Fast Exact Toffoli Network Synthesis of Reversible Logic

Robert Wille and Daniel Große

Group for Computer Architecture (Prof. Dr. Rolf Drechsler)
University of Bremen, 28359 Bremen, Germany
{rwille,grosse}@informatik.uni-bremen.de

***Abstract*— The research in the field of reversible logic is motivated by its application in low-power design, optical computing and quantum computing. Hence synthesis of reversible logic has become a very important research area in the last years. In this paper exact algorithms for the synthesis of generalized Toffoli networks are considered. We present an improvement of an existing synthesis approach that is based on Boolean Satisfiability. Furthermore, the principle limits of the original and the improved approach are shown. Then, we propose a new method using problem specific knowledge during the synthesis process to overcome these limits. Experimental results demonstrate improvements of the overall synthesis time up to four orders of magnitude.**

I. Introduction

Reversible logic has applications in low-power design, optical computing and especially in quantum computing. In the context of quantum computation it is known that some exponentially hard problems can be solved in polynomial time [1]. Necessarily all quantum computations are reversible. Therefore synthesis of reversible logic has become an intensively studied topic. The synthesis of reversible logic differs significantly from traditional irreversible logic. In a reversible network fan-out and feed-back are not allowed. Consequently a network for reversible logic consists of a cascade of reversible gates. The most frequently used gate type is the Toffoli gate [2] which will also be used in this paper. The idea of this gate is to invert one input line (the target line) if the product of a set of control lines evaluates to true.

For the synthesis of reversible logic several approaches have been proposed. An approach that is based on enumeration and that uses network equivalences to rewrite a limited set of gates has been presented in [3]. Other synthesis procedures use heuristics like e.g. spectral techniques [4], positive polarity Reed-Muller expansions [5] or transformation based synthesis [6]. In [7] a method has been proposed that synthesizes the reversible function in a first step and then based on transformations (using so called templates) a realization with fewer gates is computed. An exact synthesis method based on reachability analysis is described in [8]. However, this procedure is geared towards quantum gates, not Toffoli gates. Recently, in [9] the first exact synthesis approach for reversible logic using generalized Toffoli gates has been presented. This approach uses *Boolean Satisfiability* (SAT) during the synthesis process. Since the synthesis problem is transformed to the Boolean level complexity problems occur.

The contribution of this paper is twofold. First, we lift the encoding of the synthesis problem to a higher level. Based on this formulation, the SAT solver used in [9] is replaced by a *Satisfiability Modulo Theories* (SMT) solver. As shown by the experiments this already leads to a significant speed-up of the synthesis algorithm. Second, after a detailed analysis of the principle limits of these synthesis procedures we propose a completely new approach. Based on a general solver framework problem specific knowledge is available during the search process. Within this framework we specify dedicated modules to represent a network of generalized Toffoli gates. Besides a high-level and very compact problem representation, the modules allow for much more efficient decision and propagation strategies. In total, our new approach outperforms previous approaches up to four orders of magnitude.

The paper is structured as follows. Background on reversible logic and SAT is presented in the next section. Section III describes the exact synthesis algorithm. Here the basic concept of [9] is reviewed and a more general encoding on a higher level is introduced. In Section IV the limits of these approaches are discussed. Then, the solver framework and the new problem specific approach are presented. Finally, experimental results and a summary are given in the last two sections.

II. Preliminaries

A. Reversible Logic

A reversible logic gate is a n-input n-output function that maps each possible input vector to a unique output vector. In other words this function is a bijection. Many reversible gates have been studied. Generalized Toffoli gates [2] are widely used. In the rest of this paper we only consider Toffoli gates that are defined as follows:

Definition 1: Let $X := \{x_1, \ldots, x_n\}$ be the set of domain variables. A *generalized Toffoli gate* has the form $TOF(C, t)$, where $C = \{x_{i_1}, \ldots, x_{i_k}\} \subset X$ is the set of control lines and $t = \{x_j\}$ with $C \cap t = \emptyset$ is the target line. The gate maps $(x_1, \ldots, x_n)$ to $(x_1, \ldots, x_{j-1}, x_j \oplus x_{i_1} \ldots x_{i_k}, x_{j+1}, \ldots, x_n)$.

Due to restrictions in quantum mechanics as network topology only a cascade structure can be used. This structure is simply a number of Toffoli gates in a cascade.

Definition 2: The *reversible cost* (or simply, *cost*) of an implementation of a reversible function f is defined as the number of gates in the network representation that realizes f.

In the following the basics of SAT and common used techniques are given.

B. SAT

The *Boolean Satisfiability* (SAT) problem is defined as follows:

Definition 3: Let h be a Boolean function in *Conjunctive Normal Form* (CNF), i.e. a product-of-sum representation. Then the SAT problem is to determine whether there exists an assignment to the variables of h such that h evaluates to true or to prove that no such assignment exists.

In the past several (backtracking) algorithms (so called *SAT solver*) were proposed [10]–[13]. Most of them are based on three essential procedures: (1) The decision heuristic assigns values to free variables, (2) the propagation procedure determines implications due to the last assignment(s) and (3) the conflict analysis tries to resolve conflicts by backtracking that occur during the search. Advanced techniques like e.g. *efficient Boolean constraint propagation* [12] or *conflict analysis* [11] are common in state-of-the-art SAT solvers today.

Due to tremendous improvements in the recent past, several researchers investigated the combination of SAT solvers with decision procedures for decidable theories resulting in *SAT Modulo Theories* (SMT) [14], [15]. A SMT solver integrates a Boolean SAT solver with other solvers for specialized theories (e.g. bitvector logic as used later in this paper). Here the SAT solver works on an abstract representation (also in CNF) of the problem and steers the overall search process, while each (partial) assignment of this representation

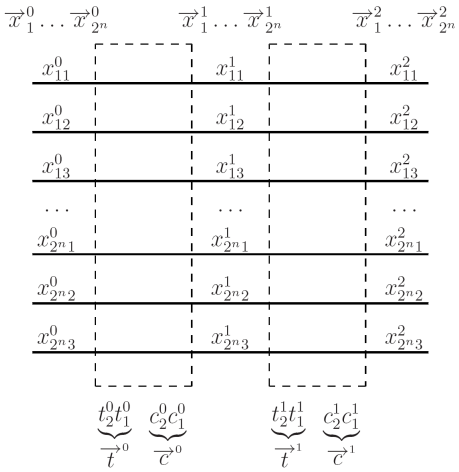

Fig. 1. Representation of the problem

has to be validated by the theory solver for the theory constraints. Thus, advanced SAT techniques together with specialized theory solvers can be utilized.

III. Exact Toffoli Network Synthesis

In this section first we briefly review the main flow of the exact synthesis algorithm for reversible logic from [9]. The basic idea is to check if there exists a Toffoli network representation for the reversible function f with d gates, where d is increased in the next iteration if no realization is found. The iterative checks are performed by (1) encoding the synthesis problem as a SAT instance and (2) testing the SAT instance for satisfiability using an off-the-shelf solver.

In the following the general formulation of the synthesis problem and two encodings are presented. The first encoding uses Boolean SAT – following the approach in [9] – and the second encoding is based on SMT.

A. Problem Formulation and Encodings

For a reversible function the synthesis problem is formulated using three constraints. The resulting constraint formula evaluates to true iff there exists a Toffoli gate network for the reversible function f (given as truth table) of cost d. Before the constraint formula is described in detail some definitions are given:

Definition 4: Let $f : B^n \rightarrow B^n$ be a reversible function. Then three variable vectors are defined:

1) $\vec{x}_i^k = (x_{in}^k x_{i(n-1)}^k \dots x_{i1}^k)$ with $1 \leq i \leq 2^n$ and $0 \leq k \leq d$ is a Boolean vector representing the input-, temporary- or output variables at depth k for line i of the truth table of f. The left side of the truth table corresponds to the vector $\vec{x}_i^0$, the right side to the vector $\vec{x}_i^d$, respectively.
2) $\vec{t}^k = (t_{\lceil \log_2 n \rceil}^k \dots t_1^k)$ with $0 \leq k < d$ is a Boolean vector representing a binary encoding of a natural number $t^k \in \{0, \dots, n-1\}$ which defines the choosen target line for the Toffoli gate at depth k.
3) $\vec{c}^k = (c_{n-1}^k, c_{n-2}^k, \dots, c_1^k)$ with $0 \leq k < d$ is a Boolean vector representing the control lines of the Toffoli gate at depth k. Assigning $c_i^k = 1$ with $(1 \leq i \leq n-1)$ means that line $(t^k + i) \mod n$ becomes a control line of the Toffoli gate at depth k.

As an example the variables needed to formulate the constraints for a function with $n = 3$ inputs/outputs and depth $d = 2$ are shown in Figure 1. Note: For each of the $2^3 = 8$ lines in the truth table $n = 3$ lines in the network with the respective vectors for input-, temporary- and output variables are used. That is the reason why overall $3 \cdot 8 = 24$ lines are considered to formulate the problem.

Based on the above definitions the synthesis problem for a reversible function f with d generalized Toffoli gates can be formulated as follows: Is there an assignment for all variables of the vectors $\vec{t}^k$ and $\vec{c}^k$ such that for each line i $\vec{x}_i^0$ is equal to the left side of the truth table while $\vec{x}_i^d$ is equal to the corresponding right side?

More formally the constraint formula is the conjunction of the following three constraints:

1) The *input/output constraints* set the input/output pair of each line of the truth table given by the reversible function f:
$$\bigwedge_{i=1}^{2^n} [\vec{x}_i^0]_2 = i \wedge [\vec{x}_i^d]_2 = f(i)$$
2) The *functional constraints* for possible Toffoli gates that are chosen by an assignment to $\vec{t}^k$ and $\vec{c}^k$ are:
$$\bigwedge_{i=1}^{2^n} \bigwedge_{k=0}^{d-1} \vec{x}_i^{k+1} = t(\vec{x}_i^k, \vec{t}^k, \vec{c}^k)$$
These constraints state that if in line i at depth k a Toffoli gate is selected (i.e. $\vec{t}^k$ and $\vec{c}^k$ are completely assigned) the variables in the next depth $k+1$ are computed from the variables at depth k with the Toffoli gate function $t(\vec{x}_i^k, \vec{t}^k, \vec{c}^k)$. The function $t(\vec{x}_i^k, \vec{t}^k, \vec{c}^k)$ covers the functionality of a Toffoli gate with target line $t^k = [\vec{t}^k]_2$ and the control lines defined by $\vec{c}^k$.
3) The *exclusion constraints* ensure that illegal assignments to $\vec{t}^k$ are excluded since not all values of $\vec{t}^k$ are necessary to enumerate all possible target lines:
$$\bigwedge_{k=0}^{d-1} [\vec{t}^k]_2 < n$$

Based on these constraints two encodings are considered in the following. By the first encoding the constraints are transformed into CNF. The resulting CNF is used as input for an off-the-shelf SAT-solver as proposed in [9]. The transformation starts with a conversion into a Boolean formula. Then, by introducing new variables the CNF form for the Boolean formula is created. As well known this can be done in time and space linear in the size of the original Boolean formula [16]. However, the resulting CNF is only a pure Boolean representation based on clauses and due to the auxiliary variables there is a significant overhead. Therefore we propose another encoding that avoids the conversion to the Boolean level. Instead the constraints are represented in bitvector logic that can be handled by SMT. All bitvector variables and most of the operators are preserved; hardly any auxiliary variables are needed. Furthermore, compared to CNF the formulation at this higher level of abstraction allows stronger implications. By using the SMT-based encoding together with an off-the-shelf SMT-solver all these advantages are exploited. The experiments demonstrate that this encoding leads to significant speed-ups already. However, as shown in the next section even more runtime can be saved by using our solver framework that exploits problem specific knowledge of the synthesis problem during the solve process.

IV. Using Problem Specific Knowledge in Exact Toffoli Network Synthesis

The two approaches presented in the previous section encode the synthesis problem of the current iteration and use SAT/SMT solvers to check if a network for the reversible function exists. These solvers provide sophisticated techniques like powerful reasoning or learning, but are only optimized for solving Boolean formulas or bitvector

constraints in general. In this section a new approach is introduced that uses problem specific information in addition to common SAT techniques. During the solve process this information becomes available since our approach represents the synthesis problem in terms of Toffoli gate modules.

First, the limits of the SAT/SMT-based approaches are discussed. Then, the framework of the new approach is presented. In the main part of this section the developed Toffoli modules, dedicated implication procedures and decision heuristics are introduced. Altogether the new representation of the synthesis problem as well as the novel strategies enables dramatic improvements in the overall runtimes.

A. Limits of Common SAT Provers

The input of a SAT solver is a Boolean function in terms of clauses; the input of a SMT solver is a description in bitvector logic. Both solvers are optimized for there particular problem representation. E.g. common SAT solvers utilize the *two literal watching scheme* to carry out implications, which exploits the special structure of clauses [12]. SMT solvers use *canonizing* to efficiently handle bitvector constraints [17]. Furthermore, highly optimized heuristics have been developed e.g. to decide the assignment of variables if no more implications are possible. Here often strategies are used, which are based on statistical information, e.g. occurrences or activities of variables [18].

All these techniques work very well if CNF formulas or bitvector logic are considered. However, these general solvers are not able to take specific properties of the problem into account. Promising problem specific strategies for the exact Toffoli network synthesis are:

- The type of the Toffoli gates (represented by $\vec{t}^k$ and $\vec{c}^k$) near to the inputs should be defined first, because the corresponding input variables are already assigned by the truth table.[1] This allows for early implications and helps to determine the types of the remaining gates or to detect conflicts faster. Thus, $\vec{t}^k$ and $\vec{c}^k$ with small k should be preferred in the decision procedure.
- If the assignment of an input line of a Toffoli gate is not equal to the assignment of the corresponding output line of the same gate, this line has to be the target line. This observation allows to imply the assignment of variables in $\vec{t}^k$.
- If the target line of a Toffoli gate is known the values of all remaining lines can be implied if there is an assignment at the corresponding input or output.

These specific strategies cannot be provided by a standard SAT or SMT solver. Moreover, extensions of standard solvers in this direction (e.g. by modifications of the heuristics) are not possible in general, because most of the problem specific information is lost while encoding the instance. SAT and SMT solvers just have a clause database or constraint database. Thus, procedures like the ones described above can only be exploited with a solver that is based on a problem specific representation.[2]

B. Framework

To formulate the exact synthesis problem we use the framework provided by *SWORD* [19]. Here, the problem is represented in terms of so called *modules*. Thus, while SAT solvers provide strategies optimized for clauses and SMT solvers for bitvector constraints, this framework makes problem specific information available in the modules and thereby allows dedicated decision and propagation strategies. Furthermore, it utilizes all sophisticated SAT techniques like conflict analysis or learning as well.

[1] This observation also holds for modules near to the outputs.

[2] In principle, this problem can be prevented by introducing additional constraints to the problem instance. But then the encoding becomes inefficient due to a very large number of constraints.

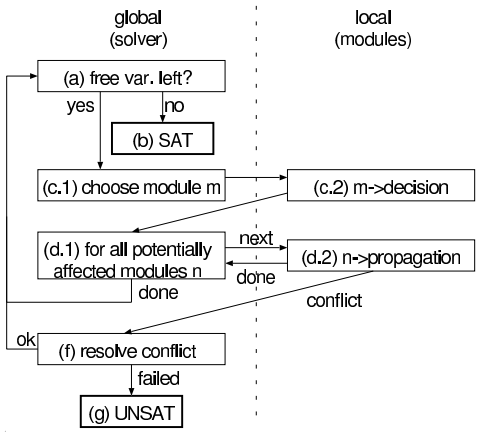

Fig. 2. Framework

The architecture of the framework is shown in Figure 2. The basic algorithm is similar to the procedures as applied in standard SAT or SMT solvers: While free variables remain (a) a decision is made (c), implications resulting from this decision are carried out (d), and if a conflict occurs, it is analyzed (f). The important difference is that the framework has two operation levels: the *global* algorithm controls the overall search process and calls the *local* procedures of modules for decision and implication. Thus, decision making and the implication engine can be adjusted for each module.

In more detail, the solver first chooses a particular module based on a *global decision heuristic* (c.1). Then, this module chooses a value for one of its variables according to a *local decision heuristic* (c.2). Afterwards, the solver calls the *local implication procedures* (d.2) of all modules that are potentially affected (d.1) by the previous decision or implication. The chosen modules imply further assignments and detect conflicts.

C. Toffoli Synthesis

In the following sections we describe the customization of the framework for the exact Toffoli synthesis. First, we introduce the developed Toffoli gate module. Then, the decision and propagation strategies including the local and global procedures are described.

1) Toffoli Module: A concrete Toffoli synthesis instance for the reversible function f with cost d consists of d instantiated modules in a cascade structure – for each depth k one module. Each module has access to its relating variables $\vec{t}^k$, $\vec{c}^k$, $\vec{x}_i^k$ and $\vec{x}_i^{k+1}$. A module corresponds to the dashed area in Figure 1. The functionality of a Toffoli gate is defined by methods of the module, i.e. a concrete Toffoli gate function is selected by assigning $\vec{t}^k$ and $\vec{c}^k$ (see also Definition 4).

2) Decision Strategies: The decision heuristic chooses a variable that is assigned if no further implication is possible. Therefore decisions are preferred that cause many implications. Both, the global and the local decision strategy is motivated by the following two observations:

- A module can imply many other assignments if the target line t^k of the represented gate is known (i.e. if $\vec{t}^k$ is completely assigned), because in this case the input $\vec{x}_i^k$ and the output $\vec{x}_i^{k+1}$ of all lines i except the target line has to be equal.
- The assignment of most of the inputs $\vec{x}_i^k$ of a gate are either given by the truth table (if $k = 0$) or they can be implied if the types of the previous gates are defined.

These observations lead to the following decision heuristics:

- Global decision heuristic
  Modules whose target lines are still undefined, are selected to

```
(1)  for each (line i)
(2)    if (x_i^k ≠ x_i^{k+1}) // input ≠ output
(3)      imply(t^k); // use value of i
(4)  for each (line i)
(5)    if (i == [t^k]_2) continue;
(6)    imply(x_i^k or x_i^{k+1});
(7)  flipTargetLine = true;
(8)  for each (c_i^k ∈ c^k)
(9)    if (c_i^k == 1 ∧ x^k_{t^k+i mod n} == 0)
(10)     flipTargetLine = false;
(11)     break;
(12) if (!flipTargetLine)
(13)   imply(x^{k+1}_{t^k}); // use value of x^k_{t^k}
(14) else
(15)   if (c^k completely defined)
(16)     imply(x^{k+1}_{t^k}); // use value of x^k_{t^k} ⊕ 1
```

Fig. 3. Propagate routine for module at depth k

make a decision. If all target lines of each module are defined, the module that still has other unassigned variables (from the vector $\vec{c}^k$ for example) is selected. During this process the modules are considered in ascending order starting with depth $k = 0$.

- Local decision heuristic
  Variables representing the target line of the respective Toffoli gate are decided first. If all target line variables are assigned, variables corresponding to the control lines are decided.

Since the overall decision strategy (global and local) ensures, that the gates become completely defined from the first gate to the last gate, there is no need for deciding the variables representing the inputs or outputs. These variables are implied after the corresponding types of the previous gates are defined.

3) Propagation Strategies: The propagation procedures of a module consider the connected variables for the implication of the values. The pseudo-code of the propagation routine is shown in Figure 3. The propagation routine is divided in three cases:

1) Propagate the position of the target line (lines 1-3)
   If the assignment of an input is not equal to the assignment of the corresponding output of the same line, then this line has to be the target line of the gate. The position t^k of this target line is assigned to $\vec{t}^k$.
2) Propagate non-target lines (lines 4-6)
   If the target line is known all outputs are implied whose corresponding inputs are assigned (except the ones at the target lines). This also holds vice versa. Thus, the output (input) is assigned to the value of the corresponding input (output).
3) Propagate the target line (lines 7-16)
   In the last step the outputs of the target line is assigned. To do so, the assignment of the control lines $\vec{c}^k$ and the corresponding input variables are considered. If a line is a control line and the input of this line is 0 (line 9), then the output of the target line has to be equal to its corresponding input (line 13). Otherwise, if additionally $\vec{c}^k$ is completely defined (i.e. no other control line with input value 0 can occur), the output of the target line has to be equal to the opposite value of the input of this line (line 16).

Altogether the presented decision and propagation strategies allows both, efficient proofs that a Toffoli network with cost d for a given function f exists or that no such network exists. In the first case the resulting network can be extracted from the assignments to $\vec{t}^k$ and $\vec{c}^k$.

V. Experimental Results

This section provides experimental results for the exact Toffoli network synthesis. As solver for the CNF encoding of [9] *MiniSat v1.14* [13] and as solver for the SMT encoding described in Section III *MathSat*[3] [14] have been used, respectively. The problem specific approach described in Section IV has been implemented in C++. All experiments were carried out on a AMD Athlon 3500+ with 1 GB of memory. The timeout was set to 5000 CPU seconds.

The results are shown in Table I. The first column provides the name of the benchmark. In column *depth* the minimal costs (i.e. the minimal number of Toffoli gates) necessary to synthesize the function is given. The next column provides the runtime of the particular synthesis algorithms in CPU seconds (denoted by *Time*). Furthermore, the improvements of the new approaches are given, i.e. the runtime of MiniSat divided by the runtime of MathSat/the problem specific approach (denoted by $Impr_{cnf}$) and the runtime of MathSat divided by the runtime of the problem specific approach (denoted by $Impr_{smt}$), respectively.

In total three sets of benchmarks have been considered: Completely specified, incompletely specified and random generated functions. The specification of *3_17* and *mod5mils* can be found on [21]; *peres*, *fredkin* and *miller* on [22], [23] and [24], respectively. The *peres-double* function is specified by cascading two Peres functions, the *mod5d1* and *mod5d2* functions realize the Grover's oracle and *graycode6* computes the graycode. The incomplete specified functions *decod24* and *ALU* are described in [5], *4mod5* specifies a divisibility checker and *rd32* defines a full adder. Here, several options for embedding these functions into reversible ones are available. First, one can chose how to set the constant inputs. Second, there is a choice in where to place the garbage outputs. In Table I the results for different configurations are documented (indicated by *-v0*, *-v1*, ...). Finally, random functions with four inputs have been generated for further experimental tests.

As the results show, for most of the benchmarks a corresponding Toffoli network can be synthesized faster by the SMT approach than by using the CNF formulation. Only in some cases the CNF approach is slightly better. However, this only holds for benchmarks that can be synthesized in less than one second, e.g. *peres* or *fredkin*. On average improvements by a factor of 10 are achieved; in the best case a factor of 179. But as can be seen the problem specific approach outperforms both approaches. Here, the runtimes are further reduced by a factor ranging from 20 to 60 for nearly all benchmarks. Furthermore, using the problem specific approach, Toffoli networks for the benchmarks *ALU-v2* and *ALU-v3* are synthesized within the given timeout. In comparison to the CNF synthesis algorithm of [9] on average speed-ups of three orders of magnitude are obtained when using the problem specific approach; for the best case (*graycode6*) a speed-up of four orders of magnitude is documented.

VI. Conclusions

It has been shown, that choosing an encoding for exact Toffoli network synthesis is crucial for the resulting runtimes. Furthermore, an algorithm has been presented that uses problem specific information during the synthesis process. This allows to incorporate dedicated strategies that improve the synthesis time dramatically. The experiments have shown speed-ups of up to four orders of magnitude using the proposed approach.

[3]Because the current free state-of-the-art SMT solver *Yices* [15] (according to SMTLIB [20]) showed some buggy behavior for certain benchmarks (also observed at SMT Competition 2007) no experiments for this solver are documented here.

TABLE I
EXPERIMENTAL RESULTS

Benchmark	Depth	CNF Time	SMT Time	SMT Impr_{cnf}	Problem Specific Approach Time	Problem Specific Approach Impr_{cnf}	Problem Specific Approach Impr_{smt}
Complete Specified Functions							
peres	2	0.01s	0.03s	0.33	< 0.01s	> 1.00	> 33.00
fredkin	3	0.03s	0.12s	0.25	< 0.01s	> 3.00	> 24.00
peres-double	4	2.35s	0.36s	6.53	0.01s	235.00	36.00
miller	5	0.23s	0.22s	1.05	< 0.01s	> 23.00	> 22.00
mod5mils	5	48.28s	3.81s	12.67	0.08s	603.50	47.63
graycode6	5	583.14s	3.25s	179.43	0.12s	4859.50	27.08
3_17	6	0.43s	0.72s	0.59	0.03s	14.33	24.00
mod5d1	7	2094.13s	135.36s	15.47	11.21s	186.80	12.07
mod5d2	8	1616.07s	56.72s	28.49	9.06s	178.37	6.26
Incomplete Specified Functions							
rd32-v0	4	2.97s	0.54s	5.50	< 0.01s	> 297.00	> 54.00
rd32-v1	5	13.51s	1.84s	7.34	0.04s	337.75	46.00
4mod5-v0	5	122.54s	12.70s	9.65	0.69s	177.59	18.40
4mod5-v1	5	413.21s	43.86s	9.42	0.48s	860.85	91.38
decod24-v0	6	6.54s	1.33s	4.92	0.02s	327.00	66.50
decod24-v1	6	6.22s	1.44s	4.32	0.09s	69.11	16.00
decod24-v2	6	7.25s	1.35s	5.37	0.02s	362.50	67.50
decod24-v3	7	28.88s	3.31s	8.73	0.18s	160.44	18.39
ALU-v0	6	1998.83s	223.48s	8.94	8.76s	228.18	25.51
ALU-v1	7	> 5000s	1692.29s	> 2.95	369.14s	> 13.54	4.58
ALU-v2	7	> 5000s	> 5000s	–	840.25s	> 5.95	> 5.95
ALU-v3	7	> 5000s	> 5000s	–	764.04s	> 6.54	> 6.54
Random Functions							
rand0	8	158.43s	22.05s	7.19	11.14s	14.22	1.98
rand1	8	758.53s	53.96s	14.06	6.93s	109.46	7.79
rand2	9	1324.90s	179.83s	7.37	63.25s	20.95	2.84
rand3	9	907.79s	195.46s	4.64	116.56s	7.79	1.68
rand4	9	2696.71s	426.98s	6.32	92.72s	29.08	4.61

VII. ACKNOWLEDGMENT

We wish to thank Rolf Drechsler and Gerhard Dueck for many helpful discussions and inspirations. Furthermore, we thank Mathias Soeken for his help in the area of SMT.

REFERENCES

[1] M. A. Nielsen and I. Chuang, *Quantum computation and quantum information*. Cambridge University Press, 2000.

[2] T. Toffoli, "Reversible computing," in *Automata, Languages and Programming*, W. de Bakker and J. van Leeuwen, Eds. Springer, 1980, p. 632, technical Memo MIT/LCS/TM-151, MIT Lab. for Comput. Sci.

[3] V. Shende, A. Prasad, I. Markov, and J. Hayes, "Reversible logic circuit synthesis," in *Int'l Conf. on CAD*, 2002, pp. 353–360.

[4] D. M. Miller and G. W. Dueck, "Spectral techniques for reversible logic synthesis," in *6th International Symposium on Representations and Methodology of Future Computing Technology*, 2003, pp. 56–62.

[5] P. Gupta, A. Agrawal, and N. Jha, "An algorithm for synthesis of reversible logic circuits," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 25, no. 11, pp. 2317–2330, 2006.

[6] D. M. Miller, D. Maslov, and G. W. Dueck, "A transformation based algorithm for reversible logic synthesis," in *Design Automation Conf.*, 2003, pp. 318–323.

[7] D. Maslov, G. W. Dueck, and D. M. Miller, "Toffoli network synthesis with templates." *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 24, no. 6, pp. 807–817, 2005.

[8] W. Hung, X. Song, G. Yang, J. Yang, and M. Perkowski, "Optimal synthesis of multiple output Boolean functions using a set of quantum gates by symbolic reachability analysis." *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 25, no. 9, pp. 1652–1663, 2006.

[9] D. Große, X. Chen, G. Dueck, and R. Drechsler, "Exact SAT-based toffoli network synthesis," in *Great Lakes Symp. VLSI*, 2007, pp. 96–101.

[10] M. Davis, G. Logeman, and D. Loveland, "A machine program for theorem proving," *Comm. of the ACM*, vol. 5, pp. 394–397, 1962.

[11] J. Marques-Silva and K. Sakallah, "GRASP: A search algorithm for propositional satisfiability," *IEEE Trans. on Comp.*, vol. 48, no. 5, pp. 506–521, 1999.

[12] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Design Automation Conf.*, 2001, pp. 530–535.

[13] N. Eén and N. Sörensson, "An extensible SAT solver," in *SAT 2003*, ser. LNCS, vol. 2919, 2004, pp. 502–518.

[14] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. Rossum, S. Schulz, and R. Sebastiani, "The MathSAT 3 System," in *Int. Conf. on Automated Deduction*, 2005.

[15] B. Dutertre and L. Moura, "The YICES SMT Solver," 2006, available at http://yices.csl.sri.com/.

[16] G. Tseitin, "On the complexity of derivation in propositional calculus," in *Studies in Constructive Mathematics and Mathematical Logic, Part 2*, 1968, pp. 115–125, (Reprinted in: J. Siekmann, G. Wrightson (Ed.), Automation of Reasoning, Vol. 2, Springer, Berlin, 1983, pp. 466-483.).

[17] C. Barrett, D. Dill, and J. Levitt, "A decision procedure for bit-vector arithmetic," in *Design Automation Conf.*, 1998, pp. 522–527.

[18] J. Marques-Silva, "The impact of branching heuristics in propositional satisfiability algorithms," in *9th Portuguese Conference on Artificial Intelligence (EPIA)*, 1999.

[19] R. Wille, G. Fey, D. Große, S. Eggersglüß, and R. Drechsler, "SWORD: A SAT like Prover Using Word Level Information," in *Int'l Conference on Very Large Scale Integration*, 2007.

[20] S. Ranise and C. Tinelli, "The Satisfiability Modulo Theories Library (SMT-LIB)," `www.SMT-LIB.org`, 2006.

[21] D. Maslov, *Reversible Logic Synthesis Benchmarks Page*, http://www.cs.uvic.ca/~dmaslov/.

[22] A. Peres, "Reversible logic and quantum computers," *Phys. Rev. A*, no. 32, pp. 3266–3276, 1985.

[23] E. F. Fredkin and T. Toffoli, "Conservative logic," *International Journal of Theoretical Physics*, vol. 21, no. 3/4, pp. 219–253, 1982.

[24] D. M. Miller, "Spectral and two-place decomposition techniques in reversible logic," *Midwest Symposium on Circuits and Systems*, 2002.