Formal Verification Techniques for Reversible Circuits

Chinmay Avinash Limaye

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
In
Computer Engineering

Michael S. Hsiao, Chair
Joseph G. Tront
Cameron D. Patterson

June 1, 2011
Blacksburg, Virginia

Keywords: Reversible Circuits, Redundancy, Binary Decision Diagram (BDD),
Equivalence Checking

Formal Verification Techniques for Reversible Circuits

Chinmay Limaye

(ABSTRACT)

As the number of transistors per unit chip area increases, the power dissipation of the chip becomes a bottleneck. New nano-technology materials have been proposed as viable alternatives to CMOS to tackle area and power issues. The power consumption can be minimized by the use of reversible logic instead of conventional combinational circuits. Theoretically, reversible circuits do not consume any power (or consume minimal power) when performing computations. This is achieved by avoiding information loss across the circuit. However, use of reversible circuits to implement digital logic requires development of new Electronic Design Automation techniques. Several approaches have been proposed and each method has its own pros and cons. This often results in multiple designs for the same function. Consequently, this demands research in efficient equivalence checking techniques for reversible circuits.

This thesis explores the optimization and equivalence checking of reversible circuits. Most of the existing synthesis techniques work in two steps – generate an original, often sub-optimal, implementation for the circuit followed optimization of this design. This work proposes the use of Binary Decision Diagrams for optimization of reversible circuits. The proposed technique identifies repeated gate (trivial) as well as non-contiguous redundancies in a reversible circuit. Construction of a BDD for a sub-circuit (obtained by sliding a window of fixed size over the circuit) identifies redundant gates based upon the redundant variables in the BDD. This method was unsuccessful in identifying any additional redundancies in benchmark circuits; however, hidden non-

contiguous redundancies were consistently identified for a family of randomly generated reversible circuits. As of now, several research groups focus upon efficient synthesis of reversible circuits. However, little work has been done in identification of redundant gates in existing designs and the proposed peephole optimization method stands among the few known techniques. This method fails to identify redundancies in a few cases indicating the complexity of the problem and the need for further research in this area.

Even for simple logical functions, multiple circuit representations exist which exhibit a large variation in the total number of gates and circuit structure. It may be advantageous to have multiple implementations to provide flexibility in choice of implementation process but it is necessary to validate the functional equivalence of each such design. Equivalence checking for reversible circuits has been researched to some extent and a few pre-processing techniques have been proposed prior to this work. One such technique involves the use of Reversible Miter circuits followed by SAT-solvers to ascertain equivalence. The second half of this work focuses upon the application of the proposed reduction technique to Reversible Miter circuits as a pre-processing step to improve the efficiency of the subsequent SAT-based equivalence checking.

Acknowledgements

I wish to extend my sincere gratitude to my research advisor, Dr. Michael Hsiao for his selfless guidance and support during my research. I was honored to get a chance to work with him and was deeply inspired by his extensive knowledge, dedication and amiable nature. His instruction for both the Testing and Verification courses were instrumental in creating the foundation for this work. Also, I am grateful for his suggestions and pointers during the individual meeting which ensured that my research work stayed on track inspite of other distractions. I would also like to thank Dr. Joseph Tront and Dr. Cameron Patterson for being a part of my research committee.

The time I spent with my lab-mates in the PROACTIVE lab at Virginia Tech was integral in shaping my academic profile as well as my personality. My interactions with all of my co-researchers were very positive and vital in ensuring that I become a valued member of the society.

My fellow researchers in CESCA also deserve a mention for their interest in my research work and inputs which often presented potential solutions by opening up new areas of research.

Chinmay Limaye

June 2011

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With advancement in semiconductor fabrication technology, the number of transistors per chip has increased exponentially over the past few decades. Integrated Circuits with millions of transistors dissipating a few Watts of energy are commonly seen in different electronic devices. Faster clocking rates and high transistor switching rate has led to significant increase in chip power consumption. Also, as transistor sizes approach atomic levels, scaling down of the feature size in CMOS designs may usher in many new problems and challenges. Although the research community is actively seeking the solution to these problems, alternative chip design techniques and new technologies may offer promise in the future.

According to the research done by R. Landauer and C. Bennett [1] [20], power dissipation will theoretically be zero (or be a very small quantity) when there is no data loss in a circuit. This characteristic property is embodied by reversible circuits. Reversible circuits composed of a cascade of reversible gates can be conveniently implemented using quantum gates. A reversible gate is a universal gate, in that every combinational circuit can be represented as a reversible circuit. This is possible through the use of garbage outputs and constant input lines thereby embedding the irreversible combinational gate function into the reversible circuit. Sequential circuits too can be represented using reversible gates with the timing elements added at the circuit boundary (similar to combinational circuits).

1.1 Optimization of Reversible Circuits

Reversible circuits are realized using quantum gates and the cost of each design is based upon the total number of gates in the design and the number of input lines per gate. Most of the existing synthesis methods rely upon the optimization of an initial sub-optimal design and hence investigation of efficient optimization techniques is necessary to ensure an acceptable realization cost. Removal of redundant logic also translates to fewer delay faults and lower possibility of masking of existing faults. Efficient synthesis and optimization techniques for reversible circuits will provide the necessary stepping stone for their adaptation and wide-spread use.

1.2 Equivalence checking for Reversible Circuits

1.2.1 Design Verification and Equivalence checking

Design Verification is an important step in digital circuit design and often proves to the bottleneck in the IC development. Out of the different Design Verification areas, Functional Verification takes most of the time and effort. Equivalence Checking ensures that the design will perform required operations within the given constraints. Equivalence checking is a hard problem since the output must be as expected for all possible input vectors, which might be exponential to the number of inputs. There are two main classes of Equivalence Checking: Simulation-based Equivalence Checking and Formal Equivalence Checking.

Simulation-based Equivalence Checking, though straight-forward, presents several challenges. It requires creation of test benches for various input scenarios. Also, a

functional coverage model is required to determine the sufficiency of the required simulations and specific directs test too might be necessary. Though circuit simulation usually is reasonably fast, it still may not be feasible to exhaustively simulate all input vectors for each scenario. Hence, symbolic simulation is needed to reduce the number of vectors needed. The creation of such symbolic test suite is also of research interest and bears some similarities to formal techniques, although not investigated in this thesis.

Formal Equivalence Checking (FEC), on the other hand, is a systematic process that uses mathematical reasoning to verify that the two designs are equivalent. FEC can overcome most of the above limitations of simulation by implicitly exhausting all possible input values. Modern day FEC techniques are based around Binary Decision Diagrams (BDDs) and Boolean Satisfiability (SAT) solvers. Extensive research has been conducted in these areas to provide high performance tools for determining the equivalence of given circuits.

1.2.2 Equivalence checking for reversible circuits

Conventional equivalence checking techniques can be applied to Reversible Circuits by treating them as AND-OR-NOT circuits. However, in this case, the resulting circuit will not be reversible and hence this approach may be inappropriate. Instead, if existing techniques are altered to ensure that the resulting circuit is also a reversible circuit, it will provide several optimization opportunities before conversion to a Circuit-SAT instance for use with state-of-the-art SAT solvers. Use of CNF clauses to determine equivalence has been researched extensively and hence conversion of a reversible circuit to this form is advisable. To reduce the total CNF-clauses and the complexity of the Circuit-SAT instance, pre-processing steps can be applied to reduce computation time. The work by

Yamashita et al. [6] explores one such approach and it also serves to test the scalability of the algorithm proposed in this thesis

1.3 Previous work

The work done by Feinstein et al [9] attempts to identify redundant gates by exhaustive simulation of different versions of the circuit created by controlled modifications. Another approach proposed by Zhong et al. [21] involves the use of a cross-point fault model to identify redundant gates. However, this work too involves exhaustive simulation after identification of potentially redundant gates. Most synthesis techniques involve a design optimization step and the optimization algorithm specific to the synthesis technique is then generalized to apply to existing circuits. However, in most cases, this algorithm involves re-synthesis of sub-circuits rather than identification of redundant gates.

1.3.1 BDD-based synthesis and reduction in number of gate inputs

A BDD-based reversible synthesis approach has been proposed by Robert Wille [2] [3]. The design obtained by this method often contains a large number of constant lines and garbage outputs. A selective re-synthesis algorithm [5] has been proposed to reduce the number of lines in the circuit and the total quantum cost of the circuit. This involves selection of a sub-circuit followed by re-synthesis of the logic function realized by that sub-circuit. If an alternate implementation with fewer constant lines and/or garbage outputs is obtained, the sub-circuit is replaced with this design. The gate lines for this

sub-circuit are merged with the rest of the circuit and it is seamlessly integrated into the existing design.

1.3.2 Template-based optimization algorithm

A reversible circuit simplification approach has been proposed by Maslov et al. [17] that involves the use of known reversible circuit structures called templates. A template is a cascade of two or more gates that has an alternate, equivalent representation usually involving fewer gates. The algorithm seeks to match gate structures in the circuit with these templates. When a match is found the gate structure is replaced by the equivalent structure utilizing fewer gates thereby reducing the circuit cost. This algorithm ensures that the number of gates and the gate lines in the circuit are reduced through local re-synthesis but it may not actually identify redundant gates. A few gate movement rules have also been formulated to facilitate grouping of gates for effective application of templates.

1.3.3. Window-based optimization algorithm

The work done by Soeken et al. [5] proposes the use of windows for optimization of reversible circuits. The main idea behind use of windows is to consider the sub-circuit formed by the gates currently within the bounds of the window. This sub-circuit is subjected to existing re-synthesis techniques for local optimization. This work presents two approaches for extracting windows from the sub-circuit, namely the Shift Window Optimization (SWO) and the Line Window Optimization (LWO). In both cases, the

circuit design was improved, but this was achieved through re-synthesis rather than removal of redundancies.

1.3.4 Optimization algorithms for Reversible Miter Circuit

For equivalence checking of reversible circuits, creation of a Reversible Miter circuit gives a performance improvement over direct use of conventional Miter circuits and SAT methods, as proposed by Yamashita et al. [6]. In case of similar gate structure in both circuits, identification of redundant gates (forming an identity gate) is trivial. However, in case of diverse structures, the work proposed the use of existing optimization techniques which focus more upon re-synthesis rather than redundant gate removal. Consequently, this results in higher resource requirement and computation time since identification of identity gates can be done without actually performing local re-synthesis. Hence, a low cost technique that achieves the same result will fare better compared to existing expensive techniques.

1.4 Contribution of this thesis

This thesis seeks to formalize identification of the non-trivial redundancies present in reversible circuits. The proposed algorithm constructs a Binary Decision Diagram (BDD) for all target lines within a window (sub-circuit) in the reversible circuit. The BDD variables are the circuit lines (both control and target lines), and redundant gates can be identified based upon the variables absent in the resulting BDD for the window of interest. Any gate that depends on any missing variable(s) as a control line would be

redundant. Consequently, this algorithm identifies both contiguous and non-contiguous redundancies beyond successively repeated gates. BDDs are constructed using the CUDD library [7] from University of Colorado. Use of BDDs does not result in high memory requirements or computation times since each iteration considers only a few gates out of the entire circuit.

A potential application of this method is redundancy removal prior to template matching or other optimization techniques which are based on re-synthesis of sub-circuits. It is possible that the application of our algorithm at intermediate stages of circuit synthesis will lower the cost of generating an optimal circuit. To demonstrate this fact, this work considers randomly generated un-optimized reversible circuits and identifies non-contiguous (non-trivial) redundancies within these circuits. Experimental results show successful identification of non-trivial redundant logic in a suite of reversible circuits. Likewise, this method also serves as a low-cost redundancy removal technique for Reversible Miter circuits to facilitate efficient equivalence checking of circuits.

1.5 Organization of this thesis

The rest of this thesis is structured as follows. In Chapter 2, we go over some of the basics of reversible circuit design, Binary Decision Diagrams and Equivalence Checking. Chapter 3 presents the basic BDD-based algorithm to identify redundant gates in reversible circuits. Chapter 4 presents the application of the algorithm for optimization of Reversible Miter circuits. In Chapter 5, we conclude the thesis and propose future work.

Chapter 2

Background

2.1 Characteristics of Reversible Circuits

If a logic function is a bijection, it can be implemented using a reversible circuit composed of reversible gates. Due to the one-to-one mapping between inputs and outputs, each gate must have the same number of input and output lines. Consequently, there is no information loss across any gate since each gate output is sufficient to determine the gate input. All irreversible combinational circuits have an equivalent representation as a reversible circuit since a reversible gate is a universal gate. Toffoli gates [8] and Fredkin gates are popularly used for the implementation of reversible logic functions [32]. Of these, only Toffoli gates are considered in this work. The four types of Toffoli gates used are NOT, CNOT, TOF and TOF4. Following is a short description of each type of gate followed by their logical gate representation.

NOT (a): $a \rightarrow a \oplus 1$

CNOT (a,b): $a,b \rightarrow a, b \oplus a$

TOF (a,b,c): $a,b,c \rightarrow a, b, c \oplus ab$

TOF4 (a,b,c,d): $a,b,c,d \rightarrow a, b, c, d \oplus abc$

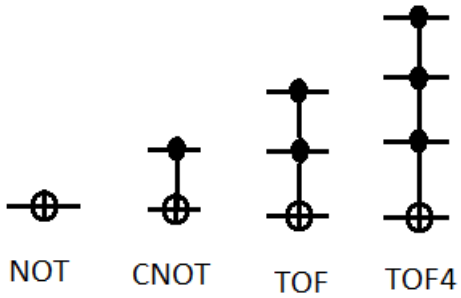


Figure 2.1: Basic Reversible Gates

For example, the NOT gate has one input, *a*, and one output, whose function is simply *a* XOR 1 (or negation of *a*). Hence, the input-output notation $a \rightarrow a \oplus 1$ states that *a* (left of →) is the input and $a \oplus 1$ (right of →) signifies the output of this gate. Similarly, the 2-input-2-output CNOT gate has the input-output mapping shown as a, b → a, b ⊕a. Another view is to consider each Toffoli gate as a multiplexer. A gate with a single control line is similar to a MUX with the gate target line and its complement as the MUX inputs and the control line as the MUX selection line. Consider the CNOT gate with inputs (*a, b*). The first (top) output is simply *a*, and the second output is *a* XOR *b*. In other words, the second output is the same as *b* if $a = 0$, and is the negation of *b* if $a = 1$. Yet another representation is obtained by considering each reversible gate as a combinational circuit composed of primitive logic gates. For the purpose of circuit analysis, this representation may prove useful. However, it converts the reversible circuit into an irreversible circuit and it may influence the analysis. Each of the above representations are shown in Figure 2.2.

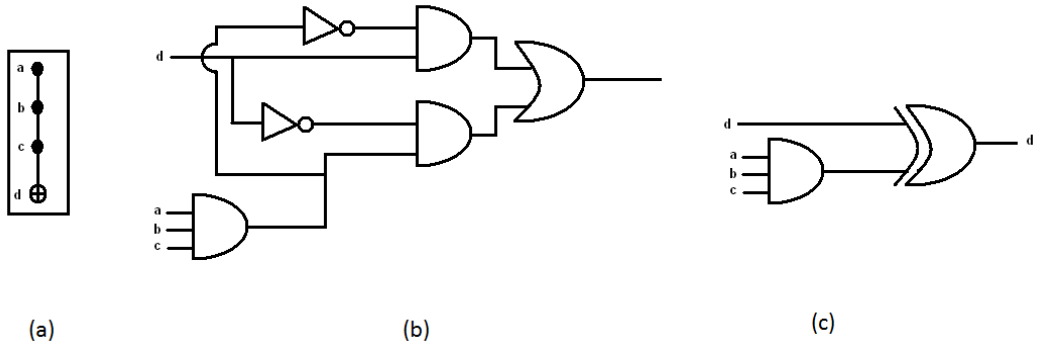


Figure 2.2: Representations of (a) Toffoli gate as (b) MUX and as (c) AND-XOR combination

A reversible circuit is fan-out free and has a single gate at each level. For example, the reversible circuit shown in Figure 2.3 (generated by Feinstein et al. [9]) has 4 inputs and 4 outputs, with 11 reversible gates, g1 to g11. Also, feedback paths are not allowed and hence all reversible circuits are combinational. State elements may be added, however, only at the boundaries of the circuit as in conventional Boolean logic circuits.

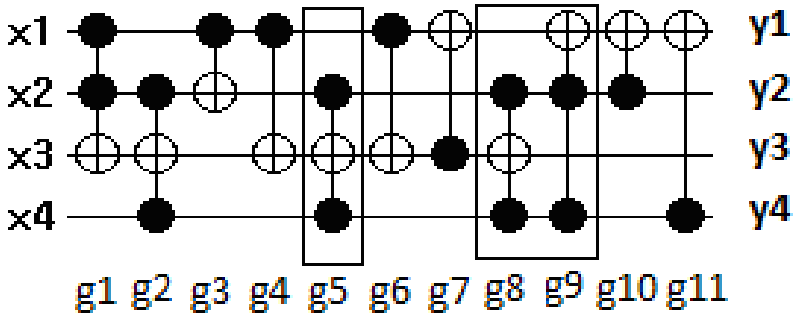


Figure 2.3: Reversible circuit "random4"

For Toffoli gates with multiple control lines, the control lines can be logically ANDed and the AND gate output connected to the multiplexer selection line. For example, consider gate g1 in Figure 2.3. The third output of g1 is the negation of x3 only if

x1=x2=1, otherwise this output is simply the same as x3. Likewise, reversible gates can also be represented as a combination of XOR and AND gates.

2.2 Existing synthesis techniques for Reversible Circuits

Different methods have been proposed for the synthesis of reversible logic, however, they are somewhat constrained. Exact synthesis methods have been proposed [10] [11] but they are suitable for functions with upto 6 variables. Heuristic methods [12] [13] can be applied for functions with more variables. However, the circuits generated may not be optimal and post-synthesis optimization is necessary. The work done by Wille et al. [2] focuses on the use of BDDs for synthesis and optimization of reversible circuits. The synthesis algorithm is capable of generating reversible circuits for large functions, but the resulting circuit may have many constant circuit lines. The follow-up work [3] improved the synthesized circuit by reducing the number of lines by an average of 17% and 40% in the best case. However, it was observed that in some cases, a reduction in the number of lines resulted in an increase in the number of gates and/or the quantum cost of the circuit. Another synthesis approach involves the generation of a sub-optimal reversible circuit followed by matching of contiguous gates in the circuit with known gate templates and performing gate reductions by replacing sub-circuits with their equivalent representations with a lower cost. [13]. Likewise, there are other synthesis approaches which follow the above pattern of producing a sub-optimal design and subsequently optimizing it. Thus, efficient generation of optimal reversible circuits remains a challenging task and identification of redundancy plays an important role in the circuit design.

2.3 Redundant gates in conventional and reversible circuits

Redundant gates are defined as those gates whose removal will not affect the output of the circuit. Redundancies may be specifically introduced into the circuit to ensure robustness or may be unintentional due to design flaws. A simple example of redundant logic would be a function *f* given as $f = (ab)' + b'$. This can be simplifies as $(a'+b') + b'$, which reduces to $a' + b'$. Thus, in the original function, the second term b' is redundant and its removal will not affect the circuit output.

In case of reversible circuits, redundancy is usually not desirable since absence of fan-outs negates the possibility of multiple control paths and getting higher fault tolerance. Reversible circuits are implemented using quantum gates and redundant gates contribute to higher quantum cost for the circuit. Most reversible circuit synthesis approaches involve an intermediate optimization step and presence of redundancies in the final circuit design indicates lacunae in the synthesis algorithm. When two identical reversible gates are placed next to each other, they form an identity function making both gates jointly redundant. Also, reversible circuits may contain non-contiguous redundant gate sets as explained in later sections. Identification and removal of such redundancies is particularly important for optimization of Reversible Miter circuits, as explained in later sections.

2.4. Analysis and identification of redundant gates

For conventional combinational circuits, the FIRE algorithm [14] is a low-cost technique that identifies redundant faults in conventional Boolean logic by extensive use of logic

implications. Similarly, it is possible to eliminate redundant gates from reversible circuits by application of pre-processing techniques. Recall that reversible circuits can be viewed as combinational circuits containing XOR, AND and NOT gates or as a cascade of MUXs. Implications can be generated for either of the above two representations of reversible circuits. However, because reversible circuits do not have fan-out structures, and because there are no direct implications for an XOR gate, implications may not be the most suitable method for the identification of redundant gates. Thus, unless the reversible gate is a simple NOT gate, a single control line cannot have an implication on the gate output.

Likewise, generation of a simple CNF (conjunctive normal form) formula for the reversible circuit has limited use as it can identify only successively repeated gates. Identification of repeated gate redundancy is possible by adding the constraint that the target output line and the corresponding input line are each other's complement. Subsequently, the CNF formula may be given to a SAT-solver, and if the formula is unsatisfiable, it indicates that the sub-circuit under consideration is redundant. However, this approach is not helpful in detecting non-contiguous redundant gates. In this paper, we aim to identify contiguous as well as non-contiguous redundant logic in a reversible circuit.

2.5 Binary Decision Diagrams

With the ever-increasing complexity of digital circuits, functional descriptions in form of logic equations or Karnaugh Maps grow exponentially with the number of variables. As

proposed by Akers et al. [22], a Binary Decision Diagram (BDD) provides a concise 'implementation free' description that yields meaningful results about the logical structure and the testing requirements of the function involved. A BDD presents a 'diagram' which tells the user how to determine the output based upon the function inputs and can be used to implement large digital functions.

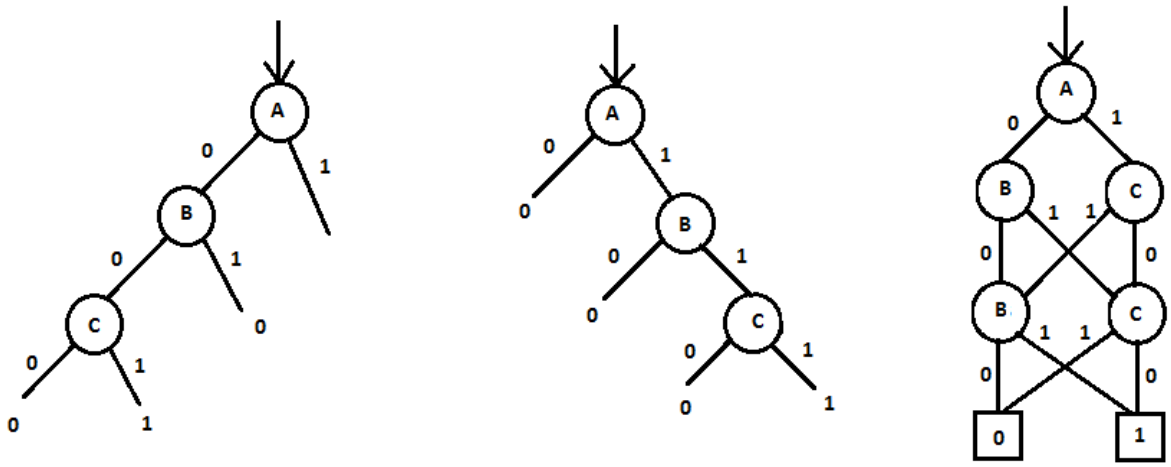
2.5.1 Definition of a BDD

A Binary Decision Diagram (BDD) is a rooted, directed acyclic graph with

- one or two terminal nodes of out-degree zero labeled 0 or 1
- a set of variable nodes *u* of out-degree two.

2.5.2 Explanation and examples

Consider the switching function, f = A+B'C. One way to determine the binary value of f would be to begin by considering the value of A. If A = 1, then f = 1 and we are done. If A = 0, we consider B. If B = 1, then f = 0 and again we are done. Otherwise, we go for C and its value will be the value of f. Figure 2.4 shows a simple diagram of this procedure. We enter at the node indicated by the arrow and then simply proceed downward through the diagram, noting at each node the value of its variable and then taking the indicated branch. When a 0 or 1 value is reached, this gives the value of f and the process ends. Figure 2.4 shows similar diagrams for some simple functions.



(a) f = A + B'C (b) f = ABC (c) f = A ⊕ B ⊕ C

Figure 2.4: BDDs for simple logical functions

2.5.3 Construction of BDDs

Binary Decision Diagrams are essentially an if-then-else structure and much work has been done to formalize the approach for optimal construction of a BDD for the given variable order [23]. An ITE (if-then-else) construct is described as:

An ITE is a ternary directed acyclic graph in which each leaf is labeled with TRUE, FALSE, or a literal, and each internal node has three out-edges pointing to the if-then-else-parts. The meaning of a leaf is the label on the node, and the meaning of an internal node is defined recursively as:

(if meaning(if-part) then meaning(then-part) else meaning (else-part)).

The notation used to represent an ITE is **(A, B, C),** where A is the if-part, B is the then-part and C is the else-part. In terms of a Boolean formula, this can be expressed as:

z = ite (f,g,h)

i.e. $z = fg + f'g$

Each primitive logic gate has a corresponding ITE notation. A given logic function can be expressed as a recursive definition of ITE. Eg: a function $f = ac + bc$ can be expressed in terms of ITE as f = ite (a, c, ite (b, c, 0)).

This notation can be conveniently converted into a BDD [19] by mapping the if-then-else parts to the corresponding BDDs and merging the common sub-BDDs of the 'then' and 'else' parts. If the ITE is a single variable then the corresponding BDD is the variable pointing to both 1 and 0. The above ITE expression can be represented as a BDD as follows:

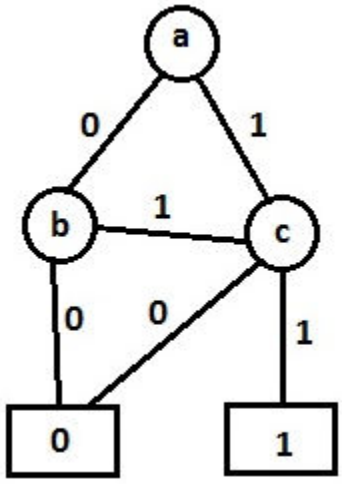


Figure 2.5: BDD for $f = ac + bc$

Thus, given a function and a variable order, an ITE structure can be used to generate the corresponding BDD. For this work, we use the CUDD package to generate the required BDDs. This package provides an API toolkit for BDD manipulation and a visual representation of the BDD in form of a ZDD, as explained in next sections.

2.5.4 Zero-suppressed BDDs

As proposed by Minato et al. [24], BDDs can be converted into Zero-suppressed Binary Decision Diagrams (ZDD) which have a more efficient structure. A BDD can be converted into a ZDD by eliminating the nodes whose 1-edge points to the 0-terminal node and then directly connecting the incoming edge(s) of the node to the other sub-graph directly. Thus, for a ZDD, a node is created only if the positive edge does not point to the 0-terminal node. The following figure compares the BDD and ZDD [25] created for the function: $f = ab + cd$

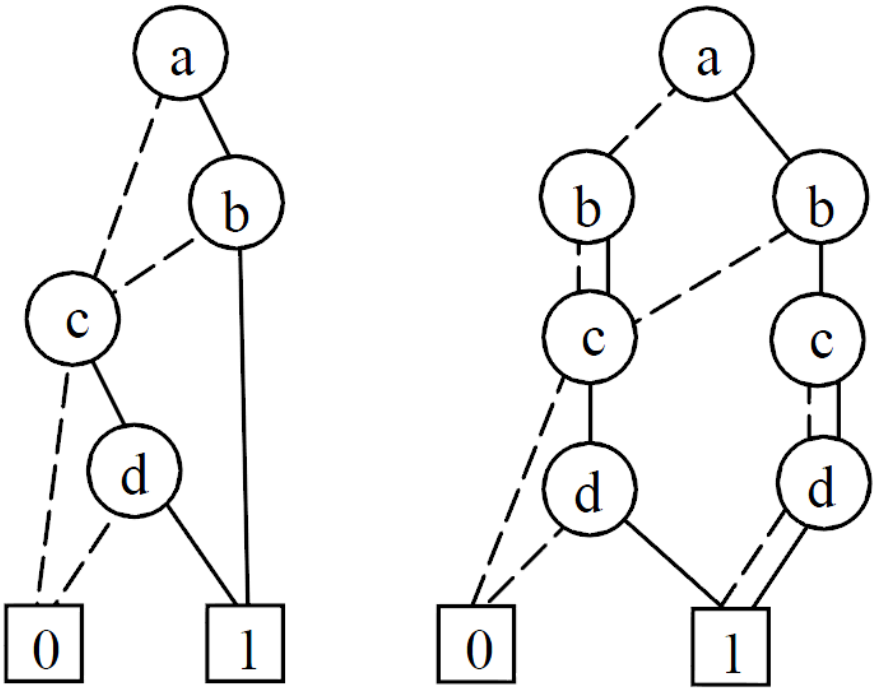


Figure 2.6: BDD and ZDD for $f = ab + cd$

The CUDD package used for BDD construction provides visual representation of the constructed BDD in form a ZDD. For our purposes, the ZDD is sufficient to show the variables present in the corresponding BDD. This is because the only variables

suppressed by the ZDD are the ones which result in a constant 0 output for all inputs. The presence of such variables is not possible in a reversible circuit as it violates the one-to-one mapping between the inputs and outputs of a reversible circuit.

2.6 CUDD package for BDD manipulation

The CUDD (Colorado University Decision Diagram) package [7] provides functions to manipulate Binary Decision Diagrams (BDDs), Algebraic Decision Diagrams (ADDs) and Zero-suppressed Binary Decision Diagrams (ZDDs). This package provides a rich set of APIs which can be directly called from C programs. For this work, the CUDD library was linked with a C++ program through the g++ compiler to provide an efficient method for generation of BDDs. This work uses the CUDD package as a black box utilizing the exported functions for construction of BDDs without concerning itself with internal details.

Prior to use of any of the package functions and structures, the package itself must be initialized by calling `Cudd_Init()`. The CUDD package maintains its own special hash tables called unique tables which guarantee that each node is uniquely labeled. The unique table, along with a few support data structures, constitutes the `DdManager` structure. Manager instance must be initialized prior to use of the package by calling `Cudd_Init()` on it. The `DdNode` structure and `Cudd_bddIthVar()` function are used to create new nodes in the decision diagram and under the control of a unique `DdManager` instance. The `DdManager` handles memory allocation, de-allocation and garbage collection and all memory can be reclaimed by calling the `Cudd_Quit()`

function on a `DdManager` instance. Our work mainly uses the following functions and corresponding data structures from the CUDD package:-

Function:- **Cudd_bddIthvar**

Signature:-

```
DdNode * Cudd_bddIthVar( DdManager * dd,
                         int  i
                        )
```

Description:-

Retrieves the BDD variable with index i if it already exists, or creates a new BDD variable. Returns a pointer to the variable if successful; NULL otherwise.

Function:- **Cudd_bddXor**

Signature:-

```
DdNode * Cudd_bddXor(   DdManager * dd,
                        DdNode * f,
                        DdNode * g
                       )
```

Description:-

Computes the exclusive OR of two BDDs f and g. Returns a pointer to the resulting BDD if successful; NULL if the intermediate result blows up.

Function;- **Cudd_bddAnd**

Signature:-

```
DdNode * Cudd_bddAnd(  DdManager * dd,
                       DdNode * f,
                       DdNode * g
                       )
```

Description:-

Computes the conjunction of two BDDs f and g. Returns a pointer to the resulting BDD if successful; NULL if the intermediate result blows up.

Function:- **Cudd_Not**

Signature:-

```
DdNode *Cudd_Not (DdNode * node
                  )
```

Description:-

Returns the complemented version of a pointer.

Function:- **Cudd_bddXor**

Signature:-

```
DdNode * Cudd_bddXor( DdManager * dd,
                      DdNode * f,
                      DdNode * g
                      )
```

Description:-

Computes the exclusive OR of two BDDs f and g. Returns a pointer to the resulting BDD if successful; NULL if the intermediate result blows up.

Function:- **Cudd_bddAnd**

Signature:-

```
DdNode * Cudd_bddAnd( DdManager * dd,
                      DdNode * f,
                      DdNode * g
                      )
```

Description:-

Computes the conjunction of two BDDs f and g. Returns a pointer to the resulting BDD if successful; NULL if the intermediate result blows up

Function:- **Cudd_SupportIndex**

Signature:-

```
int * Cudd_SupportIndex( DdManager * dd, manager
                         DdNode * f DD whose support is sought
                       )
```

Description:-

Finds the variables on which a DD depends. Returns an index array of the variables if successful; NULL otherwise. The size of the array equals the number of variables in the manager. Each entry of the array is 1 if the corresponding variable is in the support of the DD and 0 otherwise.

Function:- **Cudd_Init**

Signature:-

```
DdManager * Cudd_Init( unsigned int  numVars,
                       unsigned int  numVarsZ,
                       unsigned int  numSlots,
                       unsigned int  cacheSize,
                       unsigned long  maxMemory
                       )
```

Description:-

Creates a new DD manager, initializes the table, the basic constants and the projection functions. If maxMemory is 0, Cudd_Init decides suitable values for the maximum size of the cache and for the limit for fast unique table growth based on the available memory. Returns a pointer to the manager if successful; NULL otherwise.

Function:- **Cudd_Quit**

Signature:-

```
void Cudd_Quit( DdManager * unique )
```

Description:-

Deletes resources associated with a DD manager and resets the global statistical counters. (Otherwise, another manager subsequently created would inherit the stats of this one.)

Function:- **Cudd_DumpDot**

Signature:-

```
int Cudd_DumpDot( DdManager * dd, manager
                  int  n
                  DdNode ** f,
                  char ** inames,
                  char ** onames,
                  FILE * fp
                  )
```

Description:-

Writes a file representing the argument DDs in a format suitable for the graph drawing program dot. It returns 1 in case of success; 0 otherwise (e.g., out-of-memory, file system full). Cudd_DumpDot does not close the file: This is the caller responsibility. Cudd_DumpDot uses a minimal unique subset of the hexadecimal address of a node as name for it. If the argument inames is non-null, it is assumed to hold the pointers to the names of the inputs. Similarly for onames. Cudd_DumpDot uses the following convention to draw arcs:

- solid line: THEN arcs;
- dotted line: complement arcs;
- dashed line: regular ELSE arcs.

The dot options are chosen so that the drawing fits on a letter-size sheet.

2.7 CNF and Boolean Satisfiability

2.7.1 CNF and SAT solvers

Most Equivalence Checking methods have Binary Decision Diagrams (BDDs) or Boolean Satisfiability (SAT) based methods as the underlying engine. BDDs often face the problem of memory explosion for large circuits and hence SAT-based methods have taken the centre stage. A SAT problem accepts the input inform of a Conjunctive Normal Form (CNF) and determines an assignment for the CNF literals or concludes that no assignment exists. A CNF is a conjunction of clauses, while a clause is a disjunction of literals. A literal can be represented by either a positive polarity or negative polarity of a signal, and has a value of 1 when the signal is at the corresponding polarity.

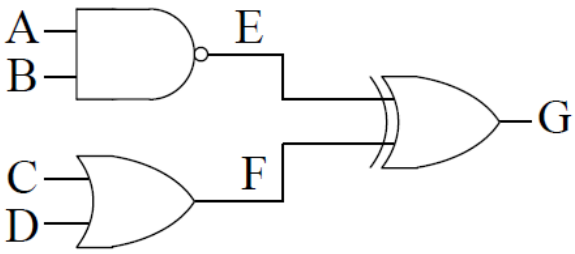


Figure 2.7: A simple combinational circuit

For the circuit shown in Figure 2.7 its CNF formula is

$(A \vee E)(B \vee E)(\neg A \vee \neg B \vee \neg E)$
$(\neg C \vee F)(\neg D \vee F)(C \vee D \vee \neg F)$
$(E \vee F \vee \neg G)(E \vee \neg F \vee G)(\neg E \vee F \vee G)(\neg E \vee \neg F \vee \neg G)$

To solve a CNF formula every clause needs to be satisfied, i.e. at least one literal in every clause needs to be true. For example in the CNF formula above, for clause $(\neg C \vee F)$ to be true, either C = 0 or F = 1 needs to be satisfied. Although the SAT problems are NP-Complete in nature [15], existing SAT Solvers (Eg: Zchaff [26], MiniSat [27], GRASP

[28]) have been known to successfully solve very large CNF formulas with more that a million variables and a million clauses.

2.7.2 Miter Circuits

A Miter circuit is created from two circuits. To check the equivalence of two circuits, C1 and C2, a Miter circuit can be constructed to compare their outputs through XOR gates. As shown in the figure, the corresponding primary inputs (PIs) of C1 and C2 are tied together and the corresponding outputs are connected to a dual-input XOR gate.

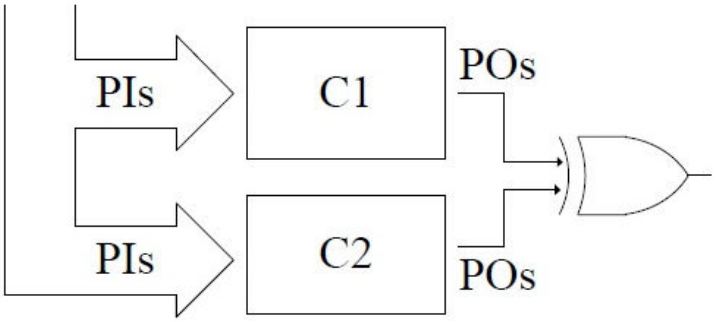


Figure 2.8: Conventional Miter Circuit

The equivalence property of the two designs is denoted by the XOR gate output being constant 0. If the problem is to be solved by a Satisfiability (SAT) based approach, the Miter circuit needs to be converted to a CNF formula. In addition to these clauses a unit clause must be added to CNF formula to force the XOR gate 1. Addition of this clause implies that the corresponding output pair has dissimilar values and the SAT Solver is to provide an input assignment that will not violate this constraint. For circuits with multiple outputs, an OR gate can be used to connect all XOR gates and the output of this OR gate is forced to 1 in CNF clauses. If a SAT Solver proves that no assignment exists for the CNF formula such that the output can be 1, the equivalence of the designs is proved.

Chapter 3

Identification of Redundant gates using BDDs

3.1 Redundant gates in Reversible Circuits

Given the two-step process followed by most synthesis algorithms, it is quite possible that redundant gates will be present in reversible circuits. Redundant gates increase the total gate count and are also responsible for presence of redundant faults. Both of these have an adverse effect on the yield. As presented by Feinstein et al. [9], there is a possibility of redundant logic being present in non-optimal reversible circuits. In addition to this, redundant logic may appear as a combination of gates rather than as a single gate. Further, this multi-gate redundant logic may be composed of a set of *non-contiguous* gates, making their detection challenging and difficult. Also, it is possible that two or more sets of non-contiguous redundant gates might overlap in the circuit increasing the complexity of the problem.

3.2 Brute force method for redundancy identification

The work by Feinstein uses a brute force approach to identify redundant gates. This involves creation of a copy of the circuit with 'controlled modifications' and comparison of this modified circuit with the original circuit using equivalence checkers. These controlled modifications involved removal of two or more gates from the circuit. Upon comparison through equivalence checkers, if the original circuit and the modified

circuit are equivalent, then gates removed from the original circuit do not affect the circuit output i.e. these gates are redundant.

For large circuits, random selection of gates is necessary as exhaustive gate enumeration and deletion is not possible. Two sets of non-contiguous redundant gates were identified in the reversible circuit "random4", as illustrated in Figure 3.1, through exhaustive gate deletion followed by simulation-based equivalence checking. In this circuit, no single gate is individually redundant. However, a set of gates, either {g4, g6}, or {g5, g8, g9}, could be removed without changing the functionality of the circuit.

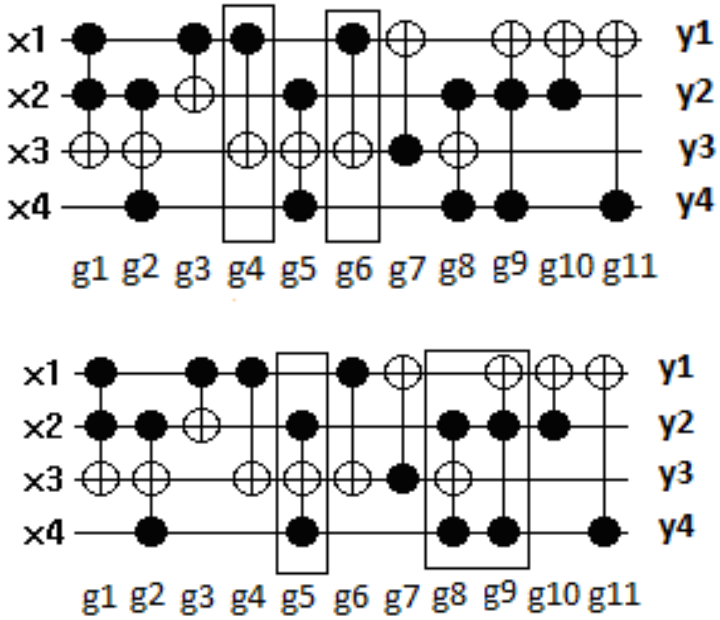


Figure 3.1: Sets of non-contiguous redundant gates in the circuit "random4"

The approach presented above is suitable only for small circuits with a few inputs. As the number of inputs and gates increases, exhaustive checking is not feasible and a random approach may not yield results. This thesis aims to improve the above method and provide an efficient and scalable method for redundant gate identification.

3.3 Use of BDDs for redundant gate identification

3.3.1 Selection of sub-circuits

By definition, a change in the input values of a redundant gate does not affect the output of the circuit. Thus, it is necessary to enumerate all possible input values, implicitly or explicitly, before declaring a gate to be redundant. However, this needs to be performed for every gate, as well as for every group of gates. Recall that this group of redundant gates may not be contiguous, making this search space extremely large.

To tackle this problem, only small sections of the circuit are considered at a time. This is achieved by sweeping through the reversible circuit using windows of gates, where a window is a contiguous set of gates. Within each window, we will examine and check if there exists any redundant single or multi-gate logic. This sweep of the circuit is performed using overlapping windows. Consequently, all possible instances of redundant logic smaller than the window size can be detected by our method.

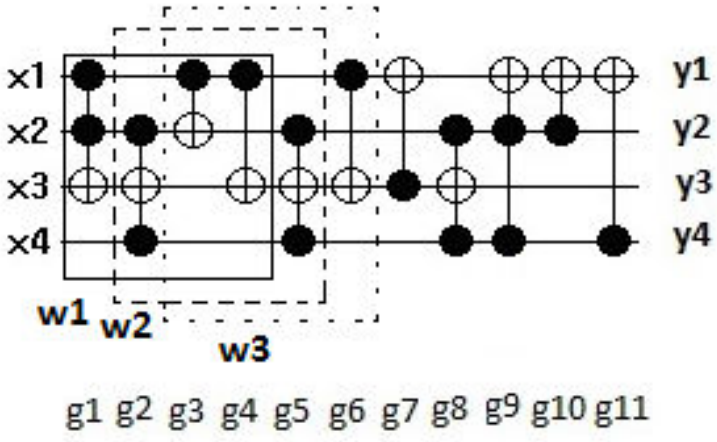


Figure 3.2: Window of size 4 swept over reversible circuit random4

The above figure shows three consecutive windows for a window size of 4. The sub-circuit (formed by window w1) consists of gates g1, g2, g3, g4. The second sub-circuit is

formed by shifting the window ahead by one gate. It consists of gates g2, g3, g4 and g5. The entire circuit is analyzed sub-circuit by sub-circuit by sweeping windows of varying size. As can be seen, this approach effectively scans the entire circuit and hence no potentially redundant non-contiguous gate group is missed.

3.3.2 Construction of BDD

The generation of logic functions for every window of the circuit is performed by viewing each sub-circuit as a net of XOR, AND and NOT gates. The CUDD package provides APIs for creating a BDD for the above primitive gate types. The BDDs formed for each of these functions can be appended and optimized to form the BDD for a given circuit output. A BDD implicitly represents the logic function of the gates involved in the window. It is constructed by traversing the complete input space of this sub-circuit and hence contains the necessary information to conclude whether a reversible gate output line is independent of given input lines within the current window of interest. A BDD needs to be generated only for those lines which are a target line at least once in the sub-circuit. The nodes of a BDD generated for a reversible circuit line indicate the variables upon which the output of that line depends upon.

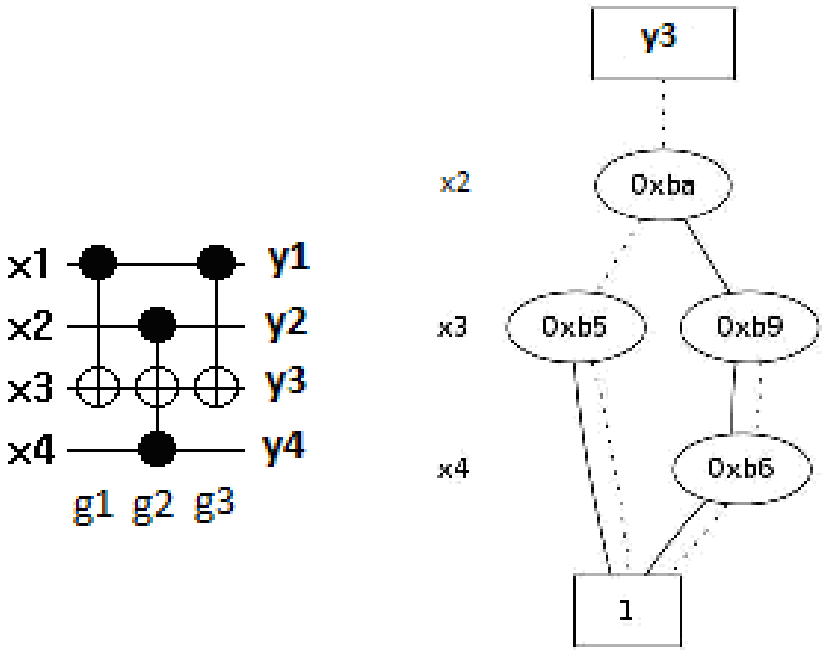


Figure 3.3: BDD generated for sub-circuit {g1, g2, g2}

Consider the sub-circuit and its BDD illustrated in Figure 3.3. The gate output on the target line y3 is independent of the line x1, although x1 is a control line for two gates in this sub-circuit. Thus, the signal on x1 plays no part in deciding the output y3 for this particular window. This implies that all gates within this window which depend on signal x1 do not affect the output. Hence, gates g1 and g3 are cumulatively redundant. This can be proved by applying the 'moving rule' followed by the 'deletion rule' as presented in [16].

In general, reversible gates that depend upon one or more redundant signals in the current sub-circuit, as identified by the corresponding BDD, are redundant. This assertion holds true in other sub-circuits with non-contiguous gates operating on different target lines, as shown in Figure 3.4. The target lines y1 and y3 are present in the BDD as the two root nodes. Having two root nodes for a BDD is equivalent to considering the intersection of

the two component BDDs. Variables x2 and x4 are absent in the BDD, and thus any gate that depends on x2 and x4 would be redundant. In this case, gates g1, g4, and g5 can be safely removed without changing the logic function. This can be proved by constructing the truth tables for the original sub-circuit and the optimized sub-circuit.

Table 3.1 presents the truth table for the sub-circuit illustrated in Figure 3.4. As seen in Figure 3.5, the redundant gate set {g1, g4, g5} can be eliminated resulting in an optimized version of the sub-circuit. The truth table is the same for both the un-optimized and optimized versions and it can be trivially verified for all input vectors for the sub-circuit.

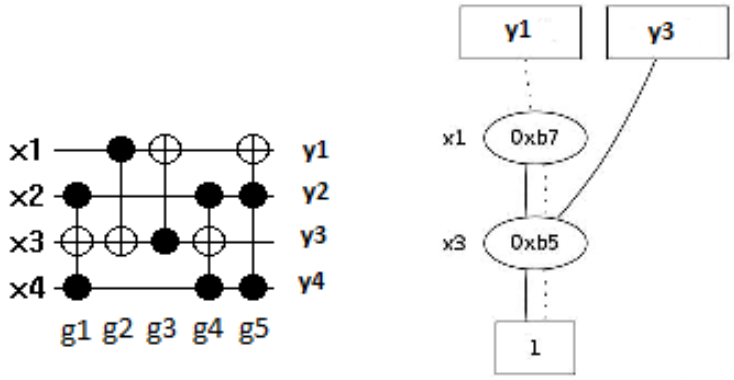


Figure 3.4: BDD generated for the sub-circuit {g1, g2, g3, g4, g5}

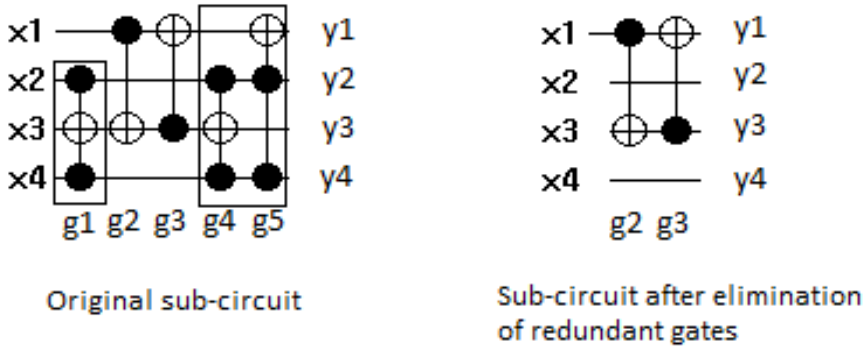


Figure 3.5: Sub-circuit before and after redundancy elimination

x1	x2	x3	x4		y1	y2	y3	y4
0	0	0	0		0	0	0	0
0	0	0	1		0	0	0	1
0	0	1	0		1	0	1	0
0	0	1	1		1	0	1	1
0	1	0	0		0	1	0	0
0	1	0	1		0	1	0	1
0	1	1	0		1	1	1	0
0	1	1	1		1	1	1	1
1	0	0	0		0	0	1	0
1	0	0	1		0	0	1	1
1	0	1	0		1	0	0	0
1	0	1	1		1	0	0	1
1	1	0	0		0	1	1	0
1	1	0	1		0	1	1	1
1	1	1	0		1	1	0	0
1	1	1	1		1	1	0	1

Table 3.1: Truth Table for redundant sub-circuit

Redundant gates need not be two instances of the same gate. As seen in the redundant sub-circuit in Figures 3.5 and 3.6, three gates can be removed without changing the final outputs of the circuit. Both of these sub-circuits are taken from “random4” reversible circuit.

Our work identifies non-contiguous redundant gates for all sub-circuits obtained by a window-based sweep of the complete circuit. Since reversible circuits are composed of a cascade of reversible gates with a single gate at each level, the following lemma can be derived.

Lemma : In a reversible circuit, redundant gates (either contiguous or non-contiguous) in a sub-circuit are also globally redundant.

Proof: Consider a reversible circuit consisting of windows A, B, and C, linearly cascaded. Without loss of generality, let the window of interest be window B. Suppose there exists a group redundant logic in window B, and let B’ be the new window obtained

after removing the redundant logic from B. Since the logic function for B' is identical to that of B, the linear cascading of A, B', and C would result in an identical global function before redundancy removal. ∎

The above lemma allows for a safe removal of redundancies (could be non-contiguous) within a local window for reversible circuits, making our algorithm both practical and feasible.

3.4 Algorithm

The proposed algorithm considers windows (sub-circuits) of the given reversible circuits similar to a peep-hole optimization technique. This is done by sweeping a window of a given size over the entire circuit, from left to right, considering different sub-circuits present "under the window" for each iteration. During the sweep, sub-circuits sized between 3 gates to *k* gates are analyzed. Currently, the window is simply swept over the circuit, shifted ahead by a single gate at a time. Window sizes of 3, 4, up to *k* gates are in use. Future work can focus upon having a dynamic window size based upon a heuristic that maximizes the probability of finding redundant logic.

The complete algorithm is as follows:

1. For window size $w = 3$ to k
2. Consider the sub-circuit currently in the window of size w
3. Create a BDD for each output line of the window which is a target line for at least one gate.
4. Check if any variable is missing in the constructed BDDs.
5. Perform intersection of the set of missing input variables in each BDD for all target lines.
6. Gates with one or more control lines in the intersection set obtained in step 5 are redundant.
7. Repeat if some redundant gates are found

The CUDD library from University of Colorado was used to construct the BDDs. The above algorithm considers at most 6 reversible gates in each window and hence the BDDs are very easy to construct. The time taken for BDD generation depends upon the number and type of gates in the circuit. In our experiments, the execution time increased linearly with the increase in number of gates in the circuit. An increase in the number of variables increases the complexity of the BDD generated per sub-circuit and requires additional cost. However, since a BDD is generated on a per-window basis, individual BDDs do not explode in memory or cause excessive computational overhead.

3.5 Experimental Results

The window-sweeping algorithm was implemented and executed on existing benchmark reversible circuits [17] [18], in addition to our randomly synthesized reversible circuits. First, we confirmed that these optimal circuits from [17] [18] did not include redundant gates, as expected. Next, randomly generated reversible circuits (without application of any optimization techniques) were considered for our experiments. Non-trivial redundant gates were indentified in many circuits. The random reversible circuits were chosen such that there were no successively repeated gates since identifying that redundancy is trivial.

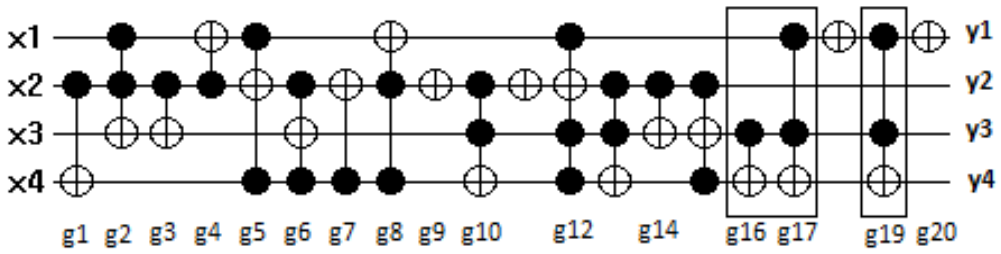


Figure 3.6: Redundant gates in circuit Rand_01

For the circuit Rand_01 shown in Figure 3.6, the BDDs generated for a window of size 4 over gates g16 to g19 contained only variables x1 and x4, thus any gate in this window that depends on x3 is redundant. This was true for all the generated BDDs within this window. Line x3 is the control line for three gates: g16, g17 and g19 and hence these three gates are redundant. After removal of these three gates, gates g18 and g20 also become redundant thereby eliminating the last 5 gates in the reversible circuit. Thus, gates g16 to g20 form an identity circuit which may be of use for creating new templates and extend the work done in [16].

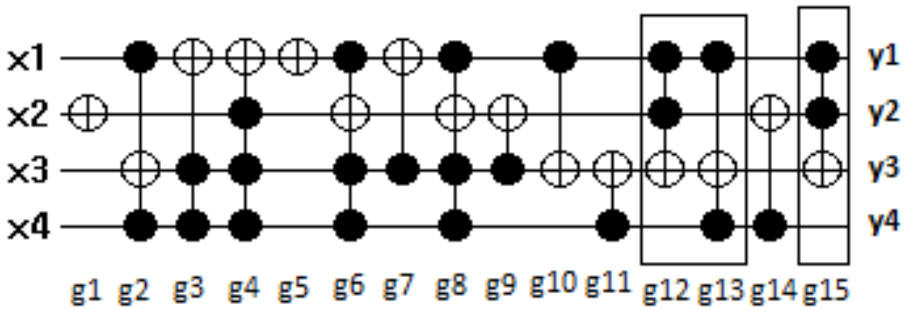


Figure 3.7: Redundant gates in circuit Rand_02

Consider circuit Rand_02 shown in Figure 3.7. A window of size 4 over the gates g12 to g15 identified three redundant gates as shown in the figure. The intersection of the sets of variables eliminated by both the generated BDDs contained x1. The gates with x1 as one of its control lines get eliminated as redundant gates.

Tables 3.2 and 3.3 sum up the results for a few randomly-generated reversible circuits obtained on a system with Ubuntu 10.04 OS, Pentium IV 3.2 GHz processor and 1 GB RAM. For small circuits with 4 inputs/outputs, upto 25% gates were identified as redundant. For circuits with more I/O and large numbers of gates, the number of redundant gates identified declined steadily. For circuits with larger number of I/O, we

created two sets of benchmarks. The first set places denser control signals, while the second one has less dense control signals. The circuit names with a '_d' suffix (where d=2 to 4) were generated with the less dense control constraint such that each gate will have 4 or fewer control lines. These circuits are presented in Table 3.3. Due to use of very few control lines per gate, the probability of presence of redundancy in this suite of circuits increases. As expected, more redundancies were present and identified. Also, as the number of allowed control lines per gate decreased from 4 to 2, the occurrences of redundancy increased. In general, due to large number of inputs and high gate count, only limited redundant logic was encountered in the random circuits. Nevertheless, our method was able to find non-trivial redundant logic in these large circuits as in smaller circuits. The execution times were very short. For circuits with 2000 reversible gates, approximately 8.5 seconds were needed.

Circuit name	Number of I/O	Initial gate count	Final Gate count	Time reqd. (secs)
Rand_01	4	20	15	0.090
Rand_02	4	15	12	0.049
Rand_03	3	20	18	0.084
Rand_4_60	4	60	55	0.220
Rand_4_100	4	100	91	0.416
Rand_4_200	4	200	188	0.836
Rand_4_800	4	800	780	3.328
Rand_5_500	5	500	495	1.992
Rand_6_800	6	800	790	3.236
Rand_7_1000	7	1000	996	4.076
Rand_10_1000	10	1000	998	4.140
Rand_12_1200	12	1200	1198	5.092

Table 3.2: Redundant gates in randomly generated reversible circuits

Circuit name	Number of I/O	Initial gate count	Final Gate count	Time reqd. (secs)
Rand_6_800_4	6	800	792	3.128
Rand_6_800_3	6	800	782	3.396
Rand_6_800_2	6	800	776	3.368
Rand_7_1000_4	7	1000	994	4.208
Rand_7_1000_3	7	1000	988	4.180
Rand_7_1000_2	7	1000	986	4.244
Rand_10_1000_4	10	1000	994	4.120
Rand_10_1000_3	10	1000	988	4.284
Rand_10_1000_2	10	1000	978	4.280
Rand_12_1200_4	12	1200	1194	5.052
Rand_12_1200_3	12	1200	1196	5.212
Rand_12_1200_2	12	1200	1170	5.264
Rand_20_2000_4	20	2000	1994	8.440
Rand_20_2000_3	20	2000	1990	8.816
Rand_20_2000_2	20	2000	1986	8.676

Table 3.3 Redundant gates in constrained pseudo-randomly generated circuits

3.6 Limitations of the method

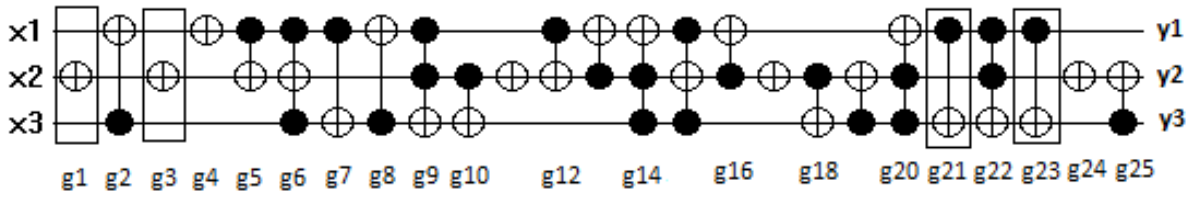


Figure 3.8: Unidentified redundancies in a 3-bit reversible circuit

The proposed algorithm was less successful for 3-bit circuits, partially due to the reason that we will illustrate using Figure 3.8. By applying the 'moving rule' followed by the 'deletion rule' as presented in [10], the gates {g1, g3} and {g21, g23} are redundant and can be eliminated. However, this is not detected by window sweeping. In the case of {g1, g3}, both are NOT gates with the control line same as the target line. Generation of a BDD for this target line will still list x2 as a required variable. For gates {g21, g23}, the intermediate gate, g22 has x1 as a control line and hence BDD lists x1 as a required variable. After elimination of g21 and g23, variable x1 will still affect the output on line x3 and hence the BDD is ineffective in identifying redundancies in such situations. This also shows the complexity of this problem, in which multiple approaches may be required to identify all redundancies present in the reversible circuit.

3.7 Summary

Our algorithm compares favorably with existing optimization techniques. The template-based circuit simplification technique proposed by D. Maslov requires the application of multiple templates to the sub-circuit, gradually reducing the number of gates until all

redundant gates are eliminated. In contrast, generation of BDD for the sub-circuit identifies all redundant gates at once quickly. Also, our method identifies redundant gates rather than performing a re-synthesis of the circuit. This may assist in circuit analysis and diagnosis in the future. Our method faces the limitation of being unable to identify known redundant gates in a few circuit structures. However, this drawback can be overcome by application of pre-processing techniques to identify structures which may pose problems and deal with them as appropriate. Inspite of use of BDDs, our algorithm has short execution times, mainly due to use of circuit windows and selective generation of BDDs. Most existing optimization techniques were developed as an extension to circuit synthesis procedure. Our algorithm is the first attempt to develop an independent algorithm to identify redundant gates in a reversible circuit.

Chapter 4

Equivalence checking for Reversible Circuits

4.1 Equivalence checking for digital circuits

Equivalence checking methods have been well-established for irreversible circuits which include combinational and sequential circuits. Also, a few equivalence checking strategies have been developed for reversible circuits [29] which involve the use of QuIDDs [30] or QMDDs [31]. Equivalence checking of combinational circuits primarily involves the creation of a Miter circuit and examination of the circuit outputs for all valid circuit inputs. The equivalence of the two circuit designs is verified through generation of CNF clauses for the circuit and feeding them to a SAT-solver. This procedure can be applied to reversible circuits as well. However, given the reversible nature of the circuits, the Miter circuit can be modified to determine circuit equivalence at a much lower cost.

4.2 Creation of Miter circuit for Reversible Circuits

Instead of tying together the circuit inputs and comparing the circuit outputs via XOR gates, the two circuits are cascaded end-to-end and the circuit inputs are compared with the outputs, as presented in the work done by Yamashita et al. [6]. Given two quantum (or reversible) circuits C_1 and C_2, their *reversible miter* is defined to be one of the following circuits: $C_1 \cdot C_2^{-1}$, $C_2^{-1} \cdot C_1$, $C_2 \cdot C_1^{-1}$, or $C_1^{-1} \cdot C_2$

For two equivalent circuits, a cascade of the first circuit and the inverse of the second circuit will yield the exact same input vector at the output of the Miter circuit, as illustrated in Figure 4.1.

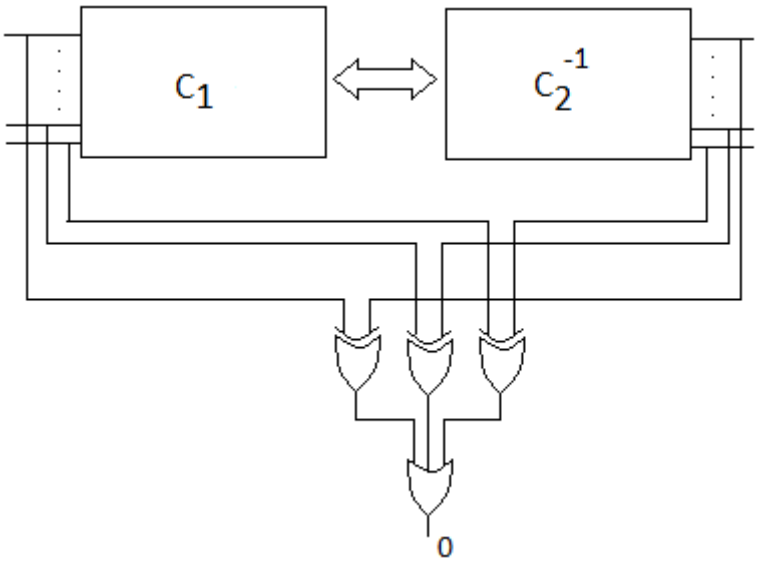


Figure 4.1: Block diagram of a Reversible Miter circuit

4.2.1 Basic Idea

Every reversible gate is its own inverse and hence two identical reversible gates placed next to each other form an identity function. Each reversible Miter places the last gate of C_1 next to the last gate of C_2 (recall that C_2 is inverted). If these two gates are identical, they can be cancelled out as redundant gates making the second-to-last gates in each circuit as the next adjacent gate pair around the join of the two circuits. Even if the last two gates are not equivalent, it is possible to check if they are a part of a larger group of non-contiguous redundant gates. If the two circuits are functionally equivalent, then all

the gates in the Miter circuit will eventually get cancelled out without need for simulation-based or SAT-based equivalence checking.

Application of existing methods of simplification [5] [16] may prove to be expensive. We propose the use of our BDD-based redundancy detection algorithm to identify and eliminate redundant gates in reversible Miters. As discussed earlier, our algorithm is not resource intensive and has short runtime. Its application to equivalence checking shows that the algorithm is scalable and Reversible Miter circuits with hundreds of input lines and thousands of gates can be processed without memory blowup.

4.2.2 Generation of CNF-clauses for Reversible Miter circuits

After simplification of Reversible Miter, a few gates may still remain in the Miter. In this case, CNF-clauses are constructed for this reduced circuit and given to a SAT-solver to determine equivalence. Due to simplification of the two circuits, the number of CNF clauses generated is much less compared to those generated after forming a conventional Miter circuit for reversible circuits. One approach to generation of CNF clauses is to view the CNOT gate as an XOR gate with a bypass wire and a Toffoli gate as an XOR-AND combination and a bypass. This representation is identical to the one presented in Figure 2.8. As proposed by in the work by Yamashita et al, a more efficient clause generation exists, presented as follows:

Consider a Toffoli gate whose control bits are x1 and x2, and target bit is x3. Since the Toffoli gate does not modify two of its inputs, there is no need for separate output variables. A new variable y1 must be added for the target bit. The logical consistency is

given by the condition y1 = (x1 · x2) ⊕ x3 which can be expressed by the following six clauses.

Case x1 = 0 or x2 = 0:
Clauses: $(x_1 + x_3 + y_1)$ · $(x_1 + x_3 + y_1)$ · $(x_2 + x_3 + y1)$ · $(x_2 + x_3 + y_1)$.
Case x1 = x2 = 1 :
Clauses: $(x_1 + x_2 + x_3 + y_1)$ · $(x_1 + x_2 + x_3 + y_1)$

The above set of clauses is generated for each gate of the reversible circuit regardless of the circuit structure. In the next step, a set of clauses is added which are satisfied only by those variable combinations where some circuit output differs from the respective circuit input. In this step, it is possible to reuse some of the y variables introduced in the earlier step. Without loss of generalization, it can be explained as follows:

Let a new *y* variable corresponding to the i-th primary output be y_{Oi} . (If there is no target bit on the i-th bit-line, we do not introduce a new variable for the i-th primary output, since the input and the output functions on the i-th bit-line are the same, and the following clauses need not be added)

A new variable z_i is required to express the functional consistency of the i-th bit-line. Namely, we consider that z_i becomes 1 only when $x_i \neq y_{Oi}$. For this condition, we add the following clauses.

Case $z_i = 0$. Clauses: $(z_i + x_i + y_{Oi})$ · $(z_i + x_i + y_{Oi})$.
Case $z_i = 1$. Clauses: $(z_i + x_i + y_{Oi})$ · $(z_i + x_i + y_{Oi})$.

Finally we add $(z_1 + z_2 + \ldots + z_n)$ where n is the number of bit-lines of the circuits. Since z_i=1 means that the input and the output functions on the i-th bit-line are different, the two circuits are different when $(z_1 + z_2 + \ldots + z_n)$ is satisfied.

Thus, the above construction generates a SAT formula that is satisfied only by those input combinations for which the corresponding outputs of two circuits produce different

values. A CNF-SAT formula constructed for a miter grows linearly with the size of the miter. Compared to conventional Miters, Reversible Miter have a key advantage of being significantly smaller due to gate cancellations as explained in next section.

4.3 Optimization of Reversible Miters

We now discuss the application of our technique to Reversible Miters. Our algorithm performs a window-based sweep of the Reversible Miter. Our algorithm is modified to consider windows of varying size around the join of the two reversible circuits in the Miter. A total of 10 window sizes are considered. In each window, half of the gates come from circuit C_1 and the other half come from C_2^{-1}. The number of redundant gates identified in each window is stored and the window with maximum gates eliminated is chosen. The redundant gates are canceled out and the process is repeated for the reduced Miter circuit till all the gates are eliminated or no redundant gates are identified in any of the 10 windows.

Instead of eliminating gates in pairs (when a pair of identical gates is seen), our method can eliminate multiple gates at once. In reversible circuits, it is possible that eliminating two gates as redundant by a greedy approach may inhibit identification of a larger group of non-contiguous redundant gates and result in a sub-optimal design.

4.4 Results

Reversible Miter circuits were constructed for small equivalent reversible circuits implementing a given function by utilizing different gate structures in each design.

Circuits with more than a hundred inputs and thousands gates were also considered. However, due to lack of large benchmarks circuits with multiple implementations of given functions, we formed a reversible miter of two copies of the same circuit to test our approach for scalability. Table 4.1 presents the results of our experiments.

Circuits forming Reversible Miter		No. of I/O	Total gates in Miter	Total redundant gates deleted	Time reqd. (secs)
nth_prime_inc29	nth_prime_inc29	5	58	54	0.252
ham7-tc	ham7-25-49	7	48	8	0.084
hwb10-3595	hwb10-3595	10	7190	7190	16.30
hwb11-8214	hwb11-8214	11	16428	10158	23.47
hwb11-11600	hwb11-8214	11	19814	12	0.128
ham15tc1	ham15tc1	15	264	264	0.716
ham15-tc1	ham15-70	15	202	52	0.340
ham15-109	ham15-70	15	179	42	0.272
mod1024adder	mod1024adder	20	110	110	0.232
hwb20ps	hwb20ps	25	148	148	0.400
mod1048576adder	mod1048576adder	40	420	420	1.084
gf2^16_301	gf2^16_301	45	602	602	1.384
gf2^18_375	gf2^18_375	51	750	750	1.648
hwb50ps	hwb50ps	56	486	486	1.516
gf2^32_1148	gf2^32_1148	96	2296	2296	5.132
gf2^50_2647	gf2^50_2647	150	5294	5294	13.14
gf2^64_4285	gf2^64_4285	192	8570	8570	21.92
hwb200ps	hwb200ps	208	2706	2706	8.213
gf2^100_10297	gf2^100_10297	300	20594	20594	53.02
hwb500ps	hwb500ps	509	7996	7996	35.84

Table 4.1: Redundant gates canceled in Reversible Miter circuits

As shown in the table, our approach can process large circuits in a relatively short time. As expected, most Reversible Miters for two copies of a single circuit have all their gates marked as redundant and thus can be identified as equivalent. However, as mentioned earlier, our method still faces a few limitations with respect to the identification of redundant gates dependent upon gate lines used by a non-redundant gate in the window. Our algorithm will skip past gates that it cannot identify as redundant. However, if none

of the gates in a 10-gate window around the join of the two circuits get marked as redundant, the algorithm will terminate. When the window size exceeds 10, BDD-based redundancy detection becomes ineffective – the maximum window size in the actual proposed algorithm was limited to 10 gates. Consequently, this limits the identification of redundancy in known equivalent Reversible Miter circuits. Nevertheless, the elimination of gates is helpful to reduce the total number of clauses if a SAT-solver based technique is to be used subsequently for equivalence checking.

Another observation to be made is that the time taken is dependent upon the total number of gates in the circuit rather than the number of gate lines. Although our algorithm constructs BDDs, it focuses on a sub-circuit in each iteration and constructs a BDD only for the target lines in the sub-circuit. Since BDDs need to be computed for all 10 windows to choose the optimal set of redundant gates to be eliminated, the time required increases linearly with the number of gates in the Reversible Miter.

4.5 Summary

As indicated by the obtained results, application of our algorithm results in significant reduction in the total gates present in the Reversible Miter. This translates to fewer CNF-clauses and higher performance for the SAT-solver. In the ideal case, our algorithm cancels out all the gates in the Reversible Miter establishing equivalence without the need for subsequent processing. The limitation of our algorithm becomes apparent when a Reversible Miter formed of two-copies of the same circuit still has a few gates left over after the application of our algorithm. This limitation is the manifestation of the drawback

explained in the earlier section for 3-bit reversible circuits. However, in most cases, our algorithm results in significant reduction of the size of the Reversible Miter.

Reduction in the gate count translates to generation of fewer CNF clauses to represent the Reversible Miter and hence the SAT-solver requires shorter time to analyze these clauses for satisfiability. In addition to removing contiguous gates forming an identity gate, our algorithm also eliminates non-contiguous redundant gates thereby identifying more gates at a time.

Chapter 5

Conclusion and Future Work

We have presented a new low-cost method for the identification of non-trivial redundant gates in reversible circuits. This approach can be easily extended to larger reversible circuits and the cost grows only linearly with the circuit size. The proposed sweeping algorithm works best when applied at the intermediate stage between generation of a sub-optimal reversible circuit and application of optimization techniques. As indicated by the results, non-trivial redundant logic was successfully identified in a suite of reversible circuits. In a few circuit structures, the algorithm was not as successful as expected. However, this limitation can be offset by use of pre-processing techniques to identify potentially problematic gate structures which reduce the effectiveness of our algorithm.

An equivalence checking method for reversible circuits was in existence prior to this work, but the application of our algorithm can be used to improve this method. Instead of employing expensive optimization techniques initially developed for circuit synthesis, use of our low-cost algorithm will prove to be efficient. Also, formation of Reversible Miters of thousands of gates with hundreds of input lines provided large reversible circuits to test the scalability of our approach. As expected, due to window-based analysis and selective generation of BDDs, our approach presents short execution times without resource blowup.

In a few Reversible Miters composed of two copies of the same circuit, our algorithm failed to identify all gates as redundant. As mentioned earlier, this limitation can be avoided by use of appropriate pre-processing techniques.

Compared to existing techniques, use of our algorithm presents multiple improvements for reversible optimization techniques. Identification and removal of redundant gates will reduce both circuit delay faults and masking of existing faults. When applied to Reversible Miters, identification of contiguous as well as non-contiguous redundant gates will ensure shorter execution times by removing more redundant gates at a time. Also, the results show a significant reduction in the total gate count for Reversible Miters thereby reducing the total CNF clauses for the subsequent SAT-solver.

An investigation of pre-processing steps and possible modifications for the algorithm are necessary to ensure that all redundant gates are identified. Likewise, instead of generating a BDD for each and every window, research is necessary to determine a window size that maximizes the probability of finding redundant logic. This window size will change dynamically as different sections of the reversible circuit are analyzed for redundancy.

Bibliography

[1] R. Landauer. Irreversibility and heat generation in the computing process. *IBM J. Res. and Dev.*, 5:183.191, 1961.

[2] Wille, R.; Drechsler, R.; "BDD-based synthesis of reversible logic for large functions," *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, pp.270-275, 26-31 July 2009.

[3] Wille, R.; Soeken, M.; Drechsler, R.; , "Reducing the number of lines in reversible circuits," *Design Automation Conference (DAC), 2010 47th ACM/IEEE* , pp.647-652, 13-18 June 2010

[4] Maslov, D.; Dueck, G.W.; Miller, D.M.; "Toffoli network synthesis with templates," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems",* vol.24, no.6, pp. 807-817, June 2005

[5] Soeken, M.; Wille, R.; Dueck, G.W.; Drechsler, R.; , "Window optimization of reversible and quantum circuits," *IEEE 13th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS), 2010*, pp.341-345, 14-16 April 2010

[6] Yamashita, S.; Markov, I.L.; , "Fast equivalence-checking for quantum circuits," *IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH),2010* vol., no., pp.23-28, 17-18 June 2010

[7] F. Somenzi, 'The CUDD Package", University of Colorado at Boulder, Version 2.4.2 available at http://vlsi.colorado.edu/~fabio/.

[8] T. Toffoli, "Reversible Computing", MIT Lab for Comp. Sci., Tech Memo MIT/LCS/TM-151, 1980

[9] Feinstein, D.Y.; Thornton, M.A.; Miller, D.M.; "Partially Redundant Logic Detection Using Symbolic Equivalence Checking in Reversible and Irreversible Logic Circuits," *Design, Automation and Test in Europe, 2008. DATE '08* , pp.1378-1381, 10-14 March 2008

[10] Golubitsky, O.; Falconer, S.M.; Maslov, D.; "Synthesis of the optimal 4-bit reversible circuits," *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pp.653-656, 13-18 June 2010.

[11] Grosse, D.; Wille, R.; Dueck, G.W.; Drechsler, R.; , "Exact Multiple-Control Toffoli Network Synthesis With SAT Techniques," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol.28, no.5, pp.703-715, May 2009

[12] Miller, D.M.; Maslov, D.; Dueck, G.W.;, "A transformation based algorithm for reversible logic synthesis," *Design Automation Conference, 2003. Proceedings*, pp. 318- 323, 2-6 June 2003

[13] Maslov, D.; Dueck, G.W.; Miller, D.M.; "Toffoli network synthesis with templates," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems",* vol.24, no.6, pp. 807-817, June 2005.

[14] Iyer, M.A.; Abramovici, M.; "FIRE: a fault-independent combinational redundancy identification algorithm," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol.4, no.2, pp.295-301, Jun 1996.

[15] S. Cook, The complexity of theorem proving procedures," in *Proc. of the Third annual ACM symposium on Theory of Computing*, pages 151-158, 1971.

[16] Maslov, D.; Dueck, G.W.; Miller, D.M.; "Simplification of Toffoli networks via templates," *16th Symposium on Integrated Circuits and Systems Design, 2003. SBCCI 2003. Proceedings.* pp. 53- 58, Sept. 2003

[17] D. Maslov. Reversible logic synthesis benchmarks page. http://webhome.cs.uvic.ca/~dmaslov/, May 2009.

[18] R. Wille, D. Grose, L Teuber, G. W. Dueck, and R. Drechsler. RevLib: An online resource for reversible functions and reversible circuits. http://www.revlib.org.

[19] Kevin Karplus; "Using if-then-else dags for multi-level logic minimization," *Technical report ICSC-CRL-88-29*, *Board of Studies in Computer Engineering, University of California at Santa Cruz,* Dec. 1988

[20] C. Bennett, "Logical reversibility of computation," *IBM J. Research & Development*, vol. 17, no. 6, pp. 525-532, Nov. 1973.

[21] Jing Zhong; Muzio, J.C.;, "Using Crosspoint Faults in Simplifying Toffoli Networks," 2006 IEEE North-East Workshop on Circuits and Systems, pp.129-132, June 2006

[22] Akers, S.B.; "Binary Decision Diagrams," *IEEE Transactions on Computers*, vol.C-27, no.6, pp.509-516, June 1978

[23] Lam, W.K.C.; Brayton, R.K.; , "On relationship between ITE and BDD," *IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD'92),* pp.448-451, Oct 1992

[24] Minato, S.; , "Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems," *30th Conference on Design Automation,* pp. 272- 277, June 1993

[25] A. Mishchenko, "An introduction to zero-suppressed binary decision diagrams", Technical report, Portland State University, June 2001

[26] [Online] http://www.princeton.edu/~chaff/zchaff.html

[27] [Online] http://minisat.se/

[28] [Online] http://vlsicad.eecs.umich.edu/BK/Slots/cache/sat.inesc.pt/~jpms/grasp/

[29] Wille, R.; Grosse, D.; Miller, D.M.; Drechsler, R.; "Equivalence Checking of Reversible Circuits," *39th International Symposium on Multiple-Valued Logic (ISMVL '09)*, pp.324-330, May 2009

[30] Viamontes, G.F.; Rajagopalan, M.; Markov, I.L.; Hayes, J.P.; "Gate-level simulation of quantum circuits," *Design Automation Conference, Asia and South Pacific (ASP-DAC'03)*, pp. 295- 301, Jan. 2003

[31] D. M. Miller and M. A. Thornto; "QMDD: A decision diagram structure for reversible and quantum circuits," *IEEE Int'l Symposium on Multiple-Valued Logic (ISMVL '09)*, p.30, May 2006

[32] M. Nielsen and I. Chuang, Quantum Computation and Quantum Information, Cambridge University Press, 2000