ESOP-based Toffoli Gate Cascade Generation

K. Fazel, M. A. Thornton
Dept. of Computer Science and Engineering
Southern Methodist University
Dallas, TX, USA
{kfazel, mitch}@engr.smu.edu

J. E. Rice
Dept. of Mathematics & Computer Science
University of Lethbridge
Lethbridge, AB, Canada
j.rice@uleth.ca

***Abstract*—An ESOP-based Toffoli gate cascade synthesis algorithm is presented. The algorithm is capable of generating a cascade of reversible gates for logic functions with large numbers of qubits. The algorithm is fast as it uses a simple cost metric heuristic during a recursive divide-and-conquer function to determine NOT and Toffoli gate placement**

I. INTRODUCTION

Reversible computing has of late been gaining a great deal of attention from researchers. There are a number of reasons for this attention. One reason is the fact that a computer based on reversible logic operations should be able to reuse a fraction of the signal energies that comes close to 100% [1], a result first identified in [2]. Another reason for the interest in reversible computing is in its' relationship to quantum computing; indeed, reversible computing can be considered a special case of quantum computing [3] and many researchers believe that logic synthesis for classical reversible circuits is a first step towards synthesis of quantum circuits [4].

This paper describes a technique for synthesizing a logic function, represented as an exclusive-or sum-of-products (ESOP), to a cascade of reversible Toffoli gates. This technique is of particular interest since it is one of few in the literature that bases its starting representation and intermediate manipulation on a list of cubes (or products) rather than a truth table. We first present a rather simplified technique based on a straightforward mapping of cubes to reversible gates. This first technique generates circuits requiring a higher than usual number of qubits, and so we subsequently present an optimization intended to reduce the number of required qubits.

The paper progresses as follows: Section II provides some brief background on reversible logic and the ESOP representation, while Section III gives an overview of related work. We then present the basic method in Section IV, followed by a description of an optimization in Section V and our experimental results in Section VI. Since this is relatively new we intend to continue developing and applying optimizations, and some of these are detailed in Section VII, Conclusions and Future Work.

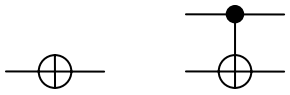
xy	*x'y'*
00	00
01	10
10	01
11	11

A)

xy	*f(xy)*
00	0
01	0
10	0
11	1

B)

Figure 1. A) Example reversible function. B) Example irreversible function.



A) B)

Figure 2. A) NOT and B) TOF1 gate

II. BACKGROUND

A. Reversible Logic

According to Shende *et al.,* a gate is reversible if the (Boolean) function it computes is bijective [4], and a circuit is considered reversible if it consists entirely of reversible gates. For example, the function whose truth table is given in Fig. 1A) is reversible while the function in Fig. 1B) is not.

There are many reversible gates that may be used in synthesis; however in this work only NOT and Toffoli (TOF) gates are used. The symbols used for these gates are shown in Figure 2. A NOT gate has the usual behavior; that is, $(x) \rightarrow (x \oplus 1)$ where $\oplus$ denotes the exclusive-or operator. A Toffoli (TOF) gate has similar behavior, but is controlled by one or more other qubits whose values are not affected; for example, a TOF2 gate with two control bits x and y, has the behavior $(x, y, z) \rightarrow (x, y, xy \oplus z)$. We refer to the computing elements, or "signals" in a reversible function as qubits; the reader is referred to [3] for details on quantum computing.

B. ESOPs

An exclusive-or sum-of-products (ESOP) representation is a variation of the more traditional sum-of-products (SOP). For example, $f = xy + yz$ is in a sum-of-products form. If we replace the OR (+) operator with exclusive-or ($\oplus$) then the function is an ESOP, for example $f = x'yz \oplus yz$. ESOPs have a variety of nice properties, as detailed in [5], and any Boolean function can be represented as an ESOP.

III. Related Work

Although there are a variety of synthesis techniques for reversible logic in the literature, here we will describe only those that utilize an ESOP or similar representation.

Gupta *et al.* [6] present a reversible logic synthesis technique based on a related representation, the positive-polarity Reed-Muller (PPRM) expansion. Rather than use an approach requiring one gate for each term in the expansion they utilize a tree structure to enable investigation of all possible factors of each term, and thus are able to construct a circuit that shares factors. The PPRM representation, while canonical, is a special type of ESOP, with a more rigorous definition, and thus will almost always have more terms than the ESOP representation used in our work.

Maslov and Dueck have also conducted an analytical comparison of their technique and EXOR PLAs, a structure that can implement ESOP representations [7]. They find that their reversible cascade with minimal garbage (RCMG) model compares favorably to an EXOR PLA, particularly for one class of functions for which the ESOP representation has exponential complexity. However, their analysis is based entirely on circuit complexity.

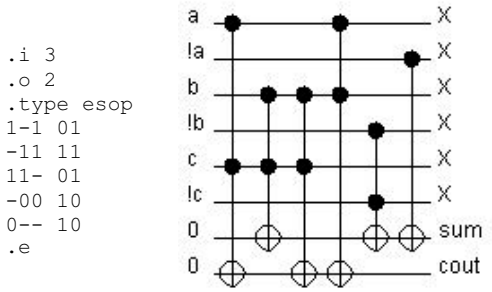
Finally, Perkowski and other researchers have also looked into using ESOPs as a starting point for reversible logic synthesis [8]. In [8] a new class of reversible gates is introduced, allowing modification of two qubits, but requiring a significantly higher level of complexity. The technique in [8] also requires a factorization of each of the ESOPs representing the multiple outputs. This method reported achieving good results in terms of gate numbers – in many cases they required only one gate per product term in the ESOP representation – but did require the use of some garbage lines.

IV. ESOP Mapping Method

The ESOP mapping method can be described as follows. We assume a circuit is given in an ESOP cube-list representation. The circuit to be generated will require $2n+m$ qubits where n is the number of inputs to the function and m is the number of outputs (of course, in a reversible function, $m=n$ in the best case). The $2n$ qubits correspond to each input in the positive and negative polarity. The algorithm then generates a Toffoli gate for each cube of each output in the list of ESOP cubes. The basic algorithm is outlined in Figure 3. An example is shown in Figure 4.

```
basicCascadeGen(esop)
    cascade.toffoliList = empty
    //create qubits
    foreach i in esop.inputs
      //add a qubit representing literals of i
      cascade.addQubit(i, positive)
      cascade.addQubit(i, negative)
    foreach o in esop.outputs
      //add a constant 0 qubit for each output
      cascade.addQubit(o, constant 0)
    //create TOF gates
    foreach c in esop.cubes
      foreach o in esop.outputs
        If c in onset(o)
          //add a toffoli gate
          t = new ToffoliGate
          t.target = cascade.getQubit(output)
          for each literal in c
            t.addControl(cascade.getQubit(literal))
          cascade.addToffoli(t)
```

Figure 3. ESOP Mapping Method



A) B)

Figure 4. A) Full Adder ESOP Cubelist. B) ESOP Mapping Method applied to A)

V. Optimization

The first optimization to consider is how to reduce the number of qubits; for an n-input reversible function most synthesis techniques require only n qubits, while our basic method requires $2n+m$. In many cases it may be likely that the negated form of a literal is not used, and when it is used, it can be generated by using a NOT gate on the qubit supplying the non-negated form of the literal.

We have implemented a technique by which a cost metric is computed for each allowable variable. The cube list is then sorted in such a way that, for the variable with the lowest cost metric, all cubes with its non-negated form appear before all cubes containing its negated form. For cubes where the variable is a don't care, we arbitrarily place that cube in the non-negated list. This process is then repeated within these two halves, and so on. Due to the use of an ESOP representation, the algorithm is able to freely reorder a cube list into the two halves, giving the algorithm greater flexibility in determining NOT and TOF gate placement.

The variable *v* cost metric is shown in (1). The cost metric gives lower scores to variables in a cube list that:

- are balanced with respect to its literals
- appear frequently

$$\text{cost}_v = \alpha \frac{1}{\sum |v_i|} + \beta \left| \sum v_i \right|$$

$$\text{where} \begin{cases} v \text{ in cube}_i \begin{cases} v \text{ is positive}, v_i = 1 \\ v_i = -1 \end{cases} \\ v_i = 0 \end{cases} \quad (1)$$

A standard cube is normally represented as a string consisting of '0', '1', and '-' terms for each variable, depending if the variable is present in negative or positive polarity, or is a don't care. As (1) suggests, '0' is converted to -1, '1' is converted to +1, and '-' is converted to 0. Additionally, the two coefficients α and β are used to control the effect of either portion of the cost metric. α controls the weight of the variable frequency term and β controls the balanced variable term. One may see that the α term becomes smaller, as the summation term in the denominator becomes larger, which corresponds directly with the number of times a variable appears in a cube list. The β term gets bigger if the variable is not balanced with respect positive/negative literal appearance.

The resulting cascade uses *n*+*m* qubits rather than *2n*+*m* generated in the initial method. Figure details the algorithm. Figure 6 shows the full adder in Figure 4 after the algorithm is applied to it.

VI. Experimental Results

For this work we used EXORCISM-4 [9] to preprocess the benchmarks and produce a minimal ESOP representation. The implementation of the algorithms is in C++. The results were generated on a 3.00 GHz Intel D CPU with 2GB RAM.

Results for a number of binary functions (not necessarily reversible) given as **pla** benchmarks are reported in Table I. Each result is the smallest one produced by this technique in terms of the number of Toffoli gates generated. For each circuit the following information is given:

- circuit : circuit name
- in : number of inputs
- out : number of outputs
- esopCubes : number of ESOP Cubes due to EXORCISM-4
- TOF Gates: number of Toffoli gates generated
- NOT Gates : number of NOT gates generated
- α : α value used during cost metric calculation
- β : β value used during cost metric calculation
- esopTime: cpu time expended to generate an ESOP cube list from the **pla** (for reference purposes only)
- cascadeGenTime : cpu time expended to perform the optimized cascade generation algorithm

```
reorder(cubes, polarity, vars)
    if(cubes.isEmpty || vars.isEmpty)
      return cubes
    bestVar = calcBestVar(cubes, vars)
    {pCubes, nCubes} = split(cubes, bestVar)
    pReorder = reorder(pCubes, positive, vars – bestVar)
    nReorder = reorder(nCubes, negative, vars – bestVar)
    if(polarity==negative)
      nReorderNots = addNots(nReordered, bestVar)
    reorderedCubes = reconnect(pReorder, nReorderNots)
    return reorderedCubes

cascadeGen(esop)
   reorderedCubes = reorder(esop.cubes, positive, esop.vars)
   cascade = convertCubesToToffoli(reorderedCubes)
   removeExtraNots(cascade)
```

Figure 5. Cascade Generation with Not Insertion

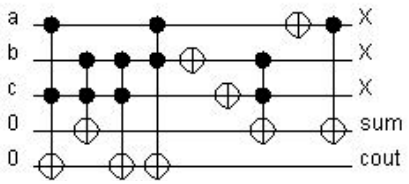


Figure 6. Full Adder with NOTs

In Table I, the synthesis times reported are all < 1 second, even for circuits with over 100 inputs. However, one may note that the full adder was generated with 6 TOF gates and 3 NOT gates, while it is known that a full adder can be optimally implemented with 4 gates total. Additionally, xor5 has 2 NOT gates, which are unnecessary. The presence of the NOT gates is due to the initial ESOP cube list. So, there is a trade off between synthesis speed and quality of the resulting cascade in terms of the number of TOF gates.

Additionally, the results make a distinction between the number of TOF gates and number of NOT gates generated. The number of gates is purely based on the number of shared cubes of the ESOP cube list. One must also be aware these cascades may be comprised of TOF gates with a large number of control qubits. In general, circuits with large input sizes will generally have large TOF sizes as well.

The most noteworthy observation is how the algorithm is able to handle circuits with fairly large input sizes. This is a characteristic that is not shared by many other cascade generation algorithms. In the literature, other algorithms are reported to handle circuits with input sizes of up to 20 and for special cases. We attribute this capability to the fact that the algorithms presented here do not rely upon truth tables as input and also do not traverse a large search space during gate placement. The largest circuit generated was frg2, which has 143 inputs and 139 outputs.

Additionally, varying sets of coefficients were set up such that $\beta = 1 - \alpha$. This was done to gauge the effectiveness of either parameter in determining the best split variable. In our experiments, no clear set of coefficients was found that consistently generated cascades with fewer NOT gates.

VII. Conclusions & Future Work

We presented an ESOP based algorithm for cascade synthesis that is able to generate cascades for functions with large number of input variables. This is in part due to the algorithm not being reliant upon exponentially-sized input, such as truth tables. The algorithm is fast as it uses a simple cost metric heuristic during a recursive divide-and-conquer method to determine NOT and TOF gate placement. However, the resulting cascades require $n+m$ qubits

We believe this approach may be a viable initial mapping procedure that can be followed by circuit quality enhancing optimizations to be developed in the future. Other future work includes developing techniques for reducing the number of initial ancilla inputs by augmenting the cascade with additional gates that allow for the "literal" outputs to generate function outputs.

References

[1] M. P. Frank, "Introduction to Reversible Computing: Motivation, Progress, and Challenges," Proc. 2nd Conference on Computing Frontiers, pp. 385–390, 2005.

[2] LR. Landauer, "Irreversiblbility and Heat Generation in the Computing Process," *IBM Journal of Research and Development* **5**, pp. 183–191 (July 1961).

[3] M. A. Nielsen and I. L. Chuang, **Quantum Computation and Quantum Information**. Cambridge University Press, 2000.

[4] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes, "Synthesis of Reversible Logic Circuits," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, **22**(6), pp. 710–722, (June 2003).

[5] B. Steinbach and A. Mishchenko, "SNF: a Special Normal Form for ESOPs," Proc. Reed-Muller Workshop, pp. 66–81, 2001.

[6] P. Gupta, A. Agrawal, and N. K. Jha. "An Algorithm for Synthesis of Reversible Logic Circuits," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, **25**(11), pp. 2317–2330, (Nov. 2006).

[7] D. Maslov and G. W. Dueck, "Complexity of Reversible Toffoli Cascades and EXOR PLAs", Proc. 12th International Workshop on Post-Binary ULSI Systems, Tokyo, May 2003 (downloaded Mar. 2007 from http://www.cs.unb.ca/profs/gdueckreversible/esop.pdf).

[8] M. H. A. Khan and M. A. Perkowski, "Multi-Output ESOP Synthesis with Cascades of New Reversible Gate Family", Proc. 6th International Symposium on Representations and Methodology of Future Computing Technology, (RM), pp. 144–153, 2003

[9] A. Mishchenko and M. Perkowski, "Fast Heuristic Minimization of Exclusive-Sums-or-Products", 5th International Reed-Muller Workshop, pp. 242–250, 2001.

TABLE I. Experimental Results

Circuit	in	out	esopCubes	TOF Gates	NOT Gates	α	β	esopTime (s)	cascadeGenTime (s)
bw	5	28	22	251	11	1.00	0.00	0.02	0.00
xor5	5	1	5	5	2	1.00	0.00	0.00	0.00
5xp1	7	10	31	61	29	0.50	0.50	0.01	0.00
risc	8	31	27	133	19	0.50	0.50	0.02	0.00
cordic	23	2	776	1546	711	0.75	0.25	14.67	0.02
ttt2	24	21	60	92	41	1.00	0.00	0.08	0.00
vg2	25	8	184	214	286	1.00	0.00	0.11	0.02
bc0	26	11	167	562	158	0.75	0.25	0.47	0.00
in7	26	10	35	64	34	0.75	0.25	0.02	0.00
chkn	29	7	144	147	202	0.50	0.50	0.14	0.02
term1	34	10	540	702	127	0.25	0.75	2.62	0.02
apex2	39	3	1637	1755	1005	0.75	0.25	45.27	0.06
seq	41	35	248	1877	272	0.25	0.75	1.29	0.02
apex1	45	45	288	1348	306	0.75	0.25	1.31	0.03
apex3	54	50	258	2045	278	0.75	0.25	6.90	0.05
dalu	75	16	1472	3472	644	1.00	0.00	0.11	0.00
e64	65	65	65	129	64	0.75	0.25	0.06	0.03
example2	85	66	205	280	81	0.25	0.75	3.35	0.05
x4	97	71	299	460	155	1.00	0.00	2.41	0.05
apex5	117	88	398	541	163	0.50	0.50	4.98	0.16
ex4	128	28	316	321	417	0.75	0.25	0.91	0.20
apex6	135	99	409	569	236	1.00	0.00	6.86	0.14
frg2	143	139	1116	1971	339	0.50	0.50	184.70	0.72
i2	201	1	257	257	536	0.75	0.25	0.80	0.28