IEICE TRANS. INF. & SYST., VOL.E97–D, NO.9 SEPTEMBER 2014

PAPER *Special Section on Multiple-Valued Logic and VLSI Computing*

# Synthesis of Quantum Arrays from Kronecker Functional Lattice Diagrams

**Martin LUKAC**[†a)], **Dipal SHAH**[††], **Marek PERKOWSKI**[††], ***Nonmembers***, ***and*** **Michitaka KAMEYAMA**[†], ***Member***

**SUMMARY** Reversible logic is becoming more and more popular due to the fact that many novel technologies such as quantum computing, low power CMOS circuit design or quantum optical computing are becoming more and more realistic. In quantum computing, reversible computing is the main venue for the realization and design of classical functions and circuits. We present a new approach to synthesis of reversible circuits using Kronecker Functional Lattice Diagrams (KFLD). Unlike many of contemporary algorithms for synthesis of reversible functions that use $n \times n$ Toffoli gates, our method synthesizes functions using $3 \times 3$ Toffoli gates, Feynman gates and NOT gates. This reduces the quantum cost of the designed circuit but adds additional ancilla bits. The resulting circuits are always regular in a 4-neighbor model and all connections are predictable. Consequently resulting circuits can be directly mapped in to a quantum device such as quantum FPGA [14]. This is a significant advantage of our method, as it allows us to design optimum circuits for a given quantum technology.

***key words:*** *reversible circuits synthesis, kronecker lattices, quantum computing*

## 1. Introduction

The synthesis of reversible (permutative) circuits is an important problem in quantum computing because reversible circuits are an important component in various quantum algorithms. Reversible circuits appear as oracles in Grover algorithm [7], as modulo arithmetic part in Shor algorithm [23], as components in Deutsch-Jozsa algorithm [4] as well as in parts of *quantum simulation* algorithms [9] such as the many-body problem. While the Grover and Shor algorithms are two of the most famous quantum algorithms, the simulation of many-body systems is one of the most important quantum mechanical problems to be solved. Thus the design of circuits with small gate count for these algorithms and problems is crucial in the development of a competitive full-scale quantum computer.

Currently some of the well known algorithms are based on function transformation approaches [10], [15], [16], ESOP transformation [6] or representation transformation [25]. Most of these algorithms however generate the final circuit containing $n \times n$ Toffoli gates ($n$ input bits and $n$ output bits). This is quite problematic because in quantum technology such gates do not exist; they have to be designed from $3 \times 3$ Toffoli gates, CNOT gates and NOT gates. Consequently circuits containing such large gates are post processed, large gates are decomposed into small reversible primitives and only then the circuit is transformed into truly quantum gates and is minimized. Using post-processing to reduce the cost of the circuits is however problematic because such minimization can lead to highly non optimal circuits: designing circuits directly from smaller Toffoli gates permits to minimize the circuit in such places that cannot be attained when synthesizing circuits using arbitrary large Toffoli gates.

In this paper we provide an extended study of the synthesis of reversible circuits using Kronecker Functional Lattice Diagrams (KFLD) method that was originally proposed in [22]. The main contributions of this paper are:

1. We provide details on the decompositions used in the KFLD algorithm as well as details on the algorithm itself.
2. We improve the algorithm proposed in [22] by optimizing the KFLDs by two optimization algorithms.
3. We show that the KFLD method is superior to other state-of-art algorithms by providing a set of new updated benchmark results.

This paper is organized as follows. Section 2 provides background on Kronecker Functional Lattice Diagrams (KFLDs). Section 3 presents method for creating a Kronecker Functional Lattice Diagram using positive Davio gate as a basic building block. Section 4 shows method to convert Kronecker Functional Lattice Diagram into a quantum circuit consisting of $3 \times 3$ Toffoli gates, Feynman gates and NOT gates. Section 5 presents two optimization methods for the KFLDs. Experimental results are given in Sect. 6 and finally Sect. 7 concludes the paper and discusses future work.

## 2. Background

A decision diagram (DD) for an arbitrary logic function of $n$ variables $f(x_1, \ldots, x_n)$ is a rooted directed acyclic graph (DAG) $G = (V, E)$ with two types of nodes, terminal nodes and non-terminal nodes. A non-terminal node is labeled with Boolean variable $x_n$ and has two child nodes, $low(v) \in V$ and $high(v) \in V$. A terminal node has no child nodes and is labeled with logic 0(1). The edge $e \in E$ from a node to a $low$($high$) child presents assignment of variable $x_n$ to

Manuscript received December 4, 2013.
Manuscript revised April 11, 2014.

[†]The authors are with the Graduate School of Information Sciences, Tohoku University, Sendai-shi, 980–8579 Japan.
[††]The authors are with the Department of Computer and Electrical Engineering, Portland State University, Portland, OR, USA.
a) E-mail: lukacm@ecei.tohoku.ac.jp

DOI: 10.1587/transinf.2013LOP0015

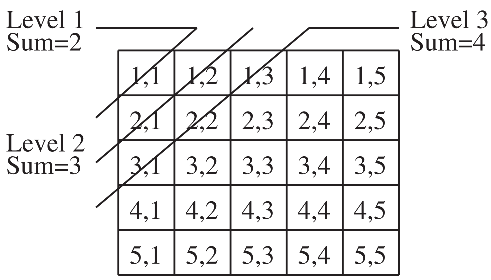



**Fig. 1** The Enumeration of cells of the Akers array [2]. Lattice is only a part of Akers array, starting from top left corner.

logic 0(1). A Decision Diagram is *free* if each variable is encountered at most once in each path in the DD from the root to a terminal node. A DD is *ordered* if it is free and the variables are encountered in the same order on each path from the root node to a terminal node.

Expanding an arbitrary logic function of $n$ variables $f(x_1, \ldots, x_n)$ using positive Davio expansion, negative Davio expansion or Shannon expansion [5], [8] as shown below in Eqs. (1), (2) and (3) respectively, results in a Kronecker Decision Diagram.

$$f = x_1(f_0 \oplus f_1) \oplus f_0 \tag{1}$$

$$f = \overline{x_1}(f_0 \oplus f_1) \oplus f_0 \tag{2}$$

$$f = \overline{x_1} f_0 \oplus x_1 f_1 \tag{3}$$

Kronecker Decision Diagrams were extended to Kronecker Lattice Diagrams (KLDs) in [20]. A Lattice Diagram uses a regular structure to represent relations between the individual logic components. The regular structure is specified by a diagonal matrix where every entry of the matrix $L[i, j]$ is a node (Fig. 1). For every node $L[i, j]$, $L[i{+}1, j]$ ($L[i, j{+}1]$) is the left (right) predecessor; $L[i, j-1]$ ($L[i-1, j]$) is the left (right) successor and $L[i+1, j-1]$ ($L[i+1, j-1]$) is the left(right) neighbor.

**Definition 1** (Lattice Diagram)**.** *(LD) for a single output functions is represented by a Matrix L in which,*

1. *The root node of the diagram is* $L[1, 1]$ *corresponds to the output of the lattice.*
2. *Non-zero entries* $L[i, j]$ *realize a logic function of the expansion variable and of the right and left predecessors.*
3. *Every terminal node has no logical predecessor and every non-terminal node has one or two logical predecessors and successors.*
4. *Every node without the right successor is an output node.*
5. *Every non-output (leaf) node provides its output to one or both of its successors hence creating connections in a regular manner to its successors in the upper level.*
6. *For every leaf node there exists a logic path to the output.*
7. *All other entries that do not represent logic nodes in the matrix have value 0 and can be eliminated from the network of logic circuit.*

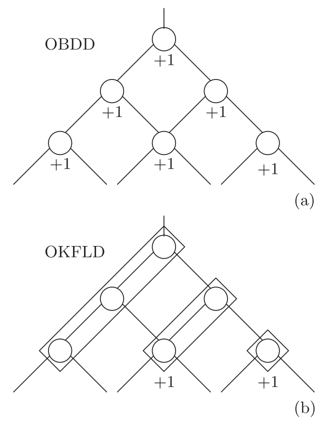



**Fig. 2** Comparison of required ancilla bits in a OBDD (a) and in a OKFLD (b).

**Definition 2** (Ordered Lattice Diagram)**.** *An Ordered Lattice Diagram is a Lattice Diagram in which there is at most one variable on a diagonal.*

**Definition 3** (Ordered Kronecker Functional Lattice Diagram)**.** *(OKFLD) is an ordered LD over* $X_n$ *together with a uniquely determined decomposition type* $d_i$ *(Eqs. (1) - (3)) assigned to each variable* $x_i, (i \in \{1, \ldots, n\})$*. The function* $f_G : B^n \to B$ *represented by an OKFLD, G, over* $X_n$ *is defined as:*

1. *If G consists of a single node labeled with* 0(1)*, then G is an OKFLD for* $f = 0 (f = 1)$.
2. *If G has a root v with label* $x_i$*, then G is an OKFLD for*

$$\begin{cases} \overline{x_i} f_{low(v)} \oplus x_i f_{high(v)} & : d_i \text{ is Shannon (S)} \\ f_{low(v)} \oplus x_i f_{high(v)} & : d_i \text{ is positive Davio (pD)} \\ f_{low(v)} \oplus \overline{x_i} f_{high(v)} & : d_i \text{ is negative Davio (nD)} \end{cases} \tag{4}$$

*Where* $f_{low(v)}(f_{high(v)})$ *are the functions represented by the OKFLD rooted* $low(v)(high(v))$.

Further details on different instances of Lattice Diagrams can be found in [18]–[20].

The main advantage of OKFLD over OBDD (Ordered binary decision diagrams) is in the ability to reduce the number of ancilla bits. As reported in [25] the OBDD based synthesis of reversible circuits requires one ancilla bit per node of BDD. The OKFLD analyzed in [22] showed that due to the usage of the positive Davio expansion and the Lattice structure requires however only one ancilla bit per layer of OKFLD (Fig. 2)! Also in OBDD each node uses Shannon expansion, the various different OKFLDs permits to replace each node by different expansion and thus simplify even more the internal circuit wiring.

In this paper we concentrate on KFLDs that use only positive Davio for node expansion. Consequently from now on all references to OKFLD, expansions or rules of simplification are intended for OKFLDs that use only positive Davio expansion.

## 3. Creating a Kronecker Functional Lattice Diagram (KFLD)

A KFLD is created by performing a level-by-level expansion of the function represented by the root node. The root node is expanded first using the pD expansion to create two child nodes. Next for all nodes at the same level, cofactors of the nodes are created again using pD expansion. Joining operations are performed on some cofactors (geometric neighbors) to create a combined node. The non-joined cofactors are converted to nodes.

Figure 3 shows positive Davio joining operations that can be performed on any two cofactor-nodes $y$ and $z$ of geometric neighbor nodes $r$ and $s$ when both cofactors are nonconstant. Unlike OBDDs and OKFDDs, the joining operations in OKFLDs can also be applied on the non-isomorphic nodes. The process of node expansion and joining operation are continued until all nodes terminate with constant values.

On geometric neighbor nodes with isomorphic cofactor-nodes the joining operation result in simple nodes where the expansion variable is not propagated to the next levels. This is shown in Fig. 4.

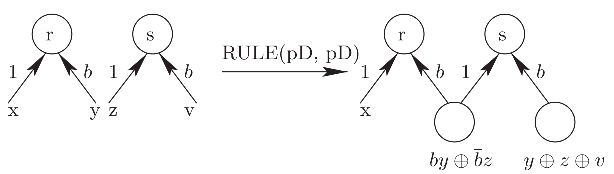



**Fig. 3** Positive Davio (pD, pD) joining rules used for joining non-isomorphic nodes in KFLDs.

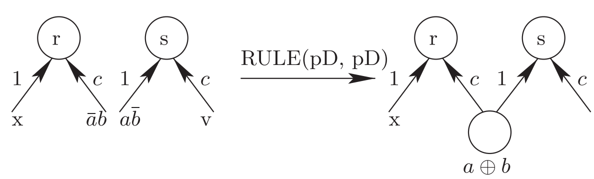



**Fig. 4** Positive Davio (pD, pD) joining rules used for joining isomorphic nodes in KFLDs.

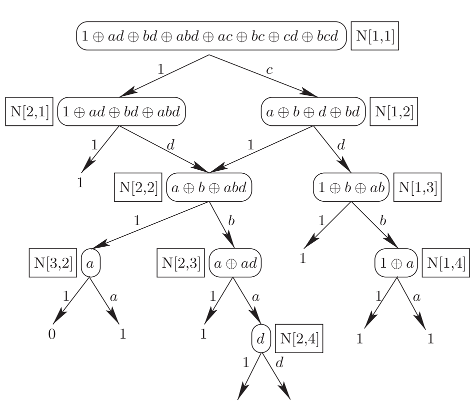



**Fig. 5** KFLD created with Positive Davio gate for function $f = 1 \oplus ad \oplus bd \oplus abd \oplus ac \oplus bc \oplus cd \oplus bdc$.

**Example 1** (Creating KFLD). *Let a function be defined by a Positive Polarity Reed Muller form shown in Eq. (5).*

$$f = 1 \oplus ad \oplus bd \oplus abd \oplus ac \oplus bc \oplus cd \oplus bdc \tag{5}$$

*The creation of the KFLD uses the following steps:*

1. *Variable $c$ is selected for expansion of the root node in the first level to create second level nodes.*
   - *The left node resulting from the pD expansion is thus $f_{\bar{c}} = 1 \oplus ad \oplus bd \oplus abd$*
   - *The right node resulting from pD expansion is given by $f_c \oplus f_{\bar{c}} = a \oplus b \oplus d \oplus bd$ with $f_c = 1 \oplus ad \oplus abd \oplus a \oplus b \oplus d$*
2. *For the second level of expansions the variable $d$ is selected.*
   - *The right co-factor of the node $f_{\bar{c}} = 1 \oplus ad \oplus bd \oplus abd$ is given by $a \oplus b \oplus ab$.*
   - *The left co-factor of the node $a \oplus b \oplus d \oplus bd$ with respect to variable $d$ is $a \oplus b$.*
3. *The joining operation on this cofactor is computed as shown in Eq. (3)*

$$\begin{aligned} & d(a \oplus b \oplus ab) \oplus \bar{d}(a \oplus b) \\ & \quad = da \oplus bd \oplus dab \oplus \bar{d}a \oplus \bar{d}b \\ & \quad = a \oplus b \oplus abd \end{aligned} \tag{6}$$

*Which is the left node on the third level in Fig. 3. The KFLD is completed by applying similar steps to all nodes. and the final KFLD is shown in Fig. 5.*

## 4. The KFLD Algorithm for Reversible Circuits

### 4.1 Logic Expansion Mapping

The KFLD nodes can be expanded using three expansions (Eqs. (1) - (3)). Each of the expansions can be mapped directly to a particular reversible gate. In this paper we only use the positive Davio expansion and Davio expansion can be directly mapped to a Toffoli gate:

- Positive Davio Cell. It is mapped directly to a Toffoli gate as shown in Fig. 6. The inputs $a$ and $b$ of the positive Davio gate in Fig. 6 act as control qubits of the Toffoli gate and input $c$ acts as a target qubit of the Toffoli gate.

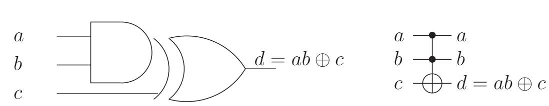



**Fig. 6** Representation of the positive Davio cell as a Toffoli gate and its reversible counterpart.

### 4.2 Lattice2QA Algorithm

Inputs to the algorithm are synthesized KFLD (Sect. 3) and functional output (root node) node of the KFLD. The output of the algorithm is a reversible logic circuit that consists of a cascade of Toffoli gates. The quantum circuit is created by, forming layers of cascades of gates. Every node in the KFLD is transformed into one of the three gates the Toffoli, the 2-qubit Feynman gates or the 1-qubit NOT gates which is the unique characteristic of our method. The nodes terminating with constant values are transformed to a Feynman gate, a NOT gate or a wire.

The algorithm starts by performing a preorder traversal of the KFLD to find every output of the quantum circuit. The building of the queue Q uses a recursive function shown in the pseudo code 1.

**Algorithm 1** The recursive function used to find all output nodes

```
1:  function NEXT(i, j)
2:      if L[i, j] == 0||L[i, j] == 1||L[i, j] ∈ V then
3:          Q ← {}
4:      else
5:          QL ←NEXT(i + 1, j)
6:          QR ←NEXT(i, j + 1)
7:          Q ← cat(QL, QR)
8:          V ← L[i, j]
9:          if E(L[i, j], L[i − 1, j]) == 0 then
10:             Q ← cat(L[i, j], Q)
11:         end if
12:     end if
13:     return Q
14: end function
```

The algorithm traverses the lattice starting from the root node $L[1, 1]$ top-down and from left to right. The algorithm maintains a list of previously visited nodes $V$ and recursively populates the queue $Q$. Each visited node is first checked whether it is a constant or if it has already been visited (line 2). For any non visited and non constant node the algorithm first searches the left predecessor and then the right predecessor (lines 5 and 6). The resulting queues from left and right predecessors are concatenated (line 7) and the current node is added to the list of visited nodes $V$ (line 8). Finally in lines 9 and 10 the current node is checked if it has the right successor and if not it is added to the queue $Q$.

**Example 2.** *Consider the KFLD shown in the Fig. 5 with the functional output f representing the root node. For convenience all nodes of the KFLD are labeled as shown in the Fig. 5 (square blocks). Using the recursive algorithm from the pseudo code 1 the obtained queue of output nodes is* $Q = \{N[1, 1], N[2, 3], N[2, 4], N[1, 2], N[1, 3], N[1, 4]\}$

Each node is then transformed to one particular gate depending on its predecessors and successors. If any of the predecessors is constant the Node will be transformed into a Feynman gate otherwise the node is transformed into a

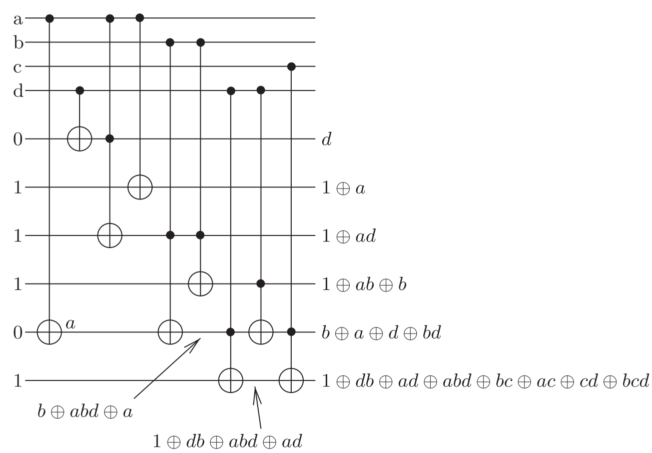



**Fig. 7** Quantum circuit for the positive Davio Lattice from Fig. 5.

Toffoli gate. A Toffoli gate in the circuit receives one control input from the variable that was used in expansion and another control input from the output of the gate one layer above. This rule is invariably true for all Toffoli gates in the created circuit. The Feynman gates in the circuit receive input from the variable used for expansion of the same node. Using these transformation rules, a layer of cascade of reversible gates is created by traversing left for every output node in the queue $Q$.

**Example 3.**
- *N[1, 1]: as the right predecessor N[1,2] is not constant the* $3 \times 3$ *Toffoli gate is used. The output of N[2, 1], the output of N[1, 2] and the expansion variable c represent the target and the two control bits respectively. This is shown as the rightmost gate in Fig. 7 (bottom layer).*
- *N[2, 1]: as the left predecessor is constant 1, no further node needs to be explored by traversing left and constant 1 will act as the target input for a Toffoli gate. The two control inputs of N[2, 1] are the expansion variable d and the output signal of the node N[2,2]. The node N[2, 1] represents the second gate in the bottom layer in Fig. 7. This completes the bottom layer of the quantum circuit represented by the KFLD of Fig. 5.*
- *Other layers of cascade of Toffoli gates are completed in a similar fashion in order to complete the final circuit. The final quantum circuit is shown in Fig. 7.*

Observe that each gate is either a 3-qubit Toffoli gate or a 2-qubit Feynman gate and thus Toffoli gates with many inputs characteristic to most contemporary algorithms [1], [15] are entirely avoided.

## 5. Quantum Circuit Optimization by Creating Efficient KFLD

As was illustrated in Sect. 3 (Figs. 3 and 4) the merging of geometric non-isomorphic neighbor nodes reintroduces into the lattice variables used in nodes expansion. This causes repetition of variables in the subsequent stages, which increase number of nodes and size of the KFLDs. The repeti-

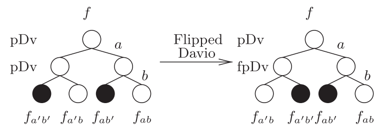



**Fig. 8** Flipped positive davio node.

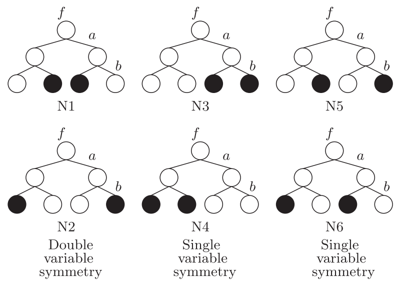



**Fig. 9** Representation of six symmetries.

tion of variables in KFLDs is greatly influenced by the order of used expansion variables in the creation of the KFLDs; different order of expansion variables will create different number of isomorphic nodes on each level and will result in KFLD with different number of levels.

Hence an efficient selection of variable order is essential to create optimum KFLD and respective quantum circuit. In this paper we explore two distinct methods that minimize variable repetition by searching for optimum variable order selection.

Moreover notice that the variable repetition converges (any function can be represented by a KFLD with a finite number of levels) because the reintroduced expansion variables can be in the worst case used to expand the nodes into constant values (Fig. 5 variable $d$).

### 5.1 Adjacent Isomorphic Nodes Replacement

One of the simplest heuristics for variable ordering introduced in [17], [21] is to replace adjacent isomorphic (symmetric function) nodes by a single node. The best variable order allows to merge the largest amount of isomorphic nodes. Additionally to increase the probability of having adjacent isomorphic nodes, a flip Davio operation (Fig. 8) is performed[†].

For any pair of variables $x_i$ and $x_j$ there are four cofactors, $fx_ix_j$, $fx_i'x_j$, $fx_ix_j'$, $fx_i'x_j'$. The function is symmetric in these two variables if any two of the four cofactors are equivalent. Symmetry in variables can be used by negation of any one of the variables. Symmetry created by negation of any one variable is called skewed-symmetry. For clarity we show in Fig. 9 the six possible symmetries introduced in [24]. These symmetry rules are used in creating optimum KFLD and to minimize the quantum cost of the circuit.

To search for geometric symmetries we use the window permutation algorithm [21]. This algorithm proceeds by selecting a level (and repeated for every level) $i$ in the KFLD and exhaustively searching all $k!$ permutations of the $k$ adjacent variables starting at level $i$. This is done by selecting $k!-1$ pair wise exchanges followed by up to $k(k!-1)/2$ pair wise exchanges to restore the best permutation obtained during the process. Figure 10 shows the variable permutations which are explored when applying a window of size $k = 3$ starting at variable $b$. Total five permutations are explored with four adjacent variable swaps, then three additional variable swaps are used to restore the best permutation. The window permutation algorithm is practical for functions up to five variables.

[†]Similar to the flip Shannon operation for PSBDD presented in [24].

| | |
|---|---|
| a,b,c,d,e | Initial |
| a,c,b,d,e | swap(b,c) |
| a,c,d,b,e | swap(b,d) |
| a,d,c,b,e | swap(d,c) |
| a,d,b,c,e | swap(c,b) |
| a,b,d,c,e | swap(d,b) |
| a,b,c,d,e | swap(b,d) |
| a,c,b,d,e | swap(b,c) |
| a,c,d,b,e | swap(b,d) |

**Fig. 10** Example of a window permutation algorithm.

| | |
|---|---|
| a,b,c,d,e | Initial |
| a,c,b,d,e | swap(b,c) |
| a,c,d,b,e | swap(b,d) |
| a,d,c,b,e | swap(d,c) |
| a,d,b,c,e | swap(c,b) |
| a,b,d,c,e | swap(d,b) |
| a,b,c,d,e | swap(b,d) |
| a,c,b,d,e | swap(b,c) |
| a,c,d,b,e | swap(b,d) |

**Fig. 11** Example of a sifting algorithm.

### 5.2 Sifting Algorithm

To optimize further the KFLD the Sifting algorithm presented in [21] and originally intended for the OBDD minimization was also used. Th sifting algorithm searches for the best position of a variable by moving one variable from level to level while keeping all other variables on a fixed position. For each variable in the KFLD, one selected variable is swapped with its successor until becoming the next to last variable. Applying this to other variables, the best variable order is stored and the variables are placed in their respective optimal position. An example of Sifting algorithm is shown in Fig. 11.

## 6. Experimental Results

Two programs were created and are used for the experimentation. The program *Lattices* creates KFLD for a given Boolean function, and the *Lattice2QA* creates a quantum circuit from a KFLD. The programs are implemented in C++ and the experiments were done on a Intel 2.4GHz Core2 Duo processor with 2GB of memory.

To evaluate the performance of our approach we compared the result with four different algorithms respectively introduced in [1], [15], [25] and [16]. The reason for selecting these four algorithms is the similarity of the approach in the case of [25], the latest algorithm and the top of the state of art [16] and two well known algorithms [15] and [1]. The different algorithms do not always use the same benchmark functions for evaluations the results are presented in two distinct tables.

Table 1 shows comparison of results between the KFLD and the method from [16] and Table 2 compares our KFLD with algorithms from [1], [15] and [25]. The evaluation counts the number of gates used to built the circuit and the quantum costs of the reversible gates are computed using the method used in the contemporary CAD algorithms [11], [12], [15].

Table 1 shows the name of the function benchmark, the number of gates (G) and the quantum cost (C) for the algorithm from [16] and the number of gates and the quantum cost for the KFLD in columns one, two, three, four and five respectively.

The column one in Table 2 shows the name of the benchmark function, column two (P.I.) shows total number of real inputs and column three (G.I.) depicts number of ancilla bits added to the final circuit created by Lattice2QA. Run time for the algorithm is marked on column four (CPU). For each compared algorithm two columns (G)

**Table 1** Comparison of the results with algorithm from [16].

| Fu. | Algo. [16] | | KFLD | |
|---|---|---|---|---|
| | #G | C | #G | C |
| 5xp1 | 58 | 786 | 123 | 379 |
| cu | 28 | 781 | 248 | 872 |
| dc1 | 31 | 127 | 45 | 129 |
| dc2 | 51 | 1084 | 365 | 813 |
| ham7 | 37 | 67 | 22 | 58 |
| decode | 89 | 399 | 124 | 364 |
| f2 | 14 | 112 | 31 | 91 |
| root | 48 | 1811 | 398 | 1398 |
| sqr6 | 54 | 583 | 78 | 367 |
| wim | 23 | 139 | 30 | 84 |
| z4ml | 34 | 489 | 79 | 331 |
| inc | 75 | 892 | 158 | 758 |
| misex1 | 42 | 332 | 127 | 621 |
| mlp4 | 80 | 2496 | 509 | 2028 |
| bw | 287 | 637 | 168 | 504 |
| apla | 72 | 1683 | 605 | 2001 |
| cm42a | 42 | 161 | 37 | 121 |
| c7552 | 89 | 399 | 136 | 424 |
| dk17 | 34 | 1014 | 388 | 1228 |

**Table 2** Comparison of the results of synthesis algorithms.

| Benchmarks | #F.I | #G.I | CPU | #G [15] | C [15] | #G [1] | C [1] | #G [25] | C [25] | # G.SS | C.SS | G.R | C.R |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| pprm1 | 4 | 4 | <0.01 | NA | NA | NA | NA | NA | NA | 9 | 33 | 14 | 46 |
| pprm2 | 10 | 6 | 0.50 | NA | NA | NA | NA | NA | NA | 51 | 223 | 100 | 419 |
| pprm3 | 15 | 12 | 0.50 | NA | NA | NA | NA | NA | NA | 23 | 510 | 43 | 1005 |
| xnor5 | 5 | 1 | <0.01 | NA | NA | NA | NA | NA | NA | 5 | 5 | 8 | 8 |
| Cycle17_3 | 20 | 10 | 40.1 | 48 | 6057 | NA | NA | NA | NA | 920 | 4160 | 1820 | 8220 |
| 5bitadder | 10 | 5 | <0.01 | 29 | 55 | NA | NA | NA | NA | 29 | 55 | 29 | 55 |
| 8bitadder | 16 | 8 | 0.10 | 122 | 322 | NA | NA | NA | NA | 122 | 322 | 122 | 322 |
| $^{nth}$Prime3inc | 3 | 4 | <0.01 | 4 | 6 | NA | NA | NA | NA | 4 | 6 | 7 | 9 |
| $^{nth}$Prime4inc | 4 | 5 | <0.01 | 12 | 58 | NA | NA | NA | NA | 16 | 48 | 28 | 76 |
| $^{nth}$Prime5inc | 5 | 5 | 0.22 | 26 | 78 | NA | NA | NA | NA | 25 | 83 | 45 | 143 |
| $^{nth}$Prime6inc | 6 | 6 | 0.36 | 55 | 667 | NA | NA | NA | NA | 148 | 586 | 290 | 1142 |
| 2to5 | 5 | 4 | 0.12 | 15 | 107 | 20 | 100 | NA | NA | 30 | 106 | 55 | 181 |
| rd32 | 3 | 1 | <0.01 | 4 | 8 | 4 | 8 | NA | NA | 4 | 8 | 6 | 10 |
| 3_17 | 3 | 1 | <0.01 | 6 | 12 | 6 | 14 | NA | NA | 8 | 15 | 13 | 20 |
| 5mod5 | 5 | 1 | <0.01 | 10 | 90 | 11 | 91 | NA | NA | 14 | 58 | 23 | 91 |
| ham3 | 3 | 0 | <0.01 | 5 | 7 | 5 | 9 | NA | NA | 3 | 7 | 5 | 9 |
| xor20 | 20 | 0 | <0.01 | 19 | 19 | 19 | 19 | NA | NA | 19 | 19 | 37 | 37 |
| Graycode6 | 6 | 5 | <0.01 | 5 | 5 | 5 | 5 | NA | NA | 5 | 5 | 9 | 9 |
| Graycode10 | 10 | 9 | <0.01 | 9 | 9 | 9 | 9 | NA | NA | 9 | 9 | 17 | 17 |
| Graycode20 | 20 | 19 | <0.01 | 19 | 19 | 19 | 19 | NA | NA | 19 | 19 | 37 | 37 |
| 4_49 | 4 | 4 | 0.04 | 16 | 52 | 13 | 61 | NA | NA | 16 | 52 | 28 | 84 |
| hwb4 | 4 | 4 | <0.01 | 17 | 36 | 15 | 35 | NA | NA | 12 | 28 | 20 | 36 |
| 6sym | 11 | 4 | 0.37 | 20 | 62 | NA | NA | 29 | 69 | 17 | 69 | 28 | 108 |
| 9sym | 15 | 5 | 0.40 | 28 | 94 | NA | NA | 62 | 153 | 21 | 94 | 37 | 143 |
| Cycle10_2 | 12 | 6 | 27.9 | 19 | 1198 | NA | NA | 78 | 164 | 171 | 831 | 330 | 1602 |
| ham15 | 15 | 9 | 0.10 | 109 | 206 | NA | NA | 153 | 246 | 46 | 190 | 77 | 306 |
| hwb5 | 5 | 5 | 1.2 | 24 | 104 | NA | NA | 88 | 205 | 24 | 96 | 43 | 167 |
| hwb6 | 6 | 6 | 2.0 | 42 | 140 | NA | NA | 159 | 375 | 32 | 128 | 54 | 226 |
| hwb7 | 7 | 6 | 0.10 | 35 | 203 | NA | NA | 281 | 653 | 49 | 185 | 90 | 335 |
| rd84 | 8 | 7 | <0.01 | 28 | 98 | NA | NA | 104 | 304 | 20 | 68 | 37 | 121 |
| ham15 | 15 | 9 | 0.10 | 109 | 206 | NA | NA | 153 | 246 | 46 | 190 | 77 | 306 |
| Decode24 | 4 | 2 | <0.01 | NA | NA | 11 | 31 | 11 | 23 | 10 | 30 | 18 | 40 |
| Alu | 5 | 2 | <0.01 | NA | NA | 18 | 114 | 9 | 22 | 5 | 17 | 8 | 24 |
| ham7 | 7 | 5 | 0.10 | 23 | 81 | 24 | 68 | 61 | 107 | 22 | 58 | 37 | 81 |
| 4mod5 | 4 | 1 | <0.01 | 5 | 13 | 5 | 13 | 8 | 18 | 6 | 18 | 9 | 23 |
| rd53 | 5 | 5 | <0.01 | 16 | 75 | 13 | 116 | 34 | 75 | 11 | 39 | 17 | 53 |
| xor5 | 5 | 0 | <0.01 | 4 | 4 | 4 | 4 | 8 | 8 | 4 | 4 | 7 | 7 |

**Table 3** Summary of the evaluation results.

| Algo. | Min | Max | Avg | TAvg | RAvg | RTAvg |
|---|---|---|---|---|---|---|
| [15] | −187% | 52% | 25.3% | 9.6% | 0.03% | −47.5% |
| [1] | −38.14% | 85% | 32.8% | 12.5% | 0.11% | −36.1% |
| [25] | −407% | 77.8% | 43.1% | 2.48% | 0.45% | −62.04% |
| [16] | −87% | 51.8% | 25.3% | 9.6% | | |
| Mean | | | 31.7% | 8.5% | 0.19% | −48.54% |

and (C) shows respectively the number of gates (G) in the circuit and the quantum cost (C) in each of the three algorithms evaluated [1], [15], [25] are presented in columns five to ten. Finally, columns 13 and 14 shows the result of our algorithm when using both the symmetry as well as sifting optimization. Finally the two last columns show the number of gates and the quantum cost when the garbage/variable lines are restored to initial value.

The results of evaluations are summarized in Table 3. First column indicates the algorithm of comparison. The second column shows the worst case, i.e. the case where KFLD performed worst from all tested benchmark. The negative percent means how much more costly the circuit obtained by the KFLD was. The third column shows the best case of cost decrease. Column four shows the average of quantum cost when only benchmark functions for which KFLD obtained better (less costly) quantum circuits have been obtained. The fifth column shows the average of the improvement of the quantum cost using benchmark functions where the algorithm has been tested. Finally the two last columns show the average over only the best circuit and the average over all tested benchmark functions when the KFLD was using the variable qubit restoration. All positive percentages shows that KFLD was able to improve the cost of the tested function benchmarks.

As can be seen our algorithm was able to improve the cost of synthesized benchmarks when compared to all four algorithms on average by 8.5% and on the benchmarks where our algorithm generated less costly circuits the average cost improvement was 31.7%.

Notice that the results of the KFLD with garbage/variable bit restoration show quite negative scores; this is a natural consequence because none of the algorithms evaluated do not use the bit variable restoration. However, in order to design circuits that can be potentially used in quantum algorithms the variable bits must be restored and thus the provided results indicate an estimate on the real cost and size of circuits with such requirements.

## 7. Conclusions

We proposed a new approach to synthesize reversible and quantum circuits based on mapping the Kronecker Functional Lattice Diagram directly to a quantum circuit. When quantum technology such as Ion Trap [3] is used, minimizing the quantum cost is what really counts, not the gate cost [13] and consequently our method is more efficient. Moreover, the circuit created by our tool is always regular and can be mapped to an array of Ion trap [14] realized quantum gates. It can be also mapped with some modification to a one-dimensional array, satisfying the so-called LNNM (Linear Nearest Neighbor Model). This is a subject of further research of our group.

As future work several topics are to be studied. The variable ordering problem, the reduction of ancilla bits added during the creation of KLFD, extension to novel layouts of quantum technologies and the study of the usage of other expansions in the nodes of the KFLD. Moreover the study of more general form of KFLDs using negative controls as well as KFLDs with different nodes expansions are also to be considered.

### References


[1] A. Agrawal and N.K. Jha, "Synthesis of reversible logic," Proc. DATE, pp.710–722, 2004.

[2] S.B. Akers, "A rectangular logic array," IEEE Trans. Comput., vol.C-21, pp.848–857, 1972.

[3] J.I. Cirac and P. Zoller, "Quantum computation with cold trapped ions," Phys. Rev. Lett., vol.74, no.20, p.4091, 1995.

[4] D. Deutsch and R. Jozsa, "Rapid solution of problems by quantum computation," Proc. Royal Society of London, vol.A400, pp.73–90, 1992.

[5] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M. Perkowski, "Efficient representation and manipulation of switching functions based on ordered kronecker functional decision diagrams," Proc. Design Automation Conference, pp.415–419, 1994.

[6] K. Fazel, M.A. Thornton, and J.E. Rice, "ESOP-based Toffoli gate cascade generation," IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, pp.206–209, 2007.

[7] L.K. Grover, "A fast quantum mechanical algorithm for database search," arXiv:quant-ph/9605043, 1996.

[8] U. Kebschull, E.E. Schubert, and W. Rosenstiel, "Multilevel logic based on functional decision diagrams," Proc. European Design Automation Conferance, pp.43–47, 1992.

[9] S. Lloyd, "Universal quantum simulators," Science, vol.273, pp.1073–1078, 1996.

[10] M. Lukac, M. Kameyama, M. Perkowski, and P. Kerntopf, "Decomposition of reversible logic function based on cube-reordering," Facta Universitatis, vol.24, no.3, pp.403–422, 2011.

[11] M. Lukac, M. Perkowski, H. Goi, M. Pivtoraiko, C.H. Yu, K. Chung, H. Jee, B.-G. Kim, and Y.-D. Kim, "Evolutionary approach to quantum reversible circuit synthesis," Artif. Intell. Review., vol.20, no.3-4, pp.361–417, 2003.

[12] D. Maslov, G.W. Dueck, and D.M. Miller, "Techniques for the synthesis of reversible Toffoli networks," ACM Trans. Des. Autom. Electron. Syst., vol.12, no.4, article 41, 2007.

[13] D. Maslov and G.W. Dueck, "Improved quantum cost for n-bit Toffoli gates," Electron. Lett., vol.39, no.25, pp.1790–1791, 2003.

[14] T.S. Metodi, A.I. Faruque, and F.T. Chong, Quantum Computing for Computer Architects, Second ed., vol.6, 2011.

[15] D.M. Miller, D. Maslov, and G.W. Dueck, "A transformation based algorithm for reversible logic synthesis," Proc. DAC, 2003.

[16] N.M. Nayeem and J.E. Rice, "A shared-cube approach to esop-based synthesis of reversible logic," Facta universitatis-series: Electronics and Energetics, vol.24, no.3, pp.385–402, 2011.

[17] S. Panda, F. Somenzi, and B.F. Plessier, "Symmetry detection and dynamic variable ordering of decision diagrams," Proc. DAC, pp.628–631, 1994.

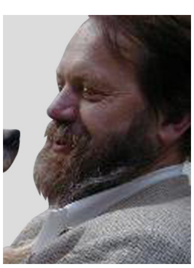
[18] M. Perkowski, L. Jozwiak, and R. Drechsler, "A canonical and/exor form that includes both the generalized reed-muller forms and kronecker reed-muller forms," Proc. Reed-Muller 1997 Conference, pp.219–233, 1997.

[19] M. Perkowski, L. Jozwiak, R. Drechsler, and B. Falkowski, "Or-

dered and shared, linearly independent, variable-pair decision diagrams," Proc. First International Conference on Information, Communications and Signal Processing, 1997.
[20] M.A. Perkowski, M. Chrzanowska-Jeske, and Y. Xu, "Lattice diagrams using reed-muller logic," IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design, pp.85–102, 1997.
[21] R. Rudell, "Dynami variable ordering for ordered binary decision diagrams," Proc. ICCAD, pp.42–47, 1993.
[22] D. Shah and M.A. Perkowski, "Synthesis of quantum arrays with low quantum costs from kronecker functional lattice diagrams," IEEE Congress on Evolutionary Computation, pp.1–7, 2010.
[23] P.W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," Proc. 35th Annual Symposium on Foundations of Computer Science (Shafi Goldwasser, ed.), pp.124–134, 1994.
[24] W. Wang and M. Chrzanowska-Jeske, "A global approach to the variable ordering problem in psbdds," Proc. IEEE International Symposium on Circuits and Systems, ISCAS, pp.117–120, 2001.
[25] R. Wille, D. Große, D.M. Miller, and R. Dreschler, "Equivalence checking of reversible circuits," Proc. 39th International Symposium on Multi-Valued Logic, pp.324–330, 2009.

**Martin Lukac** received PhD degree in Computer Engineering from Electrical and Computer Engineering Department at Portland State University. He obtained M.Sc degree in Brain Sciences from Polytechnic School, Paris France.Since 2009 he is working at Graduate School of Information Sciences, Tohoku University, Sendai, Japan as assistant professor. His research interests are Quantum Computing, Reversible Computing, Emergent technologies, Intelligent Robotics and Image Processing.

**Dipal Shah** received PhD degree in Computer Engineering from Electrical and Computer Engineering Department at Portland State University. He obtained M.S. degree in Electrical and Computer Engineering from Portland State University. Dipal is currently working at Intel Corporation in Austin, Texas, USA. Prior to joining Intel he acquired various positions at Motorola Inc and Mentor Graphics Corporation.



**Marek Perkowski** received his PhD from Technical University of Warsaw, Poland in 1980. Since 1984 he is professor at Portland State University where his research interests cover quantum computing, reversible computing, emerging technologies, VLSI design and intelligent robotics.

**Michitaka Kameyama** received the B.E., M.E. and D.E. degrees in Electronic Engineering from Tohoku University, Sendai, Japan, in 1973, 1975, and 1978, respectively. He is currently Dean and Professor in the Graduate School of Information Sciences, Tohoku University. His general research interests are intelligent integrated systems for real-world applications and robotics, advanced VLSI architecture, and new-concept VLSI including multiple-valued VLSI computing. Dr. Kameyama received the Outstanding Paper Awards at the 1984, 1985, 1987 and 1989 IEEE International Symposiums on Multiple-Valued Logic, the Technically Excellent Award from the Society of Instrument and Control Engineers of Japan in 1986, the Outstanding Transactions Paper Award from the IEICE in 1989, the Technically Excellent Award from the Robotics Society of Japan in 1990, and the Special Award at the 9th LSI Design of the Year in 2002. He is IEEE Fellow and IPSJ Fellow.