Synthese reversibler Logik

Synthesizing Reversible Logic

Robert Wille, Rolf Drechsler, Universität Bremen

Zusammenfassung Traditionelle Technologien (z. B. CMOS) stoßen durch die stetig steigende Miniaturisierung sowie das exponentielle Wachstum der Anzahl an Komponenten immer mehr an ihre Grenzen. Daher werden in Zukunft Alternativen, welche traditionelle Computerchips ersetzen oder zumindest ergänzen, benötigt. Reversible Logik mit ihren Anwendungen z. B. im Gebiet der Quantencomputer, des Low-Power Designs, des Optical Computing, des DNA Computing oder in der Nanotechnologie stellt eine solche Alternative dar und hat sich daher in den vergangenen Jahren zu einem intensiv untersuchten Forschungsgebiet entwickelt. Allerdings unterscheidet sich die Synthese von reversiblen Schaltkreisen deutlich von der traditionellen Logiksynthese. So sind insbesondere Verzweigungen (Fanouts) und Rückkopplungen (Feedback durch FlipFlops) nicht möglich, sodass reversible Schaltkreise aus Kaskaden von reversiblen Gattern bestehen. Dies erfordert komplett neue Syntheseansätze. Die vorliegende Arbeit gibt eine Einführung in das Thema und stellt ausgewählte Syntheseverfahren für reversible Logik im Überblick vor. Dabei werden reversible Funktionen und Schaltkreise zuerst eingeführt, wobei insbesondere der Aspekt der Einbettung von irreversiblen Funktionen detailliert betrachtet wird. Anschließend wird beschrieben, wie sich solche Funktionen (gegeben in Form einer Wahrheitstabelle) mit Hilfe von exakten und heuristischen Verfahren synthetisieren lassen. Da die Beschreibung in Form einer Wahrheitstabelle generell nur für kleine Funktionen anwendbar ist, wird schließlich noch ein Syntheseansatz beschrieben, welcher binäre Entscheidungsdiagramme (engl.: BDDs) verwendet und damit die Synthese deutlich größerer Funktionen ermöglicht. ►►► **Summary** Traditional technologies like CMOS more and more start to suffer from the increasing miniaturization and the exponential growth of the number of transistors. Thus, alternatives that replace or at least enhance traditional computer chips are needed in future. Reversible logic, and its applications in domains like quantum computation, low-power design, optical computing, DNA computing, and nanotechnologies, is such a possible alternative and thus has been become an intensely studied topic in the recent years. But synthesis of reversible circuits significantly differs from traditional logic synthesis. In particular, fan-out and feedback are not allowed so that reversible circuits must be cascades of reversible gates. This requires completely new synthesis approaches. This paper provides an introduction into the topic as well as an overview of selected synthesis methods for reversible logic. Thereby, we review reversible functions as well as reversible circuits and, in particular, focus on the embedding of irreversible functions into reversible ones. Then, we describe how such functions (given as a truth table) can be synthesized using exact as well as heuristic approaches. Since only small functions can be synthesized using a truth table as input, afterwards we describe a new method that exploits Binary Decision Diagrams (BDDs) for reversible logic synthesis of significantly larger functions.

Schlagwörter Reversible Logik, Quantum Computer, Synthese ►►► **Keywords** B.6 [Hardware: Logic Design]; Reversible Logic, Quantum Computation, Synthesis

1 Einleitung

Reversible Logik [2; 14; 29] realisiert Funktionen mit n Ein- und Ausgängen, wobei jede mögliche Eingangsbelegung auf eine eindeutige Ausgangsbelegung abgebildet wird (d. h. Bijektionen werden realisiert). In den letzten Jahren hat sich die Synthese von reversibler Logik zu einem intensiv untersuchten Forschungsgebiet entwickelt. Begründet ist dies insbesondere durch die Tatsache, dass reversible Logik die Basis für viele neue aufkommende Technologien darstellt, während klassische Verfahren (z. B. CMOS) durch die stetig steigende Miniaturisierung sowie das exponentielle Wachstum der Anzahl an Komponenten immer mehr an ihre Grenzen stoßen. So erwarten Wissenschaftler, dass bereits in 10

bis 20 Jahren die derzeit beobachtbare Verdoppelung der Transistorendichte aller 18 Monate (entsprechend dem *Moore's Law*) nicht mehr länger möglich sein wird (siehe z. B. [38]). Spätestens dann werden Alternativen benötigt, wobei – wie die folgenden Beispiele zeigen – reversible Logik eine wichtige Rolle spielen kann.

Low-Power Design: Die Energieabgabe und verbunden damit die Wärmeerzeugung ist für heutige Computerchips ein ernstes Problem. Dabei haben die verwendeten Materialien einen großen Einfluss auf die letztliche Wärmeabgabe. Ein viel fundamentaleres Problem wurde aber bereits in den 60ern bzw. 70ern aufgezeigt: So bewies Landauer in [14], dass irreversible Operationen (unabhängig von der verwendeten Technologie) immer zu einer Energieabgabe führen. Später zeigte schließlich Bennett in [2], dass Energieverluste nur minimiert oder gar beseitigt werden können, wenn Berechnungen ohne Informationsverlust durchgeführt werden. Dies trifft für reversible Logik zu, da hier Eingabedaten bijektiv (und damit ohne Informationsverlust) transformiert werden. Entsprechend wird reversible Logik auch immer interessanter für Anwendungen im Low-Power Design. Im Jahr 2002 konnte bereits ein reversibler Schaltkreis produziert werden, dessen Energieverbrauch so gering ist, dass er nur durch die Energie der Eingangssignale betrieben werden konnte [6].

Quantencomputer: Quantencomputer [20] bieten eine komplett neue Möglichkeit, Berechnungen durchzuführen. Anstatt mit klassischen Bits, wird hierbei mit so genannten Qubits gearbeitet. Diese erlauben nicht nur die Repräsentation der Zustände 0 und 1, sondern auch der Superposition von beiden. Damit können Qubits viele Zustände gleichzeitig repräsentieren und somit bestimmte Probleme deutlich schneller als bisherige Verfahren lösen. So kann z. B. das Faktorisierungsproblem mit Hilfe von Quantenalgorithmen in polynomieller Zeit gelöst werden [20] – für klassische Rechner sind bisher nur exponentielle Verfahren bekannt. Für kleine Zahlen können diese Verfahren auch schon physikalisch umgesetzt werden (siehe z. B. [31]). Die jeweiligen Quantenoperationen sind dabei reversibel. Daher finden Fortschritte im Entwurf von reversibler Logik zugleich auch Anwendung bei Quantenschaltkreisen. Auch wenn hier die Forschung noch in einem frühen Stadium ist, konnten bereits erste sehr viel versprechende Ergebnisse erzielt werden. So wurde im November 2007 von der kanadischen Firma D-Wave Systems ein Quantencomputer mit 28 Qubits (dem Äquivalent zum klassischen Bit) vorgestellt (siehe www.dwavesys.com).

Darüber hinaus können weitere Anwendungen von reversibler Logik im Gebiet des Optical Computings [5], des DNA Computings [2] und in der Nanotechnologie gefunden werden [18].

Den vielfältigen Anwendungsmöglichkeiten von reversibler Logik stehen allerdings auch einige Beschränkungen gegenüber. So sind aufgrund der Reversibilität Verzweigungspunkte innerhalb der Schaltung (Fanouts) wie auch Rückkopplungen (z. B. durch FlipFlops) nicht erlaubt [20]. Als Konsequenz lässt sich reversible Logik nur als eine Kaskade (d. h. Hintereinanderschaltung) von reversiblen Gattern realisieren. Dies führt dazu, dass sich die Synthese reversibler Schaltkreise deutlich von denen traditioneller Chips unterscheidet.

Die vorliegende Arbeit gibt einen Überblick über die grundlegenden Konzepte hinter reversibler Logik und beschreibt aktuelle Syntheseansätze. Neben einer Einführung in das Thema (Abschnitt 2) wird dabei konkret gezeigt, wie sich irreversible Funktionen in reversible Spezifikationen einbetten (Abschnitt 3) und anschließend daraus synthetisieren lassen. Dabei werden sowohl exakte Methoden (Abschnitt 4) als auch heuristische Verfahren (Abschnitt 5) vorgestellt. Diese erhalten die Funktion in Form einer Wahrheitstabelle und generieren daraus einen Schaltkreis bestehend aus möglichst wenig Gattern. Da die Verwendung von Wahrheitstabellen aber nur die Synthese von relativ kleinen Funktionen erlaubt, wird anschließend ein Syntheseansatz beschrieben, welcher binäre Entscheidungsdiagramme (engl.: BDDs) verwendet und damit die Synthese deutlich größerer Funktionen ermöglicht (Abschnitt 6). Abschließend werden die jeweiligen Ansätze gegenübergestellt und die Arbeit zusammengefasst.

2 Reversible Schaltkreise

Reversible Schaltkreise [2; 14; 29] realisieren Funktionen $f : \mathbb{B}^n \to \mathbb{B}^n$ über $X = \{x_1, \ldots, x_n\}$, wobei jede mögliche Eingangsbelegung auf eine eindeutige Ausgangsbelegung abgebildet wird (d. h. Bijektionen werden realisiert). Aufgrund der Reversibilität sind Verzweigungen (Fanouts) und Rückkopplungen (z. B. durch FlipFlops) nicht erlaubt [20]. Daher sind alle reversible Schaltkreise Kaskaden (d. h. Hintereinanderschaltungen) von reversiblen Gattern.

Ein reversibles Gatter hat dabei die Form $g(C, T)$, wobei $C = \{x_{i_1}, \ldots, x_{i_k}\} \subset X$ eine Menge von so genannten *Control Lines* und $T = \{x_{j_1}, \ldots, x_{j_l}\} \subset X$ mit $C \cap T = \emptyset$ eine Menge von so genannten *Target Lines* darstellt. Die Menge C kann dabei auch leer sein. Wenn alle Signale der Menge C (d. h. alle *Control Lines*) mit dem Wert 1 belegt sind oder keine *Control Lines* existieren ($C = \emptyset$), wird auf den *Target Lines* die jeweilige Operation des Gatters ausgeführt. Die Werte der übrigen Signale werden ohne Veränderung weitergeleitet.

In den vergangenen Jahren haben sich unter anderem die folgenden reversiblen Gatter durchgesetzt:

Multiple Control Toffoli Gatter [29] besitzen lediglich eine *Target Line* x_j und bilden $(x_1, x_2, \ldots, x_j, \ldots, x_n)$ auf $(x_1, x_2, \ldots, x_{i_1} x_{i_2} \ldots x_{i_k} \oplus x_j, \ldots, x_n)$ ab. Das heißt, hier wird die *Target Line* invertiert, wenn alle *Control Lines* den Wert 1 haben oder keine *Control Line* existiert.

Multiple Control Fredkin Gatter [8] besitzen zwei *Target Lines* x_{j_1} und x_{j_2}. Die Werte dieser *Target Lines* werden vertauscht, wenn alle *Control Lines* den Wert 1 besitzen oder keine *Control Line* existiert.

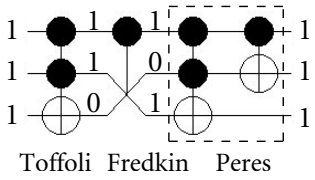


Bild 1 Reversible Gatter.

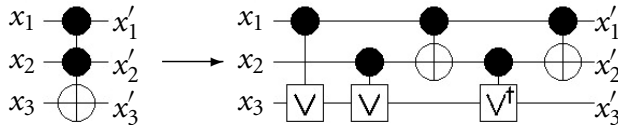


Bild 2 Umformung eines Toffoli Gatters in eine Kaskade von Quantengattern.

Peres Gatter (P) [22] haben eine *Control Line* x_i und zwei *Target Lines* x_{j_1} sowie x_{j_2}. Sie bilden $(x_1, x_2, \ldots, x_{j_1}, \ldots, x_{j_2} \ldots, x_n)$ auf $(x_1, x_2, \ldots, x_i \oplus x_{j_1}, \ldots, x_i x_{j_1} \oplus x_{j_2}, \ldots, x_n)$ ab.

Bild 1 zeigt ein Toffoli Gatter, ein Fredkin Gatter und ein Peres Gatter in einer Kaskade. Die *Control Lines* werden dabei durch ●, die *Target Line(s)* durch ⊕ dargestellt (mit Ausnahme des Fredkin Gatters, wo stattdessen ein × verwendet wird). Die annotierten Werte demonstrieren eine mögliche Berechnung dieser Kaskade. Wie man sehen kann, ist diese Berechnung in beide Richtungen (d. h. reversibel) möglich.

Die jeweiligen Gattertypen sind dabei universell, das heißt mit den einzelnen Typen lassen sich alle möglichen reversiblen Funktionen realisieren. Allerdings unterscheiden sich die einzelnen Gatter in den dafür nötigen Kosten, welche je nach Anwendung bzw. Technologie anfallen. Im Gebiet der Quantencomputer wurde bereits ein konkretes Maß eingeführt: die so genannten *Quantenkosten* [1; 15; 17]. Diese geben an, wie viele Quantenoperationen nötig sind, um einen reversiblen Schaltkreis als Quantenschaltkreis zu realisieren. Da – wie oben bereits erwähnt – Quantenschaltkreise reversibel sind, lassen sich entsprechende Beschreibungen sehr leicht aus reversiblen Schaltkreisen erzeugen. Bild 2 zeigt zum Beispiel die Umformung eines Toffoli Gatters mit zwei Control Lines in eine Kaskade von Quantengattern[1]. Hierfür sind insgesamt fünf Quantengatter nötig – daher verursacht ein solches Gatter Quantenkosten von 5.

Tabelle 1 gibt die Quantenkosten für eine Auswahl von Toffoli Gattern unterschiedlicher Größe an. Erkennbar ist dabei, dass insbesondere die Anzahl der *Control Lines* sowie der Anzahl freier (das heißt weder als *Control Lines* noch als *Target Lines* genutzter) Signale die Höhe der Kosten maßgeblich beeinflusst. Je nach Anwendungsgebiet sollten diese Kosten bei der Synthese von reversibler Logik berücksichtigt werden.

[1] Quantengatter bestehen ebenfalls aus Control Lines und Target Lines, weshalb eine ähnliche Notation wie für reversible Schaltkreise verwendet wird. Da in dieser Arbeit allerdings reversible Schaltkreise im Vordergrund stehen, wird auf eine genauere Definition der einzelnen Quantenoperatoren verzichtet. Weitere Details dazu finden sich u. a. in [20].

Tabelle 1 Quantenkosten für Toffoli Gatter.

Anzahl Control Lines	Quantenkosten
0	1
1	1
2	5
3	13
4	26, wenn ≥ 2 Leitungen nicht verbunden sind
4	29, sonst
5	38, wenn ≥ 3 Leitungen nicht verbunden sind
5	52, wenn 1 oder 2 Leitungen nicht verbunden sind
5	61, sonst
6	50, wenn ≥ 4 Leitungen nicht verbunden sind
6	80, wenn 1, 2 oder 3 Leitungen nicht verbunden sind
6	125, sonst

In den folgenden Abschnitten dieser Arbeit wird nun insbesondere auf die Synthese von Schaltkreisen bestehend aus Toffoli Gattern eingegangen. Dabei ist es das Ziel, Schaltkreise zu generieren, welche die gegebene Funktion mit möglichst wenig Gattern erzeugt. Da darüber hinaus – je nach Technologieabbildung – Quantenkosten einen immer größeren Einfluss gewinnen, werden diese in den folgenden Abschnitten ebenfalls diskutiert. Die jeweiligen Verfahren werden dabei am Beispiel eines 1-Bit Addierers veranschaulicht.

3 Einbetten irreversibler Funktionen

Tabelle 2 zeigt die Wahrheitstabelle eines 1-Bit Addierers. Dieser besitzt drei Eingänge (ein Übertragsbit c_{in} sowie die beiden Summanden x and y) und zwei Ausgänge (das Übertragsbit c_{out} und die Summe *sum*). Damit ist der Addierer klar erkennbar irreversibel, da (1) die Anzahl an Eingängen nicht mit der Anzahl an Ausgängen übereinstimmt und (2) die Funktion keine bijektive Abbildung darstellt.

Selbst das Hinzufügen eines weiteren Ausgangs (was zu einer identischen Anzahl an Eingängen und Ausgängen führt) würde diese Funktion nicht reversibel machen. Damit erreicht man lediglich, dass o. B. d. A. die ersten vier Zeilen der Wahrheitstabelle eindeutig abgebildet wer-

Tabelle 2 Wahrheitstabelle des 1-Bit Addierers.

c_{in}	x	y	c_{out}	*sum*	
0	0	0	0	0	0
0	0	1	**0**	**1**	**0**
0	1	0	**0**	**1**	**1**
0	1	1	*1*	*0*	*0*
1	0	0	**0**	**1**	**?**
1	0	1	*1*	*0*	*1*
1	1	0	*1*	*0*	*?*
1	1	1	1	1	1

Tabelle 3 Wahrheitstabelle des eingebetteten Addierers.

0	c_{in}	x	y	c_{out}	sum	g_1	g_2
0	**0**	**0**	**0**	**0**	**0**	0	0
0	**0**	**0**	**1**	**0**	**1**	1	1
0	**0**	**1**	**0**	**0**	**1**	1	0
0	**0**	**1**	**1**	**1**	**0**	0	1
0	**1**	**0**	**0**	**0**	**1**	0	0
0	**1**	**0**	**1**	**1**	**0**	1	1
0	**1**	**1**	**0**	**1**	**0**	1	0
0	**1**	**1**	**1**	**1**	**1**	0	1
1	0	0	0	1	0	0	0
1	0	0	1	1	1	1	1
1	0	1	0	1	1	1	0
1	0	1	1	0	0	0	1
1	1	0	0	1	1	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	0	1	0
1	1	1	1	0	1	0	1

den (siehe rechte Spalte in Tabelle 2). Spätestens für die fünfte Zeile ist dies aber nicht mehr möglich, da $c_{out} = 0$ and $sum = 1$ bereits zweimal vorhanden waren und nicht mehr länger eindeutig abgebildet werden können.

Dies führt dazu, dass – um irreversible Funktionen einzubetten – unter Umständen zusätzliche Ausgänge nötig werden. Konkret müssen mindestens $\lceil log(m) \rceil$ Ausgänge hinzugefügt werden, wobei m die maximale Anzahl an gleichen Ausgangsmustern auf der rechten Seite der Wahrheitstabelle angibt. Angewandt auf den Addierer, bei welchem sich bis zu 3 Ausgangsmuster wiederholen, sind also $\lceil log(3) \rceil = 2$ zusätzliche Ausgänge (und verbunden damit eine weitere Leitung) nötig, um die Funktion reversibel einzubetten. Dies führt dabei zu konstanten Eingängen (die beliebig vom Entwerfer belegt werden können) sowie zu so genannten *Garbage Outputs* und damit zu unvollständig spezifizierten reversiblen Funktionen. Da viele Syntheseansätze aber eine komplett spezifizierte Funktion als Eingabe erwarten, sollten die nicht-spezifizierten Ausgaben schließlich noch mit konkreten Werten belegt werden. Eine mögliche Einbettung des 1-Bit Addierers ist in Tabelle 3 dargestellt. Diese Funktion lässt sich nun mit den in den folgenden Abschnitten beschriebenen Verfahren automatisch synthetisieren.

4 Exakte Synthese

Exakte Synthesealgorithmen generieren einen *minimalen* Schaltkreis für eine gegebene Funktion. Das Sicherstellen der Minimalität erfordert allerdings einen enormen Rechenaufwand, weshalb sich entsprechende Methoden oft nur auf sehr kleine Funktionen anwenden lassen. Trotzdem ist es sinnvoll, exakte Verfahren zu betrachten, da sie (1) die Synthese kleinerer Realisierungen als die derzeit bekannten erlauben, (2) die Evaluation von heuristischen Methoden (zumindest für kleine Funktionen) ermöglichen bzw. (3) für die Generierung von Basisbausteinen verwendet werden können.

Dieser Abschnitt beschreibt das exakte Verfahren, wie es in [10] vorgestellt wurde[2]. Hier wurde das Syntheseproblem als eine Sequenz von Entscheidungsproblemen (genauer: von Erfüllbarkeitsproblemen; kurz: SAT-Problemen) beschrieben. Für eine gegebene Funktion $f : \mathbb{B}^n \to \mathbb{B}^n$ wird geprüft, ob sie sich mit $d = 1$ Toffoli Gattern realisieren lässt. Ist dies nicht möglich, wird d iterativ erhöht bis schließlich eine Realisierung ermittelt werden kann. Für die jeweiligen Prüfungen wurde das Syntheseproblem als eine Instanz des Erfüllbarkeitsproblem (SAT-Instanz) kodiert und anschließend mit einem SAT-Beweiser gelöst.

Die konkrete Kodierung der SAT-Instanz wird im folgenden für den Addierer aus Tabelle 3 sowie $d = 4$ beschrieben. Ziel ist es, eine Instanz zu kodieren, welche erfüllbar ist, wenn ein Schaltkreis mit $d = 4$ Gattern die zu synthetisierende Funktion (d. h. den Addierer) realisiert. Zu diesem Zweck werden die folgenden Booleschen Variablen und Bedingungen generiert:

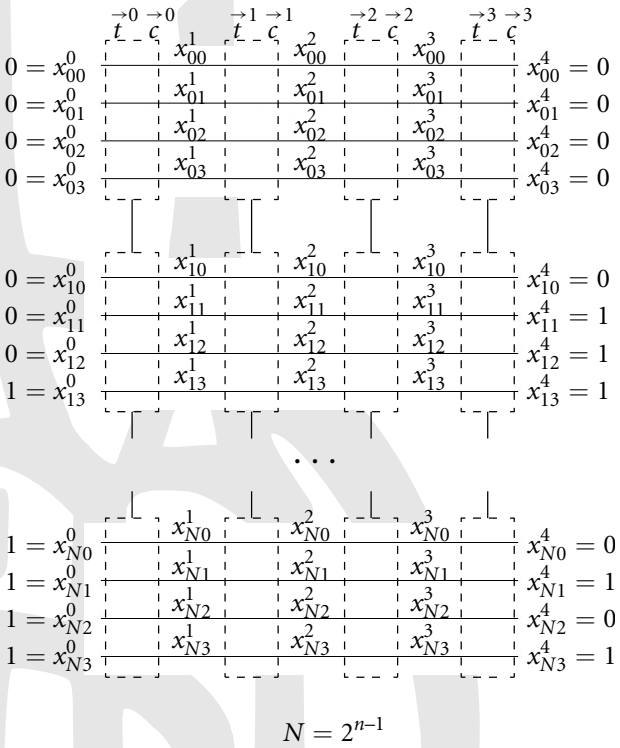


Bild 3 Exakte SAT Formulierung für $d = 4$.

Zuerst werden Variablen $x^k_{i0}, x^k_{i1}, \ldots, x^k_{in-1}$ eingeführt, welche die Eingänge (für $k = 0$), die Ausgänge (für $k = d$) sowie die internen Signale (für $1 \leq k \leq d-1$) des Schaltkreises für jede Zeile i der Wahrheitstabelle repräsentieren. Die linke Seite der Wahrheitstabelle korrespondiert somit zu $x^0_{i0}, x^0_{i1}, \ldots, x^0_{in-1}$, während die rechte Seite zu $x^d_{i0}, x^d_{i1}, \ldots, x^d_{in-1}$ korrespondiert. Bild 3 zeigt die entsprechenden Variablen für das Addierer-Beispiel.

[2] Dabei wurden Schaltkreise mit minimaler Anzahl reversibler Gatter synthetisiert. Ähnliche Verfahren existieren aber auch zur Minimierung der Quantenkosten (z. B. [9; 12]).

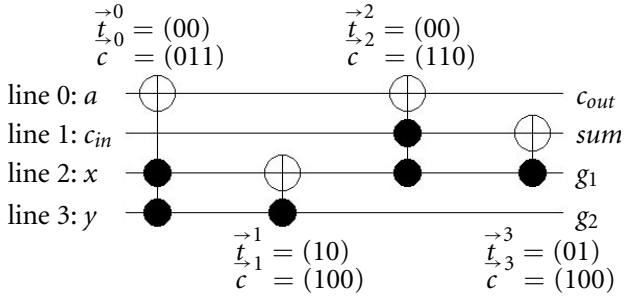


Bild 4 Schaltkreis generiert mit Hilfe der exakten Synthese.

Danach werden Variablen $t^k_{\lceil \log_2 n \rceil}, t^k_{\lceil \log_2 n \rceil - 1}, \ldots, t^k_1$ und $c^k_1, c^k_2, \ldots, c^k_{n-1}$ mit $0 \leq k < d$ eingeführt. Diese repräsentieren den Typ der jeweiligen Toffoli Gatter an Stelle k (der Übersichtlichkeit halber werden diese Variablen im Folgenden mit $\vec{t}^k$ and $\vec{c}^k$ abgekürzt). Dabei stellt $\vec{t}^k$ die binäre Kodierung einer natürlichen Zahl $t^k \in \{0, \ldots, n-1\}$ dar, welche die Position der *Target Line* für das betreffende Toffoli Gatter angibt. Die Variablen $\vec{c}^k$ geben Positionen der *Control Lines* an. So wird bei einer Belegung $\vec{t}^k = (0010)$ angenommen, dass sich die *Target Line* des k-ten Gatters auf Leitung $t^k = 2$ befindet. Eine Belegung $c^k_l = 1$ mit $(1 \leq l \leq n-1)$ führt dazu, dass Leitung $(t^k + l) \mod n$ als *Control Line* für das k-te Gatter angenommen wird. Bild 4 zeigt einige mögliche Belegungen von $\vec{t}^k$ und $\vec{c}^k$ zusammen mit den resultierenden Toffoli Gattern.

Schließlich werden Bedingungen generiert, welche die Kodierung des Syntheseproblems komplettieren. So werden *Constraints* generiert, welche die Variablen x^0_{ij} und x^d_{ij} entsprechend der Eingänge bzw. Ausgänge der Wahrheitstabelle belegen (siehe linke und rechte Seite von Bild 3). Darüber hinaus werden für jedes der k Gatter funktionale Bedingungen erzeugt, welche – abhängig von der Belegung von $\vec{t}^k$ und $\vec{c}^k$ sowie des Gattereingangs x^k_{ij} – den entsprechenden Gatterausgang x^{k+1}_{ij} berechnen. So sei zum Beispiel $\vec{t}^k = (01)$ und $\vec{c}^k = (001)$ (d. h. $c^k_3 = 1$). Diese Belegung führt dazu, dass für das k-te Toffoli Gatter die *Target Line* an Leitung $t^k = (01) = 1$ und eine *Control Line* an Leitung $(t^k + l) \mod n = (1+3) \mod 4 = 0$ angenommen wird. Für diesen Fall werden die folgenden *Constraints* für jede Zeile i der Wahrheitstabelle hinzugefügt:

$$\vec{t}^k = (01) \wedge \vec{c}^k = (001) \Rightarrow$$

$$\begin{aligned} & x^{k+1}_{i0} == x^k_{i0} \\ \wedge\ & x^{k+1}_{i1} == x^k_{i1} \oplus x^k_{i0} \\ \wedge\ & x^{k+1}_{i2} == x^k_{i2} \\ \wedge\ & x^{k+1}_{i3} == x^k_{i3} \end{aligned}$$

Das bedeutet, dass die Werte der Leitungen 0, 2, and 3 unverändert bleiben, während der Ausgang von Leitung 1 invertiert wird, wenn Leitung 0 mit dem Wert 1 belegt ist. Entsprechend werden analoge Bedingungen für die restlichen Fälle hinzugefügt. Als Ergebnis erhält man eine SAT-Instanz, welche erfüllbar ist, wenn eine gültige Belegung der Variablen $\vec{t}^k$ and $\vec{c}^k$ existiert, sodass für alle Zeilen der Wahrheitstabelle die gewünschte Eingangs-Ausgangs-Abbildung erreicht wird. In diesem Fall kann der resultierende Toffoli Schaltkreis anhand der Belegungen von $\vec{t}^k$ and $\vec{c}^k$ ermittelt werden (siehe Bild 4). Konnte der SAT-Beweiser keine solche Lösung ermitteln, wurde gezeigt, dass kein Toffoli Schaltkreis mit d die gewünschte Funktion realisiert.

Für den betrachteten Addierer wurden keine Lösungen für $d = 1$, $d = 2$ und $d = 3$ gefunden. Dagegen lieferte die Prüfung für $d = 4$ eine erfüllende Belegung, welche zu dem Schaltkreis in Bild 4 führte. Damit besitzt die minimale Realisierung eines Addierers vier Gatter. Der resultierende Schaltkreis hat Quantenkosten von 12.

5 Heuristische Synthese

Heuristische Syntheseverfahren können zwar keine Minimalität generieren, dafür aber mit größeren Funktionen umgehen. In der Vergangenheit wurden einige solcher Verfahren vorgestellt [11; 13; 15; 16; 19; 25–27]. In diesem Abschnitt beschreiben wir den Ansatz von [19] (auch bekannt als *MDM-Synthese*), dessen prinzipielle Vorgehensweise von vielen nachfolgenden Verfahren wiederverwendet wurde. Die grundlegende Idee des Ansatzes besteht darin, jede Zeile der Wahrheitstabelle zu traversieren und so lange Gatter dem Schaltkreis hinzuzufügen, bis die Ausgangswerte auf die Eingangswerte abgebildet wurden. Dabei werden die Gatter jeweils so gewählt, dass sie keine bereits durchlaufenen Zeilen der Wahrheitstabelle mehr beeinflussen. Außerdem werden die Gatter jeweils auf der Ausgangsseite des Schaltkreises hinzugefügt, da die Ausgangswerte auf die Eingangswerte abgebildet werden.

Im Folgenden wird die Vorgehensweise mit Hilfe von Tabelle 4 erläutert. Die erste Spalte gibt dabei die jeweilige Zeilennummer der Wahrheitstabelle an, während die zweite und dritte Spalte die Eingangs- und Ausgangsbelegung des 1-Bit Addierers von Tabelle 3 enthält. Aus Platzgründen wurden die Ein- bzw. Ausgänge mit a, b, c und d beschriftet. Die restlichen Spalten geben die transformierten Ausgangswerte für jeden einzelnen Schritt an.

Das Verfahren startet in Zeile 0. Hier sind bereits Eingangs- und Ausgangswerte gleich (beide sind mit 0000 belegt), sodass sofort zu Zeile 1 gesprungen wird. Hier müssen die Werte für c und b invertiert werden, um die Identität zu erreichen. Dazu werden zwei Gatter $TOF(\{d\}, c)$ (1. Schritt) und $TOF(\{d\}, b)$ (2. Schritt) wie in Bild 5 gezeigt, hinzugefügt. Durch die *Control Line* in Leitung d, beeinflussen diese Gatter nicht die vorherige Zeile in der Wahrheitstabelle. Für die Zeilen 2 und 3 werden die Gatter $TOF(\{c\}, b)$ sowie $TOF(\{c, d\}, a)$ benötigt um die Werte von b und a anzugleichen (Schritte 3 und 4). Dabei sind im letzteren Fall nun zwei *Control Lines* nötig um vorherige Zeilen nicht zu beeinflussen. Nach diesem Verfahren werden noch zwei weitere Gatter $TOF(\{d, b\}, a)$ (Schritt 5) und $TOF(\{c, b\}, a)$ (Schritt 6)

Tabelle 4 MDM Verfahren.

Zeile (i)	Eingang *abcd*	Ausgang *abcd*	1. Schritt *abcd*	2. Schritt *abcd*	3. Schritt *abcd*	4. Schritt *abcd*	5. Schritt *abcd*	6. Schritt *abcd*
0	0000	0000	0000	0000	0000	0000	0000	0000
1	0001	0111	0101	0001	0001	0001	0001	0001
2	0010	0110	0110	0110	0010	0010	0010	0010
3	0011	1001	1011	1111	1011	0011	0011	0011
4	0100	0100	0100	0100	0100	0100	0100	0100
5	0101	1011	1001	1101	1101	1101	0101	0101
6	0110	1010	1010	1010	1110	1110	1110	0110
7	0111	1101	1111	1011	1111	0111	1111	0111
8	1000	1000	1000	1000	1000	1000	1000	1000
9	1001	1111	1101	1001	1001	1001	1001	1001
10	1010	1110	1110	1110	1010	1010	1010	1010
11	1011	0001	0011	0111	0011	1011	1011	1011
12	1100	1100	1100	1100	1100	1100	1100	1100
13	1101	0011	0001	0101	0101	0101	1101	1101
14	1110	0010	0010	0010	0110	0110	0110	1110
15	1111	0101	0111	0011	0111	1111	0111	1111

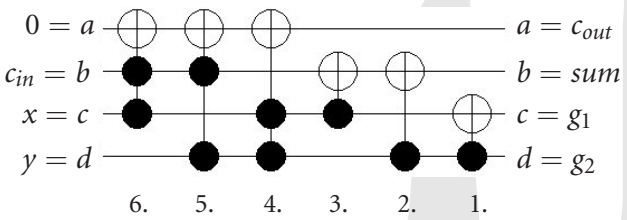


Bild 5 Schaltkreis generiert mit Hilfe der MDM-Synthese.

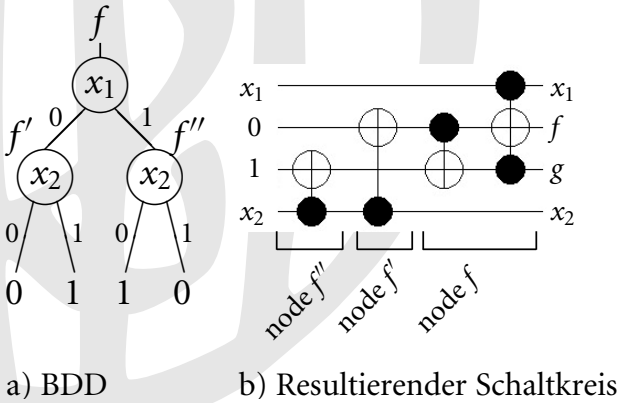
hinzugefügt, bis schließlich die Identität der Eingaben und Ausgaben erreicht wurde. Der resultierende Schaltkreis ist in Bild 5 dargestellt. Er besteht aus sechs Gattern und hat Quantenkosten von 18.

In [19] wurden darüber hinaus noch weitere Varianten dieses Verfahrens diskutiert. So kann die Wahrheitstabelle auch umgekehrt durchlaufen werden (das heißt Gatter werden hinzugefügt, sodass die Eingangswerte auf die Ausgangswerte abgebildet werden). Auch gleichzeitiges Durchlaufen von beiden Richtungen ist möglich. Zudem wurde der Ansatz in [16] noch durch ein so genanntes *Template Matching* verbessert. Dabei werden bestimmte Schaltkreisteile durch äquivalente kleinere Kaskaden (so genannten *Templates*) ersetzt, sodass die Kosten der teils sehr großen Schaltkreise nochmal deutlich gesenkt werden.

6 BDD-basierte Synthese

Die bisher vorgestellten Syntheseansätze erhalten als Eingabe jeweils die zu synthetisierende Funktion in Form einer Wahrheitstabelle. Damit sind sie aber nur für kleine Funktionen anwendbar. Aus diesem Grund wird in diesem Abschnitt schließlich noch ein weiteres (heuristisches) Verfahren vorgestellt, welches auf den Konzepten von [33] beruht. Dabei werden Schaltkreise mit Hilfe von *binären Entscheidungsdiagrammen* [4] (engl.: BDDs) erstellt. Ein BDD ist ein direkter, azyklischer Graph $G = (V, E)$ bestehend aus terminalen und nicht-terminalen Knoten, welcher eine Boolesche Funktion f repräsentiert. Jeder nicht-terminale Knoten $v \in V$ ist zudem mit einer *select-Variablen* x_i aus der Definitionsmenge von f beschriftet. Mit Hilfe der Shannon Dekomposition ($f = \overline{x}_i f_{x_i=0} + x_i f_{x_i=1}$) wird nun die Funktion für jeden nicht-terminalen Knoten $v \in V$ in zwei *Kofaktoren* $f_{x_i=0}$ und $f_{x_i=1}$ zerlegt. Ein Knoten, welcher den Kofaktor $f_{x_i=0}$ ($f_{x_i=1}$) repräsentiert, wird dabei mit $low(v)$ ($high(v)$) bezeichnet. Besteht ein Kofaktor $f_{x_i=0}$ ($f_{x_i=1}$) nur noch aus einer Booleschen Konstante wird diese mit einem terminalen Knoten repräsentiert.

Bild 6a zeigt beispielhaft einen BDD, welcher die Funktion $f = x_1 \oplus x_2$ repräsentiert. Nach der Zerlegung im Wurzelknoten, entstehen die beiden Kofaktoren $f_{x_1=0} = x_2$ und $f_{x_1=1} = \overline{x}_2$, welche durch die darunter liegenden Knoten (markiert mit f' bzw. f'') repräsentiert werden. In den letzten Jahren, haben sich BDDs als effiziente Datenstruktur für die Repräsentation von großen Booleschen Funktionen etabliert. Dabei geht man im Allgemeinen von *reduzierten* BDDs aus, welche durch die Anwendung von Reduktionsregeln [4] erzeugt wer-

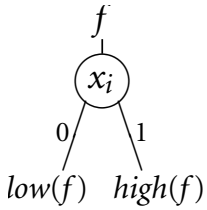
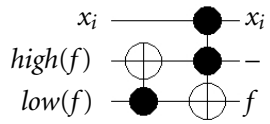
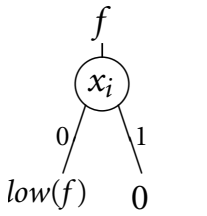
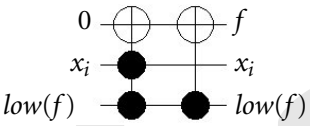
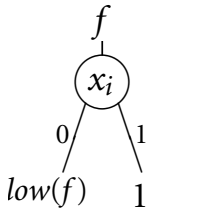
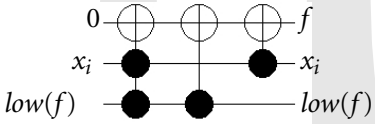
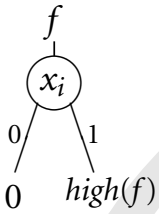
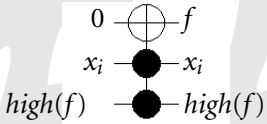
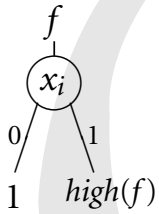
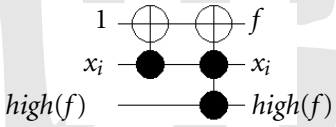
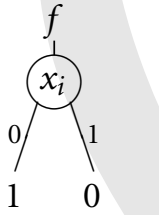
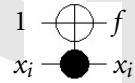
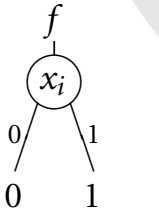
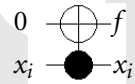


a) BDD b) Resultierender Schaltkreis

Bild 6 BDD and Toffoli Schaltkreis für $f = x_1 \oplus x_2$.

den können. Diese kompakte Funktionsrepräsentation erlaubt die Synthese von Funktionen mit mehr als hundert Variablen.

Tabelle 5 Ersetzen von BDD-Knoten durch reversible Schaltkreise.

BDD	Toffoli Circuit
	
	
	
	
	
	
	

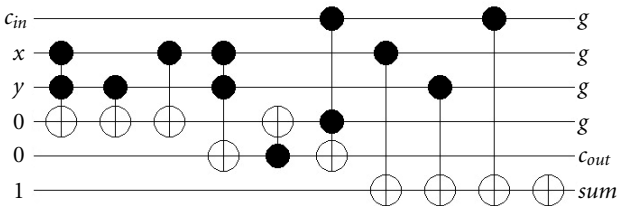


Bild 7 Schaltkreis generiert mit Hilfe der BDD-Synthese.

Sei $G = (V, E)$ ein BDD, welcher die Funktion f repräsentiert. Ein reversibler Schaltkreis dieser Funktion lässt sich dann erzeugen, indem man den BDD komplett traversiert und jeden Knoten $v \in V$ durch eine Kaskade von reversiblen Gattern ersetzt. Die jeweilige Kaskade hängt dabei von den Nachfolgern des Knotens v ab. Tabelle 5 zeigt die entsprechenden Ersetzung für alle möglichen Szenarien. Dabei ist zu beachten, dass eine zusätzliche Leitung nötig wird, wenn einer der Kofaktoren einen konstanten Wert besitzt (d. h. einer der Nachfolger des Knotens zu einem Terminal führt). Andererseits ist kein reversibles Einbetten der entsprechenden Funktionalität möglich (siehe hierzu auch Abschnitt 3).

Mit diesen Ersetzungen als Basis, kann ein entsprechendes Syntheseverfahren recht einfach mit zwei Schritten formuliert werden: Zuerst wird ein BDD für die Funktion f generiert. Dafür stehen bereits sehr effiziente Software-Pakete zur Verfügung (z. B. CUDD [28]). Anschließend wird der resultierende BDD mit einer Tiefensuche durchlaufen und für jeden Knoten die entsprechende Ersetzung durchgeführt. Dies ist linear in der Anzahl der Knoten des BDDs. Bild 6 veranschaulicht die entsprechenden Ersetzungen.

Als Konsequenz sind die resultierenden Schaltkreise in ihrer Größe durch den BDD begrenzt. Konkret bedeutet dies, dass die generierten Schaltkreise nie mehr als $3 \cdot |V|$ Gatter besitzen, wobei $|V|$ die Anzahl der Knoten des BDDs darstellt. Dies ist insofern interessant, als dass für BDDs bereits sehr viele theoretische Ergebnisse existieren, welche nun direkt auf reversible Logik übertragen werden können. Darüber hinaus ist es durch dieses Verfahren möglich, Funktionen mit mehr als hundert Variablen effizient zu synthetisieren.

Bild 7 zeigt schließlich den resultierenden Schaltkreis für das in dieser Arbeit näher betrachtete Addierer Beispiel (mit verfeinerten Ersetzungen für *Shared Nodes* [4] und Komplementkanten [3]). Das Resultat besteht aus zehn Gattern und hat Quantenkosten von 22. Außerdem ist im Gegensatz zu den bisher betrachteten Realisierungen eine weitere Leitung nötig. Insofern ist das Ergebnis der BDD-basierten Synthese leicht schlechter als bei den vorherigen Ansätzen – dafür ist sie aber nicht auf kleine Funktionen beschränkt.

7 Zusammenfassung

In der vorliegenden Arbeit wurde die Synthese von reversiblen Schaltkreisen näher betrachtet. Nach einer Einführung in das Thema wurde gezeigt, wie sich ir-

Tabelle 6 Zusammenfassung.

	Schaltkreisgröße		Skalierbarkeit
	Gatteranzahl	Anzahl Leitungen	
Exakter Ansatz	++	++	−−
MDM-Ansatz	○	++	○
BDD-Ansatz	−	−−	++

reversible Funktionen einbetten und anschließend mit Hilfe von exakten und heuristischen Verfahren synthetisieren lassen. Neben etablierten Techniken wurde dabei ein aktueller Ansatz vorgestellt, welcher BDDs ausnutzt und damit die Synthese signifikant größerer Funktionen erlaubt.

Die Vorzüge (+) und Nachteile (–) der entsprechenden Verfahren sind in Tabelle 6 nochmal kurz zusammengefasst. So garantiert die exakte Synthese optimale Resultate, ist dafür aber nur für relativ kleine Funktionen anwendbar. Etwas größere Funktionen lassen sich dagegen mit dem MDM-Ansatz synthetisieren – ohne allerdings die Minimalität sicherzustellen. Bzgl. der Leitungsanzahl sind beide Verfahren minimal, da die Funktion vorher mit minimaler Anzahl zusätzlicher Leitungen eingebettet werden kann. Im Gegensatz dazu führt der BDD-basierte Ansatz oft zu größeren Resultaten (bzgl. Gatteranzahl und Anzahl an Leitungen). Dafür kann dieser aber deutlich größere Funktionen synthetisieren. Die resultierenden Schaltkreise der einzelnen Verfahren stehen online auf RevLib [37] zur Verfügung.

Danksagung

Teile der Arbeit entstanden im Rahmen des Projektbezogenen Personenaustauschs (PPP) des Deutschen Akadamischen Austauschdienstes (DAAD). Darüber hinaus möchten wir uns für fruchtvolle Diskussionen bei Gerhard W. Dueck (University of New Brunswick, Kanada), Daniel Große (Universität Bremen, Deutschland) und D. Michael Miller (University of Victoria, Kanada) bedanken. Den anonymen GutachterInnen sei für die zahlreichen konstruktiven und hilfreichen Anmerkungen gedankt, durch welche die Lesbarkeit der vorliegenden Arbeit deutlich verbessert werden konnte.

Literatur

[1] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVinchenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *The American Physical Society*, 52:3457–3467, 1995.

[2] C. H. Bennett. Logical reversibility of computation. *IBM J. Res. Dev*, 17(6):525–532, 1973.

[3] K. Brace, R. Rudell, and R. Bryant. Efficient implementation of a BDD package. In *Design Automation Conf.*, pages 40–45, 1990.

[4] R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[5] R. Cuykendall and D. R. Andersen. Reversible optical computing circuits. *Optics Letters*, 12(7):542–544, 1987.

[6] B. Desoete and A. de Vos. A reversible carry-look-ahead adder using control gates. *INTEGRATION, the VLSI Jour.*, 33(1–2):89–104, 2002.

[7] D. Y. Feinstein, M. A. Thornton, and D. M. Miller. Partially redundant logic detection using symbolic equivalence checking in reversible and irreversible logic circuits. In *Design, Automation and Test in Europe*, pages 1378–1381, 2008.

[8] E. F. Fredkin and T. Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21(3/4):219–253, 1982.

[9] D. Große, R. Wille, G. W. Dueck, and R. Drechsler. Exact synthesis of elementary quantum gate circuits for reversible functions with don't cares. In *Int'l Symp. on Multi-Valued Logic*, pages 214–219, 2008.

[10] D. Große, R. Wille, G. W. Dueck, and R. Drechsler. Exact multiple control Toffoli network synthesis with SAT techniques. *IEEE Trans. on CAD*, 28(5):703–715, 2009.

[11] P. Gupta, A. Agrawal, and N. Jha. An algorithm for synthesis of reversible logic circuits. *IEEE Trans. on CAD*, 25(11):2317–2330, 2006.

[12] 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*, 25(9):1652–1663, 2006.

[13] P. Kerntopf. A new heuristic algorithm for reversible logic synthesis. In *Design Automation Conf.*, pages 834–837, 2004.

[14] R. Landauer. Irreversibility and heat generation in the computing process. *IBM J. Res. Dev.*, 5:183, 1961.

[15] D. Maslov and G. W. Dueck. Improved quantum cost for n-bit Toffoli gates. *IEE Electronics Letters*, 39:1790, 2004.

[16] D. Maslov, G. W. Dueck, and D. M. Miller. Toffoli network synthesis with templates. *IEEE Trans. on CAD*, 24(6):807–817, 2005.

[17] D. Maslov, C. Young, G. W. Dueck, and D. M. Miller. Quantum circuit simplification using templates. In *Design, Automation and Test in Europe*, pages 1208–1213, 2005.

[18] R. C. Merkle. Reversible electronic logic using switches. *Nanotechnology*, 4:21–40, 1993.

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

[20] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge Univ. Press, 2000.

[21] K. N. Patel, J. P. Hayes, and I. L. Markov. Fault testing for reversible circuits. *IEEE Trans. on CAD*, 23(8):1220–1230, 2004.

[22] A. Peres. Reversible logic and quantum computers. *Phys. Rev. A*, (32):3266–3276, 1985.

[23] M. Perkowski, J. Biamonte, and M. Lukac. Test generation and fault localization for quantum circuits. In *Int'l Symp. on Multi-Valued Logic*, pages 62–68, 2005.

[24] I. Polian, T. Fiehn, B. Becker, and J. P. Hayes. A family of logical fault models for reversible circuits. In *Asian Test Symp.*, pages 422–427, 2005.

[25] M. Saeedi, M. Sedighi, and M. S. Zamani. A novel synthesis algorithm for reversible circuits. In *Int'l Conf. on CAD*, pages 65–68, 2007.

[26] V. V. Shende, S. S. Bullock, and I. L. Markov. Synthesis of quantum-logic circuits. *IEEE Trans. on CAD*, 25(6):1000–1010, 2006.

[27] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes. Synthesis of reversible logic circuits. *IEEE Trans. on CAD*, 22(6):710–722, 2003.

[28] F. Somenzi. *CUDD: CU Decision Diagram Package Release 2.3.1*. University of Colorado at Boulder, 2001.

[29] T. Toffoli. Reversible computing. In W. de Bakker and J. van Leeuwen, editors, *Automata, Languages and Programming*, page 632. Springer, 1980. Technical Memo MIT/LCS/TM-151, MIT Lab. for Comput. Sci.

[30] G. F. Viamontes, I. L. Markov, and J. P. Hayes. Checking equivalence of quantum circuits and states. In *Int'l Conf. on CAD*, pages 69–74, 2007.
[31] L. M. K. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. H. Sherwood, and I. L. Chuang. Experimental realization of shor's quantum factoring algorithm using nuclear magnetic resonance. *Nature*, 414:883, 2001.
[32] S.-A. Wang, C.-Y. Lu, I.-M. Tsai, and S.-Y. Kuo. An XQDD-based verification method for quantum circuits. *IEICE Transactions*, 91-A(2):584–594, 2008.
[33] R. Wille and R. Drechsler. BDD-based synthesis of reversible logic for large functions. In *Design Automation Conf.*, 2009.
[34] R. Wille, D. Große, G. W. Dueck, and R. Drechsler. Reversible logic synthesis with output permutation. In *VLSI Design*, 2009.
[35] R. Wille, D. Große, S. Frehse, G. W. Dueck, and R. Drechsler. Debugging of Toffoli networks. In *Design, Automation and Test in Europe*, 2009.
[36] R. Wille, D. Große, D. M. Miller, and R. Drechsler. Equivalence checking of reversible circuits. In *Int'l Symp. on Multi-Valued Logic*, 2009.
[37] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler. RevLib: an online resource for reversible functions and reversible circuits. In *Int'l Symp. on Multi-Valued Logic*, pages 220–225. RevLib is available at http://www.revlib.org.
[38] V. V. Zhirnov, R. K. Cavin, J. A. Hutchby, and G. I. Bourianoff. Limits to binary logic switch scaling – a gedanken model. *Proc. of the IEEE*, 91(11):1934–1939, 2003.
[39] J. Zhong and J. Muzio. Using crosspoint faults in simplifying Toffoli networks. In *IEEE North-East Workshop on Circuits and Systems*, pages 129–132, 2006.

Manuskripteingang: 26. Juni 2009

Dr. Robert Wille studierte Informatik an der Universität Bremen, wo er 2006 sein Diplom erhielt und 2009 seine Promotion abschloss. Derzeit arbeitet er als wissenschaftlicher Mitarbeiter der Arbeitsgruppe Rechnerarchitektur weiter an der Universität Bremen. Seine wissenschaftlichen Interessen umfassen den Schaltkreisentwurf von reversibler Logik, die Entwicklung von Algorithmen zur Lösung Boolescher Erfüllbarkeit sowie die formale Verifikation von Schaltkreisen und Systemen.

Adresse: Universität Bremen, Bibliothekstraße 1, 28359 Bremen, Tel.: +49 421 218 63947, Fax: +49 421 218 7385,
E-Mail: rwille@informatik.uni-bremen.de

Prof. Dr. Rolf Drechsler hat an der Johann-Wolfgang-Goethe Universität in Frankfurt am Main Mathematik und Informatik studiert. Im Jahr 1995 schloss er sein Studium mit der Promotion ab. Von 1995 bis 2000 war er als Wissenschaftlicher Assistent an der Albert-Ludwigs-Universität in Freiburg beschäftigt, wo er 1999 habilitierte. Im Anschluss arbeitete er als Senior Engineer in der Abteilung Coporate Technologies der Siemens AG, München, bis er im Oktober 2001 die Professur für Rechnerarchitektur am Fachbereich 3, Mathematik und Informatik, der Universität Bremen annahm. Seine wissenschaftlichen Interessen umfassen den Schaltkreis- und Systementwurf mit dem Schwerpunkt formale Verifikation.

Adresse: Universität Bremen, Bibliothekstraße 1, 28359 Bremen, Tel.: +49 421 218 63932, Fax: +49 421 218 7385, E-Mail: drechsler@uni-bremen.de