} \end{cases} \]

Their truth tables appear in Table 1. From Table 1, we can verify that the output of any \( L_i \) operation is either 0 or 1, and that of \( J_i \) operation is either 0 or 2. Two other derived operations \( L_i' \), \( J_i' \) as shown in the Table 2. Next, we express a ternary function as a sum of products minterm form by using the \( L_i \) and \( J_i \) operations. The \( L_i \) and \( J_i \) operations are commutative, associative and distributive over ternary AND and OR logic. We can verify these laws given below, from Table 1. Here \( i, j, k = \{0, 1, 2\} \)

1. Commutativity
   (i) \( L_i(a) + L_j(b) = L_j(b) + L_i(a) \)
   \[ J_i(a) + J_j(b) = J_j(b) + J_i(a) \]
   (ii) \( L_i(a), L_j(b) = L_j(b), L_i(a) \)
   \[ J_i(a), J_j(b) = J_j(b), J_i(a) \]

2. Associativity
   (i) \( [L_i(a) + L_j(b)] + L_k(c) = L_i(a) + [L_j(b) + L_k(c)] \)
   \[ [J_i(a) + J_j(b)] + J_k(c) = J_i(a) + [J_j(b) + J_k(c)] \]
   (ii) \( [L_i(a), L_j(b)], L_k(c) = L_i(a), [L_j(b), L_k(c)] \)
   \[ [J_i(a), J_j(b)], J_k(c) = J_i(a), [J_j(b), J_k(c)] \]

3. Distributivity
   (i) \( L_i(a)[L_j(b) + L_k(c)] = L_i(a)\cdot L_j(b) + L_i(a)\cdot L_k(c) \)
   \[ J_i(a)[J_j(b) + J_k(c)] = J_i(a)\cdot J_j(b) + J_i(a)\cdot J_k(c) \]
   (ii) \( L_i(a)\cdot [L_j(b) + L_k(c)] = [L_i(a)\cdot L_j(b)] + [L_i(a)\cdot L_k(c)] \)
   \[ J_i(a)\cdot [J_j(b) + J_k(c)] = [J_i(a)\cdot J_j(b)] + [J_i(a)\cdot J_k(c)] \]
TABLE 2
Truth tables of ternary operations $L'_i$ and $J'_i$

<table>
<thead>
<tr>
<th>a</th>
<th>$L'_0(a)$</th>
<th>$L'_1(a)$</th>
<th>$L'_2(a)$</th>
<th>$J'_0(a)$</th>
<th>$J'_1(a)$</th>
<th>$J'_2(a)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

3 TERNARY LOGIC GATES

3.1 Ternary Feynman, Ternary Toffoli, MAX and MIN gates

The two qutrit ternary Feynman gate [16], shown in Figure 1, is defined as

$\text{Feynman}(A, B) = A + B$

where ‘+’ operator is addition over GF(3).

The ternary 3-qutrit Toffoli gate [16] in the universal ternary quantum get set, is shown in Figure 2 and is defined as

$\text{Toffoli}(A, B, C) = \begin{cases} 
Z \text{ operation on } C & \text{if } A = B = 2; \\
C & \text{otherwise};
\end{cases}$

where $A, B$ are controlling input and $Z = \{+1, +2, 01, 02, 12\}$.

In our synthesis method, ternary SWAP and NOT gates are not used explicitly. Instead, we use the ternary MAX and MIN gates [10] respectively to replace the OR and the AND in ternary quantum logic. These two gates are defined as

$\text{MAX}(A_1, A_2, \ldots A_n, B) = \begin{cases} 
A_i & \text{if } A_i \geq A_j, i \neq j \text{ and } A_i \geq B; \\
B & \text{if } \forall i, B \geq A_i;
\end{cases}$

$\text{MIN}(A_1, A_2, \ldots A_n, B) = \begin{cases} 
A_i & \text{if } A_i \leq A_j, i \neq j \text{ and } A_i \leq B; \\
B & \text{if } \forall i, B \leq A_i;
\end{cases}$

where $i = \{1, 2, \ldots n\}$ (Figure 3).

The ternary Toffoli gate can be realized without any ancilla qutrits [16, 17] by using Muthukrishnan-Stroud gates (M-S gate) and . An M-S gate is defined as

$\text{M-S}(A, B) = \begin{cases} 
\text{B shift by } Z & \text{if } A = 2; \\
\text{B} & \text{otherwise};
\end{cases}$

where $Z = \{+1, +2, 01, 02, 12\}$ [15] (Figure 4a).

3.2 Generalized Ternary Gate

In order to realize a Generalized ternary gate (GTG) defined next, the Shift gates [12] given in Table 3 are required. Here addition and multiplication
FIGURE 1
2 qutrit ternary Feynman Gate

A + B where '+' operator is addition over GF(3)

FIGURE 2
3 qutrit ternary Toffoli Gate

FIGURE 3
Ternary Multi-qutrit (a) MAX gate, and (b) MIN gate
FIGURE 4
(a) A Muthukrishnan-Stroud (M-S) gate, and (b) a Generalized Ternary Gate (GTG)

TABLE 3
Ternary Shift operations used in a GTG

<table>
<thead>
<tr>
<th>Shift gate</th>
<th>Its operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer</td>
<td>$x = x$</td>
</tr>
<tr>
<td>Single Shift</td>
<td>$x = x + 1$</td>
</tr>
<tr>
<td>Dual Shift</td>
<td>$x = x + 2$</td>
</tr>
<tr>
<td>Self Shift</td>
<td>$x = 2x$</td>
</tr>
<tr>
<td>Self Single Shift</td>
<td>$x = 2x + 1$</td>
</tr>
<tr>
<td>Self Dual Shift</td>
<td>$x = 2x + 2$</td>
</tr>
</tbody>
</table>
over $GF(3)$ are denoted by ‘+’ and ‘.’ respectively.

A Generalized Ternary Gate (GTG) is a 2-qutrit gate, as shown in Figure 4(b). The controlling input of a GTG can be used to select the 1-qutrit shift operation on the target input. The GTG is defined as

$$\text{GTG}(A, B) = \begin{cases} 
B \text{ shift } X & \text{if } A=0; \\
B \text{ shift } Y & \text{if } A=1; \\
B \text{ shift } Z & \text{if } A=2
\end{cases}$$

where $X$, $Y$, and $Z$ are any three distinct shift operations in Table 3.

**Implementation of $L_i$ and $J_i$ operations using GTG**

We can implement the $L_i$ and $J_i$ projection operations defined in Section 2.2, by using GTG as shown in Figures 5 (a) and (b) respectively. For the $L_i$ type operation, we set $b = 1$.

$$L_i(a) = \text{GTG}(a, b) = \begin{cases} 
b & \text{if } a=i; \\
b+2 & \text{if } a=i+1; \\
2b+1 & \text{if } a=i+2
\end{cases}$$

For the $J_i$ type operation, we set $b = 2$.

$$J_i(a) = \text{GTG}(a, b) = \begin{cases} 
b & \text{if } a=i; \\
b+1 & \text{if } a=i+1; \\
2b+2 & \text{if } a=i+2
\end{cases}$$

### 3.3 A new Ternary $C^2 NOT$ gate

Several multi-qutrit control operations are possible in ternary logic. We present a new definition of a 3-qutrit $C^2 NOT$ (Figure 6(a)), which is required to realize the simplification rules for ternary minterms given in the Section 4.2 below, as
3.4 A Multi-qutrit GTG

We also define a Multi-qutrit GTG (Figure 7) as

$$\text{GTG}(A_1, A_2, ..., A_n, B) = \begin{cases} 
\text{B shift } X & \text{if } A_1, A_2, ..., A_n = 0; \\
\text{B shift } Y & \text{if } A_1, A_2, ..., A_n = 1; \\
\text{B shift } Z & \text{if } A_1, A_2, ..., A_n = 2; \\
\text{B} & \text{otherwise}; 
\end{cases}$$

where $A_1, A_2, ..., A_n$ are the control inputs, and $B$ is the target input. The realization of this $C^2 NOT$ gate with M-S gates is shown in Figure 6(b).
4 PROPOSED SYNTHESIS METHODOLOGY

4.1 Overview
Consider an $m$-variable ternary quantum logic function $f(a_1, a_2, \ldots, a_m) = \sum_{i=0}^{n} (\text{minterms for one}) + \sum_{j=0}^{p} (\text{minterms for two})$ where $\sum$ implies logical ternary OR, $n$ and $p$ are respectively the number of input vectors for which $f$ is 1, and 2. Thus, for $(3^m - n - p)$ input vectors, $f$ is 0. We express the minterm for one and two by using the $L_i$ and $J_i$ operations respectively. From Table II, we can say

$$\prod_{i=1}^{m} L_0(a_i) = \begin{cases} 1 & \text{if } \forall i a_i = 0; \\ 0 & \text{if } \exists i a_i = 1 \text{ or } 2; \end{cases}$$

$$\prod_{i=1}^{m} L_1(a_i) = \begin{cases} 1 & \text{if } \forall i a_i = 1; \\ 0 & \text{if } \exists i a_i = 0 \text{ or } 2; \end{cases}$$

$$\prod_{i=1}^{m} L_2(a_i) = \begin{cases} 1 & \text{if } \forall i a_i = 2; \\ 0 & \text{if } \exists i a_i = 0 \text{ or } 1; \end{cases}$$

$$\prod_{i,p,k=1}^{m} L_0(a_i).L_1(a_p).L_2(a_k) = \begin{cases} 1 & \text{if } \forall i, p, k a_i = 0, a_p = 1, a_k = 2; \\ 0 & \text{if } \exists i, p, k a_i = 1 \text{ or } 2, a_p = 0 \text{ or } 2, a_k = 0 \text{ or } 1; \end{cases}$$

where $i + p + k = m$.

Hence, the minterms for which $f = 1$ are
1. $\prod_{i=1}^{m} L_0(a_i) = 0$,
2. $\prod_{i=1}^{m} L_1(a_i) = 1$,
3. $\prod_{i=1}^{m} L_2(a_i) = 2$, and
4. $\prod_{i,p,k=1}^{m} L_0(a_i = 0).L_1(a_p = 1).L_2(a_k = 2)$.

Similarly from Table I, the minterms for which $f = 2$ are
1. $\prod_{i=1}^{m} J_0(a_i) = 0$,
2. $\prod_{i=1}^{m} J_1(a_i) = 1$,
3. $\prod_{i=1}^{m} J_2(a_i) = 2$, and

FIGURE 8
The proposed Multi-qutrit GTG
4. \( \prod_{i,p,k=1}^{n} J_0(a_i = 0).J_1(a_p = 1).J_2(a_k = 2) \).

### 4.2 Simplification Rules

Next, we define seven simplification rules for our proposed method, the first four are derived from Table 1, and the next three from Table 2.

1. \( L_i(a).0 = 0 \), and \( J_i(a).0 = 0 \)
2. \( L_i(a).1 = L_i(a) \), and \( J_i(a).2 = J_i(a) \)
3. \( L_i(a) + 0 = L_i(a) \), and \( J_i(a) + 0 = J_i(a) \)
4. \( L_i(a) + 1 = 1 \), and \( J_i(a) + 2 = 2 \)
5. \( L_i(a).L_i'(a) = 0 \), and \( J_i(a).J_i'(a) = 0 \)
6. \( L_i(a) + L_i'(a) = 1 \), and \( J_i(a) + J_i'(a) = 2 \)
7. \( L_i'(a) = L_{i+1}(a) + L_{i+2}(a) \), and \( J_i'(a) = J_{i+1}(a) + J_{i+2}(a) \).

### Simplification Rules for reducing ancilla qutrits

For gate level realization of the projection operations \( L_i \) and \( J_i \), we need an ancilla qutrit for each. Further, to synthesize an \( m \)-variable ternary function with \( n \) minterms specified according to our proposed methodology, we have maximum of \( n\times m \) ancilla qutrits. However, we can reduce the number of ancilla qutrits by the following three simplification rules based on the new ternary \( C^2 \text{NOT} \) gate and Table 2:

8. \( L_i(a) L_i(b) + L_2(a) L_1(b) = C^2 \text{NOT}(a, b, 0) \) and \( J_i(a) J_2(b) + J_2(a) J_1(b) = C^2 \text{NOT}(a, b, 1) \)
9. \( L_i(a).L_i(a) = L_i(a) \) and \( J_i(a).J_i(a) = J_i(a) \)
10. \( L_i(a_1) L_i(a_2) \ldots L_i(a_n) = L_i(a_1, a_2, \ldots, a_n) \) and \( J_i(a_1) J_i(a_2) \ldots J_i(a_n) = J_i(a_1, a_2, \ldots, a_n), i \in \{0, 1, 2\} \).

The multiqutrit \( L_i \) and \( J_i \) operations in Rule 10 can be realized using Multiqutrit GTG as follows:

For \( L_i(a_1, a_2, \ldots, a_n), GTG(a_1, a_2, \ldots, a_n, B) = \)

\[
\begin{cases}
B + 1 & \text{if } a_1, a_2, \ldots, a_n = i; \\
B & \text{if } a_1, a_2, \ldots, a_n = i + 1; \\
2B & \text{if } a_1, a_2, \ldots, a_n = i + 2; \\
B & \text{otherwise};
\end{cases}
\]

For \( J_i(a_1, a_2, \ldots, a_n), GTG(a_1, a_2, \ldots, a_n, B) = \)

\[
\begin{cases}
B + 2 & \text{if } a_1, a_2, \ldots, a_n = i; \\
B & \text{if } a_1, a_2, \ldots, a_n = i + 1; \\
2B & \text{if } a_1, a_2, \ldots, a_n = i + 2; \\
B & \text{otherwise};
\end{cases}
\]

### 4.3 An Example

For illustrating our method, consider an arbitrary ternary function \( g(a, b) \) with its truth table given in Table 4.
TABLE 4
Truth table for an example ternary 2-qutrit function \( g(a, b) \)

<table>
<thead>
<tr>
<th>( a, b )</th>
<th>( 00 )</th>
<th>( 01 )</th>
<th>( 02 )</th>
<th>( 10 )</th>
<th>( 11 )</th>
<th>( 12 )</th>
<th>( 20 )</th>
<th>( 21 )</th>
<th>( 22 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g(a, b) )</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

FIGURE 9
Ternary quantum gate circuit obtained for \( g(a, b) \) in Table 4

The disjunction of minterms for which \( g = 1 \) is
\[
L_0(a)L_1(b) + L_1(a)L_0(b) + L_1(a)L_1(b) + L_1(a)L_2(b) + L_2(a)L_1(b)
\]
and that for which \( g = 2 \) is
\[
J_0(a)J_2(b) + J_2(a)J_0(b) + J_2(a)J_2(b).
\]
Hence, \( g(a, b) = L_0(a)L_1(b) + L_1(a)L_0(b) + L_1(a)L_1(b) + L_1(a)L_2(b) + L_2(a)L_1(b) + J_0(a)J_2(b) + J_2(a)J_0(b) + J_2(a)J_2(b). \)
By simplification rules 1 and 10, we get
\[
g(a, b) = L_0(a)L_1(b) + L_1(a)L_0(b) + L_1(a)b + C^2 NOT(a, b, 0) + J_0(a)J_2(b) + J_2(a)J_0(b) + J_2(a, b).
\]
The gate level implementation of the function \( g(a, b) \) is shown in Figure 8.

5 SYNTHESIS OF TERNARY BENCHMARK FUNCTIONS

We experimented with the ternary benchmark functions provided in [14]. Their truth tables appear in Table 5. The definition and the synthesis of the ternary benchmark functions using ternary minterm are given in the following
5.1 2-qutrit multiplication functions mul2 and mul2c

The functions are defined as:

\[ \text{mul2}(a,b) = \text{abmod3}, \text{ multiplication output}; \]

\[ \text{mul2c}(a,b) = \text{int}[ab/3], \text{carry of the multiplier}. \]

The truth table of \( \text{mul2}(a,b) \) and \( \text{mul2c}(a,b) \) appear in columns 3 and 4 respectively of Table 5.

Expressing in terms of projection operators, we get

\[ \text{mul2}(a,b) = L_1(a)L_1(b) + L_2(a)L_2(b) + J_1(a)J_2(b) + J_2(a)J_1(b). \]

\[ = L_1(a,b) + L_2(a,b) + C^2 \text{NOT}(a,b,1) \] (by Rules 8 and 10).

Similarly, \( \text{mul2c}(a,b) = L_2(a)L_2(b) = L_2(a,b) \) (by Rule 10).

5.2 2-qutrit half-adder thadd

The 2-qutrit half-adder \( \text{thadd} \) comprises the ternary half adder sum (\( \text{sumh} \)) and carry (\( \text{carryh} \)) functions, whose truth tables are given in columns 5 and 6 respectively of Table 5.

We re-write \( \text{sumh} \) according to our proposed methodology as

\[ \text{sumh} = L_0(a)L_1(b) + L_1(a)L_0(b) + L_2(a)L_2(b) + J_0(a)J_2(b) + J_1(a)J_1(b) + J_2(a)J_0(b). \]

Further simplification of \( \text{sumh} \) by using the ternary Feynman gate of Section 3.1 gives

\[ \text{sumh} = a + b. \]

Next, the carry function \( \text{carryh} \) is written as

\[ \text{carryh} = L_1(a)L_2(b) + L_2(a)L_1(b) + L_2(a)L_2(b) = C^2 \text{NOT}(a,b,0) + L_2(a,b) \] (by Rules 8 and 10).

5.3 GF(3) sum of two squares sqsum2

The definition of this ternary function is \( \text{sqsum2}(a,b) = (a^2 + b^2) \text{mod3} \) and its truth table is shown in column 7 of Table 5.

\[ \text{sqsum2}(a,b) = L_0(a)L_1(b) + L_0(a)L_2(b) + L_1(a)L_0(b) + L_2(a)L_0(b) + J_1(a)J_1(b) + J_1(a)J_2(b) + J_2(a)J_1(b) + J_2(a)J_2(b) \]

\[ = L_0(a)L_0(b) + L_0(a)L_0(b) + J_1(a,b) + C^2 \text{NOT}(a,b,2) + J_2(a,b) \]

(by Rules 8, 7 and 10).

5.4 Average function avg2

The function \( \text{avg2} \) is the integer part of the average of two ternary input variables in \( \text{modulo3} \) and defined as \( \text{avg2}(a,b) = \text{int}[ab/2] \). Its truth table is in column 8 of Table 5.

\[ \text{avg2} = L_0(a)L_2(b) + L_1(a)L_1(b) + L_1(a)L_2(b) + L_2(b)L_0(b) + L_2(a)L_1(b) + J_2(a)J_2(b) \]
TABLE 5
Truth table of four 2-qutrit benchmark functions[13]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>mul2</th>
<th>mul2e</th>
<th>sumh</th>
<th>carryh</th>
<th>sqsum2</th>
<th>avg2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

\[ L_0(a)L_2(b) + L_1(a, b) + C^2 \text{NOT}(a, b, 1) + L_2(a)L_0(b) + J_2(a, b) \]
(by Rules 8 and 10).

5.5 3-qutrit multiplication function mul3

The function mul3 is defined as
mul3(a, b, c) = abc\text{mod}3, multiplication output;
mul3(a, b) = \text{int}[abc/3], carry of the multiplier.

From the truth table of mul3(a, b, c) in column 4 of Table 6, we have
mul3(a, b, c) = L_1(a)L_1(b)L_1(c) + L_1(a)L_2(b)L_2(c) + L_2(a)L_1(b)L_2(c) +
L_2(a)L_2(b)L_1(c) + J_1(a)J_1(b)J_2(c) + J_1(a)J_2(b)J_1(c) +
J_2(a)J_1(b)J_1(c) + J_2(a)J_2(b)J_2(c)

\[ = L_1(a)[L_1(b)L_1(c) + L_2(b)L_2(c)] + L_2(a)[L_1(b)L_2(c) +
L_2(b)L_1(c)] + J_1(a)[J_1(b)J_2(c) + J_2(b)J_2(c)] + J_2(a)[J_1(b)J_1(c) +
J_2(b)J_2(c)] \]

By applying \(L_j\) and \(J_i\) operations on mul2 in column 3 of Table 5, we get
\[ = L_1(a)J_1(\text{mul2}(b, c)) + L_2(a)J_2(\text{mul2}(b, c)) +
J_1(a)J_2(\text{mul2}(b, c)) + J_2(a)J_1(\text{mul2}(b, c)) \]

\[ = \text{mul2}(a, \text{mul2}(b, c)) \]

From the truth table of mul3c(a, b, c) in column 5 of Table 6, we have
mul3c(a, b, c) = L_1(a)L_2(b)L_2(c) + L_2(a)L_1(b)L_2(c) + L_2(a)L_2(b)L_1(c) +
J_2(a)J_1(b)J_2(c)

\[ = C^2 \text{NOT}(a, b, 0)L_2(c) + L_2(a, b)L_1(c) + J_2(a, b, c) \]
(by Rules 8 and 10).
TABLE 6
Truth table of four 3-qutrit benchmark functions [13]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>mul3</th>
<th>mul3c</th>
<th>a2bcc</th>
<th>avg3</th>
<th>sqsum3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
5.6 Function $a^2bc$

This is an arbitrary function defined as $a^2bc = (a^2 + bc + c) \text{mod} 3$ with its truth table shown in column 6 of Table 6. Hence,
\[ a^2bc = L_0(a)L_0(b)L_1(c) + L_0(a)L_1(b)L_2(c) + L_1(a)L_0(b)L_0(c) + L_1(a)L_1(b)L_1(c) + L_1(a)L_2(b)L_2(c) + L_2(a)L_0(b)L_0(c) + L_2(a)L_1(b)L_1(c) + L_2(a)L_2(b)L_2(c) + J_0(a)J_0(b)J_2(c) + J_0(a)J_1(b)J_1(c) + J_1(a)J_0(b)J_1(c) + J_2(a)J_0(b)J_1(c) + J_2(a)J_1(b)J_2(c) \]

5.7 Function $avg3$ function

This function is the integer part of the average of three ternary input variables expressed as $\text{modulo} 3$, i.e., $avg3 = \lfloor a + b + c/3 \rfloor \text{mod} 3$. Its truth table is shown in column 7 of Table 6.

\[ avg3 = L_0(a)L_1(b)L_2(c) + L_0(a)L_2(b)L_1(c) + L_0(a)L_2(b)L_2(c) + L_1(a)L_0(b)L_1(c) + L_1(a)L_1(b)L_2(c) + L_1(a)L_2(b)L_0(c) + L_1(a)L_2(b)L_1(c) + L_2(a)L_0(b)L_1(c) + L_2(a)L_0(b)L_2(c) + L_2(a)L_1(b)L_1(c) + L_2(a)L_1(b)L_2(c) + L_2(a)L_1(b)L_0(c) + L_2(a)L_2(b)L_0(c) + L_2(a)L_2(b)L_1(c) + L_2(a)L_2(b)L_2(c) + J_2(a)J_2(b)J_2(c) \]

(by Rules 8 and 10)

\[ = L_0(a)C^2 \text{NOT}(b, c, 0) + L_0(a)L_2(b, c) + L_0(b)C^2 \text{NOT}(a, c, 0) + L_1(a)C^2 \text{NOT}(b, c, 0) + L_0(a)C^2 \text{NOT}(a, b, c) + L_0(b)L_2(a, c) + L_2(a)L_1(b, c) + L_2(a)L_0(b, c) + J_2(a, b, c) \]

(by Rule 7).

5.8 GF(3) sum of three squares $sqsum3$

The definition is $sqsum3(a, b, c) = (a^2 + b^2 + c^2) \text{mod} 3$ with the truth table in column 8 of Table 6.

\[ sqsum3(a, b) = L_0(a)L_0(b)L_1(c) + L_0(a)L_0(b)L_2(c) + L_0(a)L_1(b)L_0(c) + L_0(a)L_2(b)L_0(c) + L_1(a)L_0(b)L_0(c) + L_2(a)L_0(b)L_0(c) + J_0(a)J_1(b)J_1(c) + J_0(a)J_1(b)J_2(c) + J_0(a)J_2(b)J_1(c) + J_0(a)J_2(b)J_2(c) + J_1(a)J_0(b)J_1(c) + J_1(a)J_0(b)J_2(c) + J_1(a)J_1(b)J_0(c) + J_1(a)J_1(b)J_2(c) + J_2(a)J_0(b)J_1(c) + J_2(a)J_0(b)J_2(c) + J_2(a)J_1(b)J_0(c) + J_2(a)J_1(b)J_2(c) + J_2(a)J_2(b)J_0(c) + J_2(a)J_2(b)J_2(c) \]

= $L_0(a,b)L_1(c) + L_0(a,b)L_2(c) + L_0(a,c)L_1(b) + L_0(a,c)L_2(b) + \ldots$
\[ L_1(a)L_0(b,c)+L_2(a)L_0(b,c)+J_0(a)J_1(b,c)+J_0(a)C^2 NOT(b,c,1)+J_0(a)J_1(b,c)+J_1(a,c)J_1(b)+J_0(b)C^2 NOT(a,c,1)+J_1(a,b)J_0(c)+C^2 NOT(a, b, 1)J_0(c)+J_2(a, c)J_0(b)+J_2(a, b)J_0(c) \] (by Rules 8 and 10)

\[ = L_0(a,b)L_0'(c)+L_0(a,c)L_0'(b)+L_0(a)J_1(b,c)+J_0(a)C^2 NOT(b,c,1)+J_1(a,b)J_0(c)+C^2 NOT(a, b, 1)J_0(c)+J_2(a, c)J_0(b)+J_2(a, b)J_0(c) \]

(by Rule 9)

5.9 **GF(3) Product prod4**

The function \( prod4 \) is defined as \( prod4(a, b, c, d) = (abcd) \mod 3 \).

\[
prod4(a, b, c, d) = L_1(a)L_1(b)L_1(c)L_1(d)+L_1(a)L_1(b)L_2(c)L_2(d)+L_1(a)L_2(b)L_1(c)L_2(d)+L_1(a)L_2(b)L_2(c)L_1(d)+L_2(a)L_2(b)L_1(c)L_1(d)+L_2(a)L_2(b)L_2(c)L_2(d)+J_1(a)J_1(b)J_1(c)J_1(d)+J_1(a)J_1(b)J_1(c)J_2(d)+J_1(a)J_1(b)J_2(c)J_2(d)+J_2(a)J_1(b)J_1(c)J_1(d)+J_2(a)J_1(b)J_1(c)J_2(d)+J_2(a)J_1(b)J_2(c)J_2(d)+J_2(a)J_2(b)J_1(c)J_1(d)+J_2(a)J_2(b)J_1(c)J_2(d)+J_2(a)J_2(b)J_2(c)J_2(d)+J_2(a)J_2(b)J_2(c)J_1(d)+J_1(a)J_2(b)J_2(c)J_1(d)
\]

\[
= \text{mul2(mul2}(a, b), \text{mul2}(c, d)) \] (by Rules 8, 10 and projection operation on \( \text{mul2} \) in column 3 of Table 5).

5.10 **4-qutrit GF(3) Addition Function sum4**

This function is \( \text{sum4}(a, b, c, d) = (a + b + c + d) \mod 3 \). Along the lines of the result for ternary full adder in \([14]\), we have by applying the simplification rules 8, 10 and projection operation on \( \text{sumh} \) in column 5 of Table 5, the minterms of \( \text{sum4} \) as

\[
\text{sum4}(a, b, c, d) = L_0(a+b)L_1(c+d)+L_1(a+b)L_0(c+d)+L_2(a+b)L_2(c+d)+J_0(a+b)J_2(c+d)+J_1(a+b)J_1(c+d)+J_2(a+b)J_0(c+d) = \text{sumh(sumh}(a, b), \text{sumh}(c, d))
\]

5.11 **Synthesis Results**

We synthesized the ternary benchmark circuit provided in \([13]\) using our proposed synthesis methodology. The simplified forms of these functions by our method appear in Section 5.1 to 5.10 and the costs are summarized in Table 7. While the second column of the table indicates the maximum number of ancilla qutrits that may be required to synthesize the benchmark circuits, the third column shows the number of ancilla qutrit required after using our simplification rules in Section 4.2.

The fourth column shows the M-S gate count cost. The quantum gate cost of the ternary quantum benchmark circuits are evaluated in terms of the number of M-S gates used. The number of M-S gates required to realize a
Feynman gate, a Toffoli and a GTG gate are 4, 5 and 5 respectively \[11,18\]. To realize the new ternary \(C^2 NOT\) gate shown in Figure 7b, we need 8 M-S gates.

For the ternary \(sumn\) function, the number of ancilla qutrits is zero and the quantum gate cost is \((n - 1) \times 4\). In the case of ternary \(prodn\) function, the maximum number of ancilla qutrits that may be required is \(2^n \times n\). But by applying our simplification rules, we need only \((n - 1) \times 3\) ancilla qutrits for the realization of \(prodn\) function, with a quantum gate cost of \((n - 1) \times 18\) for this function.

Our proposed simplification rules reduce more than 50% of the ancilla qutrits for the functions \(mul2\), \(mul3\), \(avgn\) and \(sqsumn\). The comparison of the M-S gate count cost with that in \[19\] for these benchmark circuits are given in Table 7. This establishes that although our design has a small increase in cost for the \(sqsumn\) and \(avgn\) circuits, it is 20% less for all the other benchmark circuits.

6 CONCLUSION

In this paper, we have proposed a methodology for logic synthesis of ternary quantum circuits. We have defined a minterm based approach of expressing a ternary logic function by using \(L_i\) and \(J_i\) operations. We have also stated the simplification rules for the proposed method. By applying these simplification rules, we can realize the ancilla free \(sumn\) function. Among the ternary benchmark circuits \[13\], the quantum gate cost for the group of addition and multiplication functions \(sumn\) and \(prodn\), by our method is smaller compared to that by the earlier method in \[19\].

REFERENCES


### TABLE 7
Quantum Gate cost of Ternary Benchmark Circuit

<table>
<thead>
<tr>
<th>Circuit Name</th>
<th>Maximum Ancilla qutrit</th>
<th>Reduced Ancilla qutrit</th>
<th>our [19]</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum2</td>
<td>12</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>sum3</td>
<td>54</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>sum4</td>
<td>216</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>sum5</td>
<td>810</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>sum6</td>
<td>2916</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>sum7</td>
<td>10206</td>
<td>0</td>
<td>24</td>
</tr>
<tr>
<td>prod2</td>
<td>8</td>
<td>3</td>
<td>18</td>
</tr>
<tr>
<td>prod3</td>
<td>24</td>
<td>6</td>
<td>36</td>
</tr>
<tr>
<td>prod4</td>
<td>64</td>
<td>9</td>
<td>54</td>
</tr>
<tr>
<td>prod5</td>
<td>160</td>
<td>12</td>
<td>72</td>
</tr>
<tr>
<td>prod6</td>
<td>384</td>
<td>15</td>
<td>90</td>
</tr>
<tr>
<td>prod7</td>
<td>896</td>
<td>18</td>
<td>108</td>
</tr>
<tr>
<td>mul2</td>
<td>10</td>
<td>4</td>
<td>23</td>
</tr>
<tr>
<td>mul3</td>
<td>36</td>
<td>11</td>
<td>64</td>
</tr>
<tr>
<td>thadd</td>
<td>18</td>
<td>2</td>
<td>21</td>
</tr>
<tr>
<td>tfadd</td>
<td>63</td>
<td>4</td>
<td>42</td>
</tr>
<tr>
<td>avg2</td>
<td>12</td>
<td>7</td>
<td>38</td>
</tr>
<tr>
<td>avg3</td>
<td>51</td>
<td>16</td>
<td>89</td>
</tr>
<tr>
<td>sqsum2</td>
<td>16</td>
<td>7</td>
<td>38</td>
</tr>
<tr>
<td>squsum3</td>
<td>54</td>
<td>24</td>
<td>130</td>
</tr>
</tbody>
</table>


