Re-implementation of a Divide and Conquer method for ESOP synthesis

1st Yu-Ching, Huang
GIEE NTU
Taipei, Taiwan
r09921049@ntu.edu.tw

***Abstract*—Quantum computer is believed to be able to solve certain computational problems substantially faster than classical computers. Today's rapid advances in fabrication of quantum computers call for robust synthesis methods that map conventional logic design into quantum architecture. Establishing a connection between such logic design and reversible network is therefore crucial, since quantum circuits are reversible. Hence, logic representations using Exclusive Sum of Product (ESOP) are advantageous because their natural affinity to reversible circuits. In this paper, we implement a *Divide and Conquer* methods together with a *BDD Extract* method that effectively and efficiently solve ESOP synthesis for circuits with more than 30 inputs, according to experimental results. Furthermore, we also implement a simplification technique which is able to reduce the size of ESOP expression up to 30%.**

I. Introduction

Quantum computing is the use of quantum phenomena such as superposition and entanglement to perform computation. Recent progress in fabrication makes the practical application of quantum computers a tangible prospect. For example, IBM's current largest quantum computer contains up to 65 qubits, and it plans to build one containing 1000 qubits by 2023.

Quantum computer is believed to be able to solve certain computational problems substantially faster than classical computers. It processes qubits instead of classical bits. A qubit can be in superposition and several qubits can be entangled. All operations on qubits besides measurement, called quantum gates, must be reversible. Therefore, Quantum circuits, which is an abstraction for the physical interaction with a quantum computer, differ significantly in comparison to classical circuits. It needs to be addressed by design automation tools.

This work focus on compiling algorithms composed of classical operations into a quantum circuits. The compilation procedure typically takes two steps: (1) the combinational operation is represented in terms of a reversible logic network, and (2) the reversible gates in the network are translated into quantum gates. Toffoli gates are usually used as gates in the reversible logic networks, since efficient methods that synthesis Toffoli gates into quantum gates exist.

More specifically, we target at ESOP (Exclusive Sum of Product) synthesis. And we call ESOP synthesis as the procedure of collapsing a combinational Boolean network into an ESOP expression. An ESOP expression is two-leveled AND/XOR logic expression that resembles well-known SOP expression except from that the terms (cubes) in the expression are XORed together instead of ORed. Since ESOP expression has natural affinity for Toffoli gates, the transform between them can be done directly and efficiently. In order to facilitate this relation, it is necessary to develop an efficient and effective algorithm for ESOP synthesis.

Therefore, in this paper we re-implement the methods presented in [1] with little modifications.

The rest of the paper is organized as follows. Section II details the proposed algorithm. Section III gives some application of the algorithm. Section IV shows our experimental results. And section V concludes this work.

II. Algorithm

We first introduce two previous state-of-the-art methods in ESOP synthesis. Then we present the new *divide and conquer* algorithm. Finally, the method of translating an ESOP expression into circuit of Toffoli gates is addressed.

A. Previous Methods

1) AIG Extract: The *AIG Extract* method convert a giving AIG network into a ESOP expression. It compute the ESOP of each node in topological order which start from input nodes and end in output nodes. The ESOP expression of each subsequent internal node is computed by conjoining the ESOP expressions of its previous nodes. Also, we should negate ESOP when there is a complemented edge. Adding a True term to the ESOP expression can easily attain it. The method is explained in the following equation.

$$e_i = \begin{cases} x_i & \text{if } i \text{ is an input node,} \\ e_{l_i} \wedge e_{r_i} & \text{otherwise,} \end{cases}$$

where i is the current AIG node, e_i denotes the ESOP expression of i, x_i is input variable, and l_i and r_i are the two fanin nodes of i.

For example, figure 1a is a simple AIG. Example 1 shows the ESOP expression of each nodes.

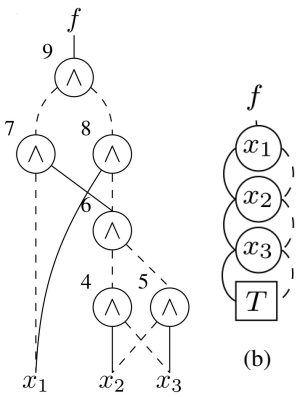

Fig. 1

Example 1:

$e_1 = x_1 \quad e_2 = x_2 \quad e_3 = x_3$

$e_4 = x_2\overline{x}_3$

$e_5 = \overline{x}_2 x_3$

$e_6 = x_2\overline{x}_3 \oplus \overline{x}_2 x_3$

$e_7 = \overline{x}_1 x_2\overline{x}_3 \oplus \overline{x}_1\overline{x}_2 x_3$

$e_8 = x_1 \oplus x_1 x_2\overline{x}_3 \oplus x_1\overline{x}_2 x_3$

$f = \overline{x}_1 x_2\overline{x}_3 \oplus \overline{x}_1\overline{x}_2 x_3 \oplus x_1 x_2\overline{x}_3 \oplus x_1\overline{x}_2 x_3 \oplus x_1$

We can easily see that *AIG Extract* method suffers from ESOP size explosion, which is the main drawback of this method.

2) BDD Extract: The alternative way is *BDD Extract*. The algorithm is presented in [3]. It convert a given Binary Decision Diagram (BDD) into an ESOP expression. It recursively computes the best expansion way for each node, obtaining a Pseudo-Kronecker Expression (PSDKRO). We briefly review the essential definitions of PSDKRO.

Let f_i^0 and f_i^1 denote the negative and positive cofactor of f with respect to x_i respectively, and $f_i^2 = f_i^0 \oplus f_i^1$ denotes Boolean difference of f with respect to x_i. A Boolean function $B^n \rightarrow B$ can be represented by any one of the following expansion.

$$f = \overline{x}_i f_i^0 \oplus x_i f_i^1 \qquad \textit{Shannon (S)}$$
$$f = f_i^0 \oplus x_i f_i^2 \qquad \textit{positive Davio (pD)}$$
$$f = f_i^1 \oplus \overline{x}_i f_i^2 \qquad \textit{negative Davio (nD)}$$

If we apply to a function f either three expansions, we get two subfunctions. For each subfunction, we can recursively apply any of the three expansions again until constant reaches. Then if we multiply out the resulting formula, we get a 2-level AND/XOR form, called a PSDKRO. The PSDKRO expression is in fact a subset of ESOP expression.

Example 2: Let $f(x_1, x_2, x_3) = x_1 + x_1 x_2$. If we first apply *S* to f, we get

$$f_1^0 = x_2 x_3 \quad \text{and} \quad f_1^1 = 1,$$

Then we decompose f_1^0 using *nD*, we get

$$(f_1^0)_2^1 = x_3 \quad \text{and} \quad (f_1^0)_2^2 = x_3,$$

We apply *pD* to $(f_1^0)_2^1$ or $(f_1^0)_2^2$, we get

$$((f_1^0)_2^1)_3^0 = 0 \quad \text{and} \quad ((f_1^0)_2^1)_3^2 = 1$$

Finally, we multiply out the expression and get

$$\begin{aligned} f &= \overline{x}_1(x_3 \oplus \overline{x}_2 x_3) \oplus x_1 \\ &= \overline{x}_1 x_3 \oplus \overline{x}_1\overline{x}_2 x_3 \oplus x_1, \end{aligned}$$

which is a PSDKRO expression as well as an ESOP expression.

Therefore, we want to find a minimum PSDKRO expression to expand f such that number of cubes and number of literals are minimized.

We introduce a new data structure for the *BDD extract* algorithm, called **Triplet Decision Diagram (TDD)**. It resembles well-known BDD, except from that each internal node in TDD has three children. Two of them are the same to BDD, i.e. postive cofactor and negative cofactor with respect to the decision variable of this node. The new one is Boolean difference w.r.t. the variable.

The methods traverse TDD twice. In the first pass, we construct TDD and calculate the best expansion of each node in terms of cube number of its ESOP expression. *Shannon*, *postive Davio*, and *negative Davio* expansion are considered for options. And in the second pass, we traverse through the constructed TDD and expand the formula according to computing results from the first pass. Finally, ESOP expression of the function f is derived.

Algorithm 1 explains the details of TDD constructing procedure, i.e. first pass of the *BDD Extract*. For a node n, we denotes the three children as $Then(n)$, $Else(n)$, and $Xor(n)$. Cost of n is represented by $n.cost$, and $n.type$ denotes the expansion type of this node among three possible expansions.

The cost $n.cost$ of a node n represents the number of cubes of the ESOP expression of n. By this procedure, we can derive the PSDKRO expression of a function with minimum number cubes, which is also a good ESOP expression.

We can already see from figure 1b, the derived ESOP expression is $x_1 \oplus x_2 \oplus x_3$. It performs much better than *Aig Extract* methods in this case and the majority of other benchmarks as well, extracting much more compact ESOP expression. However, BDD methods does not escape from its own shortcoming, i.e. lack of scalability. BDD construction suffers from memory explosion when number of input grows, and so does TDD in this work.

B. Divide and Conquer Method

Both state-of-the-art techniques fails to cope with the increasing complexity of the logic functions used on quantum computers. Therefore, we implement a *divide and conquer* approach proposed by [1] to solve it robustly and efficiently. We divide the problem into many smaller sub-problems, i.e. smaller function. The sub-problems are then independently solvable by *BDD Extract* methods. Finally, we combine the

Algorithm 1 *TDD_Construct* (n)

Input: n: current node
Output: c: cost of ESOP expression of this node
  if n == ZERO **then**
    return 0
  else if n == ONE **then**
    return 1
  else if $n.cost$ defined **then**
    return $n.cost$
  else
    c_t = *TDD_Construct*$(Then(n))$
    c_e = *TDD_Construct*$(Else(n))$
    c_x = *TDD_Construct*$(Xor(n))$
    if max(c_t, c_e, c_x) == c_t **then**
      $n.type = pD$
      $n.cost = c_e + c_x$
      Recursively_Delete$(Then(n))$
    else if max(c_t, c_e, c_x) == c_e **then**
      $n.type = nD$
      $n.cost = c_t + c_x$
      Recursively_Delete$(Else(n))$
    else
      $n.type = S$
      $n.cost = c_t + c_e$
      Recursively_Delete$(Xor(n))$
    return $n.cost$

solutions of these sub-problems together to form the final solution. Below we explain the algorithm.

We divide the main problem according to one relation.

Lemma 1: Any Boolean function f can be written as

$$f = f \wedge p_1 \oplus f \wedge p_2 \oplus \cdots \oplus f \wedge p_N \tag{1}$$

for N Boolean function p_1 to p_N over the same supports as f, if

$$p_1 \oplus p_2 \oplus \cdots \oplus p_N = 1 \tag{2}$$

It can be easily proved. We multiply f to both side of equation 2 and multiply out f on the left side, then equation 1 is obtained.

The key of this method lies in how to choose set $P = \{p_1, p_2, \ldots, p_N\}$ smartly, so that all individual $f \wedge p_i$ is easier to collapse.

One simple way is to limit P as a set of single-cube cofactors. Then $f \wedge p_i = f_{p_i} \wedge p_i$, where f_{p_i} is f cofactored by cube p_i. If we can derive the ESOP expression of each f_{p_i}, we can directly multiplied out $f_{p_i} \wedge p_i$ and obtained the ESOP expression of each term of f. And the ESOP of the original function f is simply the XOR of each term. As a result, now the sub-problem is back to ESOP synthesis for sub-function f_{p_i}. However, f_{p_i} would now be independent to all the variables in cube p_i, so it is a smaller function. If the sub-function is small enough, it is then solvable by *BDD Extract*. Below is an example of the *divide and conquer* method.

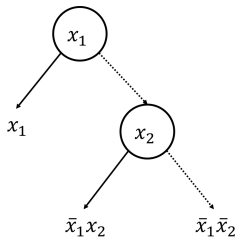

Fig. 2

Example 3: Let f be a Boolean function $B^n \to B$, where $n > 2$. By lemma 1, we can decompose f as

$$f = f \wedge p_1 \oplus f \wedge p_2 \oplus f \wedge p_3 \oplus f \wedge p_4$$

,where $p_1 = x_1 x_2$, $p_2 = x_1 \overline{x}_2$, $p_3 = \overline{x}_1 x_2$, and $p_3 = \overline{x}_1 \overline{x}_2$. Then f_{p_i} is a smaller function depending only on $n - 2$ variables.

The efficiency of the above technique is determined by how we choose P. Here we introduce a heuristic of selecting P. The idea is to minimize number of nodes in AIG network. First we temporarily cofactor each variable, one at a time. Doing each cofactor, we get two sub-circuits, i.e. postive cofactor and negative cofactor, and we count the total number of AIG nodes in the two circuits, called cofactor size. For the variable that have minimum cofactor size, we choose it to do cofactoring. The procedure is performed recursively to both of the two resulting sub-circuits until the number of supporting inputs of the current circuit is less than certain user-defined threshold, which we called it **support threshold**. In this work, **support threshold** is set to 20.

Figure 2 illustrates this method, where the resulting $P = \{x_1, \overline{x}_1 x_2, \overline{x}_1 \overline{x}_2\}$.

C. ESOP simplification

After ESOP synthesis procedure, an ESOP expression is obtained. We further implement ESOP simplification techniques. The simplification bases on two propositions.

Proposition 1: Two identical cubes, i.e. distance-0 cubes, can be removed or added into any ESOP together without changing the function of it.

Proposition 2: The Xor of two distance-1 cubes can be represented by a single cube.

Example 4: Below is some example of simplification using the two proposition

$$\begin{aligned} x_1 x_2 \oplus x_1 &= x_1 \overline{x}_2 \\ x_1 x_2 \oplus \overline{x}_1 \overline{x}_2 &= x_1 x_2 \oplus \overline{x}_1 \overline{x}_2 \oplus x_1 \overline{x}_2 \oplus x_1 \overline{x}_2 \\ &= x_1 x_2 \oplus x_1 \overline{x}_2 \oplus \overline{x}_1 \overline{x}_2 \oplus x_1 \overline{x}_2 \\ &= x_1 \oplus \overline{x}_2 \end{aligned}$$

III. Application

A. Reversible Circuits

In this section we introduce the relation between ESOP expression and reversible gates. Toffoli gates act on a number of variables (qubits). A n-input Toffoli gate has $n-1$ control

line and 1 target line. The output of each control line is the same to the input. The output of the target line is inverted if and only if all the polarity of inputs match the polarity specified for the gate.

We can build up a reversible circuit composed of Toffoli gates simply by concatenating Toffoli gates that act on the same target line. For a Boolean function of n inputs and m outputs, the resulting quantum circuit need only $n+m$ qubits.

Example 5: Let $f = \overline{x}_1\overline{x}_2x_3x_4x_5 \oplus \overline{x}_1x_2x_3\overline{x}_4 \oplus \overline{x}_1x_2x_4x_5 \oplus x_1\overline{x}_2x_4x_5 \oplus x_1x_2x_3 \oplus x_1x_2x_3x_4$, then a reversible circuit composed of Toffoli gates is depicted in Figure 3.

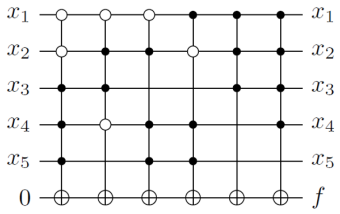

Fig. 3

The next step of quantum compilation is to translate the reversible circuit into quantum circuit. A small quantum circuit is desired. One heuristic to achieve it is to minimize the number of Toffoli gates and the number of control lines, which corresponds to the number of cubes and the number of literals in ESOP expression respectively. Therefore, the proposed ESOP synthesis algorithm which obtains a compact ESOP expression did have a good application and lead to compact quantum circuitst.

IV. Experiment Results

The algorithm is implemented and integrated in ABC. ABC is an open-source tool developed by Berkeley team, designed for logic synthesis, technology mapping, and formal verification for logic circuits. The CUDD package for BDD manipulating is also integrated in ABC. All the experiments were run on an Interl(R) Core(TM) i7-8550U CPU at 1.80 GHz, with 16GB RAM. Benchmarks are of BLIF format. After read in ABC, there were translated into AIGs by command "strash" and optimized by "dc2".

Table I shows the experimental results on a random control circuit "c432". **Support threshold** mentioned in section II-B is set to 20. The fist two columns describe the names and the numbers of supports of the POs. The next two column shows the statistics of resulting ESOP expression in our algorithm. The 5th and 6th columns are the statistics of the ESOP expression after doing simplification. And the last column gives the runtime of the implementation that finished ESOP synthesis and simplification. The results showed that the proposed algorithm performs efficiently. It solved output with more than 30 supporting input within few minutes. And the simplification techniques did reduce the ESOP expression effectively. For some output, it reduces number of cubes and literals up to 30%.

TABLE I

c432		ESOP		simplified ESOP		
PO	S #	Cube #	Literal #	Cube #	literal #	time (s)
G223gat	18	512	4609	511	4608	0.12
G329gat	27	5448	79378	3952	60855	7.05
G370gat	36	110176	2267008	85739	1811954	773.83
G421gat	36	31414	685194	25831	578819	127.39
G430gat	36	91969	1910788	69089	1468765	544.47
G431gat	36	84303	1748731	65682	1382563	470.6
G432gat	36	76912	1565663	59103	1235378	400.34

Table II shows the results on a 16-bit adder. The methods perform not so good at arithmetic circuits. When the support number of output increases, both cube number and literal number increases significantly, as well as runtime. And simplification technique did not do well.

TABLE II

16-adder		ESOP		simplified ESOP		
PO	S #	Cube #	Literal #	Cube #	literal #	time (s)
s10	23	2569	50470	2567	50460	5.28
s11	25	5142	111230	5138	111202	17.08
s12	27	10288	243042	10280	242970	28.63
s13	29	20580	527250	20564	527074	70.62
s14	31	41164	1136834	41132	1135418	211.99
s15	33	82332	2438338	82268	2437378	694.42

If we take **support threshold** = ∞, that is, we do not use the *divide and conquer* method, doing **BDD Extract** only, then all the outputs with number of supports greater than 27 failed to synthesis their ESOP expressions within an hour.

V. Conclusion

ESOP expression is a crucial representation that bridges the gap between combinational Boolean circuits and reversible circuits. It is hence promising in synthesis traditional circuits into quantum circuits. In this work, we implement a *divide and conquer* method and a *BDD Extract* method together which decently solve ESOP synthesis of circuit under 40 supporting inputs. Furthermore, we implement an ESOP simplification technique that effectively reduce the size of ESOP expression in terms of cube number and literal number. Experimental results showed that the method is effectively and efficiently in solving control circuits, and acceptable in solving arithmetic circuits.

References

[1] B. Schmitt, M. Soeken, G. De Micheli and A. Mishchenko, "Scaling-up ESOP Synthesis for Quantum Compilation," 2019 IEEE 49th International Symposium on Multiple-Valued Logic (ISMVL), Fredericton, NB, Canada, 2019, pp. 13-18, doi: 10.1109/ISMVL.2019.00011.

[2] Soeken, Mathias, et al. "Logic synthesis for quantum computing." arXiv preprint arXiv:1706.02721 (2017).

[3] R. Drechsler, "Pseudo-Kronecker expressions for symmetric functions," in IEEE Transactions on Computers, vol. 48, no. 9, pp. 987-990, Sept. 1999, doi: 10.1109/12.795226.

[4] Mishchenko, Alan, and Marek Perkowski. "Fast heuristic minimization of exclusive-sums-of-products." (2001).