Sufficiency-based Filtering of Invariants for Sequential Equivalence Checking

Wei Hu

Dissertation 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
Electrical Engineering

Michael S. Hsiao, Chair
Patrick Schaumont
Sandeep Shukla

Februrary 1, 2010
Blacksburg, Virginia

Keywords: Boolean Satisfiability(SAT), Sequential Equivalence Checking(SEC), Assume and Verify, Invariant filtering

Sufficiency-based Filtering of Invariants for Sequential Equivalence Checking

Wei Hu

(ABSTRACT)

Verification, as opposed to Testing and Post-Silicon Validation, is a critical step for Integrated Circuits (IC) Design, answering the question "Are we designing the right function?" before the chips are manufactured. One of the core areas of Verification is Equivalence Checking (EC), which is a special yet independent case of Model Checking (MC). Equivalence Checking aims to prove that two circuits, when fed with the same inputs, produce the exact same outputs. There are broadly two ways to conduct Equivalence Checking, simulation and Formal Equivalence Checking. Simulation requires one to try out different input combinations and observe if the two circuits produce the same output. Obviously, since it is not possible to enumerate all combinations of different inputs, completeness cannot be guaranteed. On the other hand, Formal Equivalence Checking can achieve 100% confidence. As the number of gates and in particular, the number of flip-flops, in circuits has grown tremendously during the recent years, the problem of Formal Equivalence Checking has become much harder — A recent evaluation of a general-case Formal Equivalence Checking engine [1] shows that about 15% of industrial designs cannot be verified after a typical sequential synthesis flow. As a result, a lot of attention on Formal Equivalence Checking has been drawn both academically and industrially.

For years Combinational Equivalence Checking (CEC) has been the pervasive framework for Formal Equivalence Checking(FEC) in the industry. However, due to the limitation of being able to verify circuits only with 1:1 flip-flop pairing, a pure CEC-based methodology requires

a full regression of the verification process, meaning that performing sequential optimizations like retiming or FSM re-encoding becomes somewhat of a bottleneck in the design cycle [2]. Therefore, a more powerful framework — Sequential Equivalence Checking (SEC)—has been gradually adopted in industry.

In this thesis, we target on Sequential Equivalence Checking by finding efficient yet powerful group of relationships (invariants) among the signals of the two circuits being compared. In order to achieve a high success rate on some of the extremely hard-to-verify circuits, we are interested in both two-node and multi-node (up to 4 nodes) invariants. Also we are interested in invariants among both flip-flops and internal signals. For large circuits, there can be too many potential invariants requiring much time to prove. However, we observed that a large portion of them may not even contribute to equivalence checking. Moreover, equivalence checking can be significantly helped if there exists a method to check if a subset of potential invariants would be sufficient (e.g., whether two-nodes are enough or multi-nodes are also needed) prior to the verification step. Therefore, we propose two sufficiency-based approaches to identify useful invariants out of the initial potential invariants for SEC. Experimental results show that our approach can either demonstrate insufficiency of the invariants or select a small portion of them to successfully prove the equivalence property. Our approaches are quite case-independent and flexible. They can be applied on circuits with different synthesis techniques and combined with other techniques.

Dedication

To my parents

Acknowledgments

I want to express my sincere gratitude to my research advisor, Professor Michael S. Hsiao, for his selfless guidance and help during my research. I am really grateful for the opportunity to work with him and to be inspired by his extraordinary passion and dedication. This work would have never been possible without his instruction for both Testing and Verification courses or his intuitive suggestions during individual meetings. I would also want to thank Professor Patrick Schaumont and Professor Sandeep Shukla for agreeing to be on my M.S. committee.

I will always cherish the time that I had in PROACTIVE. I received a lot of help from Min, Xueqi, Huy, Sandesh, Nikhil, Neha and all the other lab mates for both my research and life that I really appreciate.

I want to thank my parents who have always been there for my advanced education. I would have never finished my education without their support.

Also I want to thank Dr. Ngoc Hoang and Mr. Peter Kormendi for giving me the chance to intern at NAL Research, Dr. William Saunders and Mr. Chris Hudson for giving me the chance to Co-op at Adaptive Technologies as part of my Masters' education.

Wei Hu

December 2010

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The increased complexity of digital hardware systems has been driven by both the advances in Semiconductor technologies and the endless need of higher computing speed. Computing touches nearly every aspect of human life including medicine, finance, engineering, education, etc. Statistically, the number of transistors in the latest Intel 8-Core Xeon Nehalem-EX as of December 2010 is 2,300 million, which is 55 times the number of transistors in Intel Pentium 4 that was released back in 2000, as an evidence that the Moore's law still applies. In reality, this drastic increase of the complexity in integrated circuits (ICs) has not only shaped the IC development cycle to a very strict and semi-automated industrial flow, but also posed a tremendous problem in Functional Verification, which ensures the correctness of design before the design is actually manufactured.

In this chapter, we will introduce the design flow in IC designs. Following that, we will explain the necessity and complexity of Functional Verification and especially Equivalence Checking. We will then describe the previously proposed Sequential Equivalence Checking techniques proposed under various notions of Equivalence. Based on their limitations, we introduce the need for the technique proposed in the thesis and thus address the contribution.

At last we provide the organization of this thesis.

1.1 IC Development Flow and Levels

Figure 1.1 illustrates a flow chart for a general IC Development Flow. As indicated in

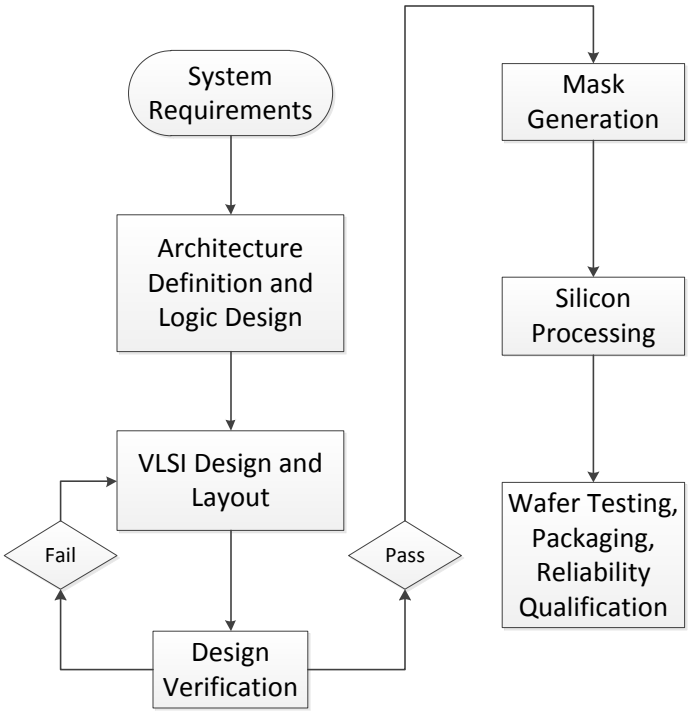

Figure 1.1: IC Development Flow

Figure 1.1, a typical IC design starts with deciding the specifications and requirements for the system. It examines issues such as chip area, power, functionality, speed, cost and other design factors while setting these specifications. At this level, the main portions of the system are illustrated with block diagrams, with no details on the contents of the blocks.

Only the input and output characteristics of each block are detailed. During Architecture Definition and Logic Design, a Behavior Level description of the design is first produced. The design is then detailed with a Register-Transfer Level (RTL) language which indicates structural information. Many tools can be used to ease the process nowadays, especially since the designs are generally complicated and a lot of general purpose functions have already been pre-defined as modules. Logic synthesis and Static Timing Analysis (STA) can then be performed by the same tools to get a Gate Level Design, which can then be converted to a lower level — Circuit level — description to be used for layout. Before chip is manufactured, Design Verification is performed to make sure all levels of the design satisfy specification, ensuring the correctness of the design.

1.2 Functional Verification and Equivalence Checking

Design Verification generally includes Functional Verification, Timing Verification, Power Verification, etc. Among all of them, Functional Verification takes most of the time and effort. From Figure 1.1 we can see that if the design fails verification, at least some part of it needs to be modified, re-synthesized, and verified again. So if Design Verification itself takes too much time, the whole time-to-market of IC development could face a bottleneck. Unfortunately that is often the case: Up to 70% of the design development time and resources are spent on functional verification.

As one of the major problems in Functional Verification, Equivalence Checking needs to be performed for almost every step of IC Design Flow. For example, after a design is synthesized from RTL to Gate Level, one needs to make sure that these two designs are equivalent and no error has been introduced during Logic Synthesis. Also, whenever any optimization is performed on any level of the design, Equivalence Checking again needs to be conducted to

ensure the equivalence of the new design and the original design. The problem of Equivalence Checking is hard because, naively, if one needs to achieve 100% confidence that one design is identical to the other, all combinations of inputs need to be tried, and the number of combinations of inputs is exponential to the number of inputs. For example, given a small design with 30 inputs, there can be 1 billion combinations of them. To make it even worse, for sequential circuits, the same input-output behavior need to be assured for all states of the design, and the potential number of states is also exponential to the number of flip-flops. Fortunately, today's advance in Formal Equivalence Checking techniques have made the problem possible to solve.

1.3 Formal Equivalence Checking

There are two main classes of Equivalence Checking: Simulation-based Equivalence Checking and Formal Equivalence Checking. While in theory, a simulation test bench has high controllability of its input ports for the Design Under Verification(DUV), test benches generally have poor controllability over internal points. With Simulation-based Equivalence Checking, there are a lot of challenges:

i) Various input scenarios need to be tested;

ii) A functional coverage model which is used to determine if simulation is enough needs to be created;

iii) Test benches need to be built;

iv) Specific direct tests need to be created;

v) A lot of time is needed for simulation.

Formal Equivalence Checking (FEC), on the other hand, is a systematic process that uses mathematical reasoning to verify that two designs are equivalent. With FEC, all the chal-

lenges that Simulation-based Equivalence Checking faces can be theoretically overcome since FEC can implicitly exhaust all possible input values. In other words, it is unnecessary to figure out how to simulate the design or create multiple scenarios to achieve high observability. Recently, There has been several major for FEC. Theoretically, Some of the recent research shows that one of the major FEC techniques — Sequential Equivalence Checking—is PSPACE-Complete [3]. Practically, based on the underlying engine including Binary Decision Diagrams (BDDs) and Boolean Satisfiability (SAT), more industrial sized circuits can now be handled.

1.4 Previous Work

Although different notions of SEC exist — which we will elaborate further in the next chapter—have been suggested, SEC fundamentally is to check whether two circuits produce the same outputs for any input sequences applied. The different notions of SEC can be broadly divided into two categories: equivalence that requires a specific initial state and equivalence that does not require such. Equivalence that requires a specific initial state only needs to guarantee that two circuits behave the same after they are brought into a state and is thus an easier notion to achieve.

In [4], a BDD-based assume and verify induction over two-time-frame model was used to demonstrate the validity of suspected redundant signals, therefore eliminating the need of state traversal during SEC. The work is recently extended by [5] which utilizes a SAT-based technique and targets on both retiming and resynthesis optimizations. In [6], a K-th invariant based SEC framework is suggested with a combination of a Bounded Model Checker to ensure equivalence in first K time-frames and an IProver to prove equivalence after K-th time-frame. A recent industrially successful approach was suggested in [2], where the authors

incorporated multiple algorithms into their own SEC toolset—Sixthsense— and the tool was able to explore equivalent signal pairs for redundancy removal on designs with up to 10,000 state elements.

Despite the success of SEC techniques targeting on Reset Equivalence, an initial state of a hardware design is not always available in practice. Although [7][8] have proposed SAT-based methods for computing reset states, deriving them remains a hard problem. Therefore, techniques targeting on equivalence without a specific initial state have also been proposed over the years.

In [9], a SAT-based method is proposed to use dual-rail encoding to represent X as initial states, converting the problem to a 2-valued equivalence problem with a known initial state. Although techniques that require a reset state can then be applied, the approach essentially doubles the number of signals. Recently, more work has emerged on approximately computing reachable state space by inductively proving invariants based on SAT. These approaches target on the equivalence that only requires initial states to be in reachable states. In [29], equivalent flip-flop pairs between circuits are explored. Potential equivalent flip-flops were first derived by random simulation, then a modified s-graph that only has flip-flops and outputs as its nodes is created to reduce the flip-flop pairs for final induction. A recent data-mining based approach proposed in [11] targets on hard-to-verify circuits. Through mining Boolean relationships among flip-flops using the data-mining tool BLOSSOM[12], powerful complex multi-node relationships are derived, verified and proven to be able to constraint search space enough to verify the equivalence of the circuits which can not be verified by ABC package[13][14]. Approaches have also been suggested to efficiently verify invariants. In [16] the authors suggested an operation CMERGE during the assumption step to efficiently reduce number of clauses while avoiding unnecessary loss of constraint. During the verify step, they refine candidate groups by performing solution-based simulation whenever

an invariant is proven to be false. In [15], a window proof technique is suggested to prove a subset of two-node invariants first, then more and more are proved with the help of an implication graph.

1.5 Contribution of this Thesis

In this thesis, we target on equivalence checking without the need of a specific initial state and propose two filtering techniques that can be applied after the potential invariants are derived, but before they are proven/verified. For many circuits, there can be too many potential invariants requiring much time to prove. In addition, we observed that a large portion of them may not even contribute to equivalence checking. Our filtering schemes thus can select a subset of powerful potential invariants towards the target-equivalence checking based on two over-approximation models that judges the sufficiency of invariants selected. The approaches are highly flexible and can be applied on circuits with very different state encodings and/or combined with other synthesis techniques. Experimental results show that our approaches can either demonstrate insufficiency of the invariants or select a small portion of them to successfully prove the equivalence property. To the best of our knowledge, no similar work has been done before.

1.6 Organization of this Thesis

The rest of the thesis is organized as follows. In Chapter 2 we go over some basics in SAT-based SEC, including basic circuit models and concept of invariants and their application in SEC. In Chapter 3 we introduce a novel implementation of deriving potential invariants by random simulation, the first invariant filtering model, the filtering scheme based on it and

the fix-point calculation to prove true invariants.In section 4 we introduce the validity check for invariants involving internal signals, the second filtering model and the filtering scheme based on it. In section 5 we conclude the thesis and propose future work.

Chapter 2

Background

2.1 Notions of Equivalence

Different notions of sequential equivalence have been proposed including Sequential Hardware Equivalence(SHE)[17], Safe Replacement and Delay Replacement [18], Three-valued Safe Equivalence [19], etc. We first introduce the notion of SHE in the next subsection.

2.1.1 Sequential Hardware Equivalence (SHE)

The notion of SHE was introduced because the classical notion of FSM equivalence was thought to be too restrictive. The notion targets on designs without reset latches but has an initializing sequence which can bring the design to a known state from an unpredicted power-up state. We first introduce some notations and definitions.

An FSM is a quintuple $(Q, I, O, \lambda, \delta)$, where Q is the set of states, I is the set of input values, O is the set of output values, λ is the output function, and *delta* is the next state function.

Definition 1. *[17] Given two states $s_0 \in Q_{D_0}$ and $s_1 \in Q_{D_1}$, state s_0 is equivalent to*

state s_1 (denoted by $s_0 \sim s_1$) if for any sequence of inputs $\pi \in I^*$, it is the case that $\lambda_{D_0} s_0, \pi = \lambda_{D_1} s_1, \pi$. We say that π distinguishes s_0 and s_1 if $\lambda_{D_0} s_0, \pi \neq \lambda_{D_1} s_1, \pi$.

Then, we introduce the notion of SHE which is characterized by Theorem 1,

Theorem 1. *[17] Designs D_0 and D_1 are sequentially hardware equivalent (denoted by $D_0 \approx D_1$) if and only if there exists an input sequence $\pi \in I^*$ such that for any state pair $(s_0, s_1) \in Q_{D_0} \times Q_{D_1}$, we have $\delta_{D_0}(s_0, \pi) \sim \delta_{D_1}(s_1, \pi)$.*

In other words, Theorem 1 states that there exists a sequence that can bring the two designs to an equivalent state pair (from any initial state pair). Since the condition of sequential hardware equivalence requires the preservation of only one initializing sequence, Safe-replacement is proposed.

2.1.2 Safe-replacement

Definition 2. *[18] Design D_1 is a safe replacement for design D_0 (denoted by $D_1 \leq D_0$), if given any state $s_1 \in Q_{D_1}$ and any finite input sequence $\pi \in I^*$, there exists some state $s_0 \in Q_{D_0}$ such that the output behavior $\lambda_{D_1} s_1, \pi = \lambda_{D_0} s_0, \pi$.*

Checking safe-replacement is more difficult than checking SHE because for D_1 to be a valid safe replacement for D_0, safe replaceability requires every state in D_1 to have at least one equivalent state in D_0, therefore every state of the replacing circuit needs to be examined for verifying safe replacement.

2.1.3 Delay-replacement

Comparing to safe-replacement, the requirement for delay-replacement is more relaxed. To define delay-replacement, we denote the $n-cycle$ delayed version of D_1 as D_1^n.

Definition 3. *[18] A design D_1 is an $n-$delay replacement for design D_0 if $D_1^n \leq D_0$. In other words, for any state $s_1 \in Q_{M_1^n}$ and any finite input sequence π, there exists some state $s_0 \in Q_{M_0}$ such that $\lambda_{M_0}(s_0, \pi) = \lambda_{M_1^n}(s_1, \pi)$.*

Delay-replacement is similar with safe-replacement since it does not make any assumptions of initialization sequences, either. However, delay-replacement is more practical and flexible than safe-replacement because it ignores the replacing design's first n cycles after power up, thus allowing better design optimizations than safe-replaceability.

2.2 Boolean Satisfiability and CNF

Binary Decision Diagrams(BDDs) and Boolean Satisfiability(SAT) based methods have been the dominant underlying engine in Sequential Equivalence Checking. Due to the memory-explosion problem that BDDs have, major research attention has shifted to SAT recently. A SAT problem is modeled as taking in a Conjunctive Normal Form(CNF) as input and finding an assignment v for it or deciding no assignment exists. As its name states, 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. For the simple circuit shown in Fig. 2.1, its CNF formula is

$$(A \vee E)(B \vee E)(\neg A \vee \neg B \vee \neg E)$$

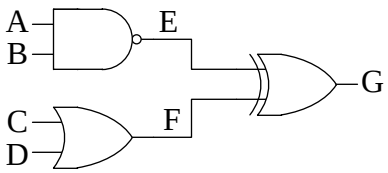

Figure 2.1: Simple Circuit

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

To solve a CNF formula every clause needs to be satisfied, meaning 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 [20], Several SAT Solvers have been proposed to be very effective in solving very large CNF formulas, e.g. There has been success stories of Zchaff[21][22] solving CNF formulas with more than 1 million variables and 10 million clauses. Some of the other state-of-the-art sat solvers includes Minisat[23][24] and GRASP[25][26]. All of these SAT solvers are derivatives of DPLL-algorithm[27] and are search-tree based. In this paper, we use Zchaff as underlying engine because it supports incremental solving.

2.3 Miter Circuit and Circuit Unrolling

A miter circuit is created from two circuits. Suppose equivalence is being checked for circuits C_1 and C_2, the miter circuit C_{miter}(Fig. 2.2) can be constructed by: 1) Tying the corresponding primary inputs(PIs) of C_1 and C_2 together, and 2) joining the corresponding primary outputs(POs) of C_1 and C_2 by XOR gates.

The equivalence property of the two designs is denoted by XOR gate being constant 0. If the problem is being solved by a Satisfiability(SAT)-based approach, after converting the miter circuit to CNF formula, a unit clause is added to force the XOR gate to 1, meaning the corresponding output-pair gets different values. When there are multiple outputs in each circuit, an OR gate can be used to connect all XOR gates and is forced to 1 in CNF instead of the individual XOR gates. If a SAT Solver proves that no assignment exists for this CNF formula, the equivalence of the designs is proved.

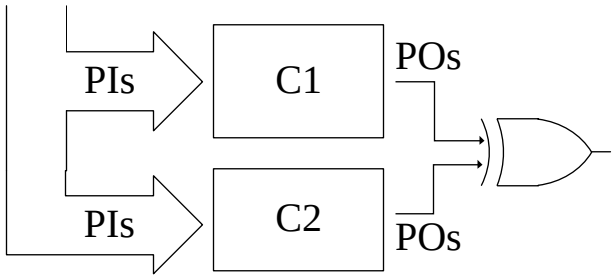

Figure 2.2: Miter Circuit

For sequential circuits, a k-time-frame unrolled circuit as shown in Fig. 2.3 can be derived for Formal Verification. In this model, in addition to PIs, flip-flop signals which serve as current state (CS) elements are treated as pseudo primary inputs (PPIs) in the left-most time-frame. On the other hand, in addition to POs, the outputs of flip-flops which serve as next state (NS) elements are treated as pseudo primary outputs (PPOs) in the right-most time-frame. The model is simply constructed by copying information of Circuit Under Verification (CUV, miter circuit for SEC specifically) for k times and connecting PPOs (the NS elements of previous time-frame) to PPI (the CS elements of the next time-frame), thus enabling signals

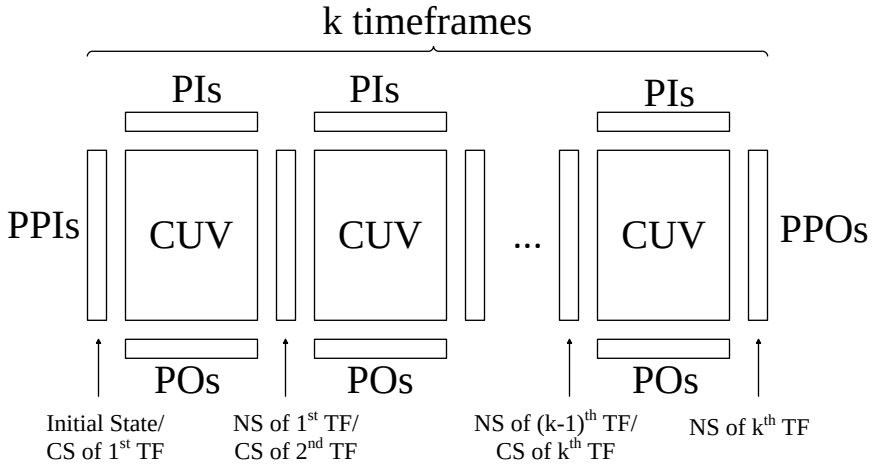

Figure 2.3: Unrolled Circuit

to propagate from the previous time-frame to the next explicitly. A two-time-frame-unrolled circuit model is the basis of the two-time-frame assume-then-verify model used for proving inductive invariants, and the multi-time-frame filtering model is used in this paper for our filtering algorithm.

2.4 Implications and Invariants

During SAT-based formal verification for sequential circuits, the relationships of signals that remain unchanged during circuit transition are denoted by invariants and can be represented by clauses. For example, suppose in a circuit, a 1 on signal A always implies a 0 on signal B, then a clause $(\neg A \vee \neg B)$ can be added to the CNF formula as an invariant constraint. Identification of a sufficient number of true invariants can effectively restrict the state space, thus proving certain properties—which is equivalence of the two circuits in our case—that holds in the restricted state space. Consider the example in Fig. 2.4, the state space inside the circle with the thickest border represents the states where the equivalence property

holds, and the inner gray area indicates the actual set of reachable states. Suppose all four invariants (Invariant A, B, C and D) labeled in the figure are true, then it is easily observed that without the need of computing the exact reachable states, the state space constrained by these four invariants is enough to prove the equivalence property. In other words, although the constrained state space is larger than the exact set of reachable states, it is well within the state set for which the equivalence property holds.

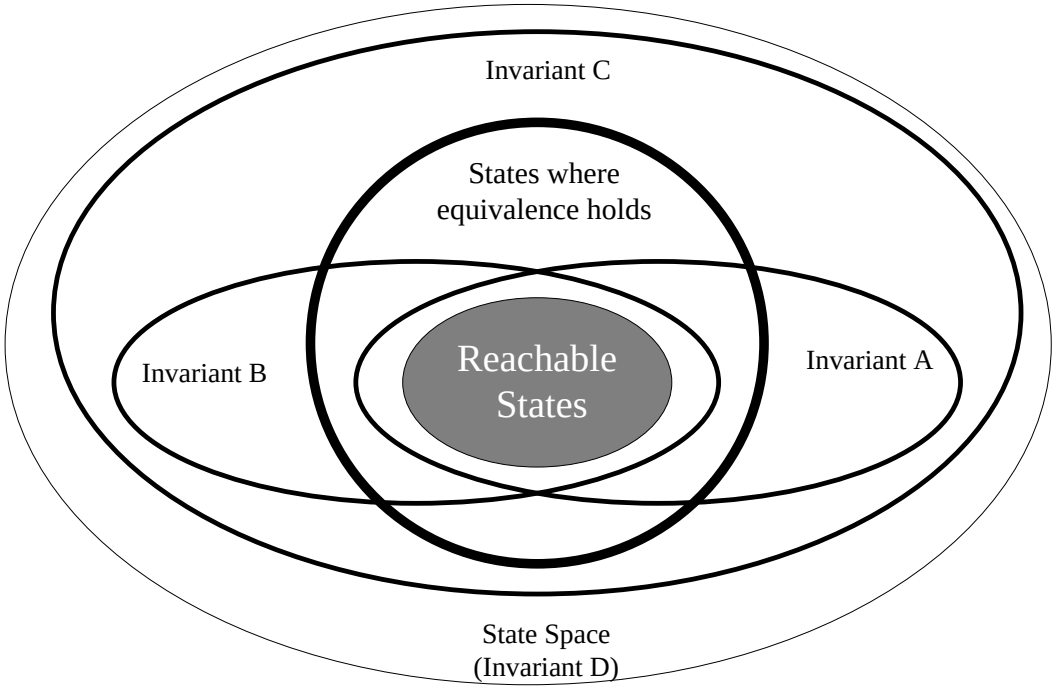

Figure 2.4: State Space

To categorize invariants, we need to categorize state space first. For a design with flip-flops unconstrained, any state is possible as a power up state, meaning there are exponential number of them. We first denote a group of states that a design can reach starting from an unknown state as reachable states. Then we denote all the other states as unreachable states. Among unreachable states, there is a group of states that can not be reached from any state. We denote this special group of unreachable states as invalid states. Based on

the categorization of state space, we can divide invariants to two kinds: static invariants that represent static signal relationships that exists in both reachable and invalid states, and inductive invariants that represent signal relationships that hold in the reachable states but may not hold in the unreachable states. Another form of static invariants are called static implications, which have been researched massively over the years[28][29][30][29]. For 2-node static implications, 3 types have been suggested for efficient computation: direct implications, indirect implications and extended backward implications. We elaborate these 3 types of static invariants in the following subsections.

2.4.1 Direct Implication

Direct implications of a signal A are the implications associated with gates where A is either input or output. There are two types of direct implications:

1), direct forward implication;

and 2), direct backward implication.

Direct forward implications are based on theory of controlling values of gates. For example, suppose A is an input of a two-input AND gate G. Signal C is the other input and signal B is the output. When A has a value 0, without knowing the value of C, we can deduce that B must have the value 0. Therefore, we denote that $A = 0$ implies $B = 0$ and 0 is a controlling value of an AND gate. The controlling values of All 4 gate types that have controlling values are shown in Table 2.1.

In Table 2.1, the third column shows the output value when one of the inputs is at a controlling value. Direct backward implications are contrapositives of direct forward implication. Consider the example that A is an input of and AND gate that outputs B, since $A = 0$ implies $B = 0$, $B = 1$ also implies $A = 1$, meaning when the output of a gate has a value

Table 2.1: Controlling Values

Gate Type	Controlling Value	Non-controlling Value	Controlling To
AND	0	1	0
NAND	0	1	1
OR	1	0	1
NOR	1	0	0

that is not a 'Controlling To' value, All of its inputs should have non-controlling values.

2.4.2 Indirect Implication

The indirect implications of a signal A being a value v are computed by plugging in both the value of the A and the direct implications of it, and performing logic simulation. Initially, the values of all other gates should be unknown, after random simulation, if the values of some gates became known(1 or 0), we denote that A being v implies these values on these signals indirectly.

Consider the example in Fig. 2.5, When A is assigned a value 1, by direct implication, both

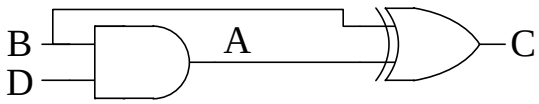

Figure 2.5: Circuit to illustrate indirect implication

its inputs B and D need to have value 1. Directly, we don't see any implication of $A = 1$ ON C. We then carry out forward logic simulation based on these value assignments. After logic simulation, we can easily see that C is evaluated to 0. Therefore, $C = 0$ is an indirect

implication of $A = 1$. To express the process of computing indirect implication of $A = v$ mathematically, the following equation is used:

$$impl[A, v] \equiv impl[A, v] \cup [logic\ simulate(impl[A, v])]$$

2.4.3 Extended Backward Implication

The extended backward implications are computed by considering both the target signal(namely, A) and the unjustified output specified gates in the implication list of the target signal. There are generally four cases depending on the gate type of the gate where the target signal is the output:

AND gate

If A is an AND gate, then

$impl[A, v] \equiv impl[A, v] \cup [\cap_{i=1}^{m} logic\ simulate(impl[A, v] \cup impl[l_i, 0])]$

The equation above means that if the implication set of signal A being v contains a signal B which is the output of an AND gate, and B has a value 0 which is a 'Controlling To' value of AND gate, we obtain an intersect of implications obtained by logic simulation after setting every inputs of the AND gate to 0 and plugging in the other implications in the implication list. That intersect of implications is the group of extended backward implications.

NAND gate

If A is a NAND gate, then

$impl[A, v] \equiv impl[A, v] \cup [\cap_{i=1}^{m} logic\ simulate(impl[A, v] \cup impl[l_i, 0])]$

The equation above means that if the implication set of signal A being v contains a signal B

which is the output of a NAND gate, and B has a value 1 which is a 'Controlling To' value of NAND gate, we obtain an intersect of implications obtained by logic simulation after setting every inputs of the NAND gate to 0 and plugging in the other implications in the implication list. That intersect of implications is the group of extended backward implications.

OR gate

If A is an OR gate, then

$impl[A, v] \equiv impl[A, v] \cup [\cap_{i=1}^{m} logic\ simulate(impl[A, v] \cup impl[l_i, 1])]$

The equation above means that if the implication set of signal A being v contains a signal B which is the output of an OR gate, and B has a value 1 which is a 'Controlling To' value of OR gate, we obtain an intersect of implications obtained by logic simulation after setting every inputs of the OR gate to 0 and plugging in the other implications in the implication list. That intersect of implications is the group of extended backward implications.

NOR gate

If A is a NOR gate, then

$impl[A, v] \equiv impl[A, v] \cup [\cap_{i=1}^{m} logic\ simulate(impl[A, v] \cup impl[l_i, 1])]$

The equation above means that if the implication set of signal A being v contains a signal B which is the output of a NOR gate, and B has a value 0 which is a 'Controlling To' value of NOR gate, we obtain an intersect of implications obtained by logic simulation after setting every inputs of the NOR gate to 0 and plugging in the other implications in the implication list. That intersect of implications is the group of extended backward implications.

For example, consider the circuit shown in Fig. 2.6. Suppose we assign a value 0 to signal

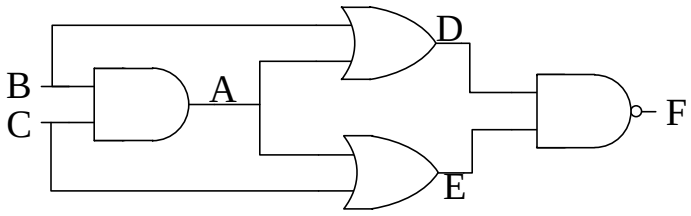

Figure 2.6: Circuit to illustrate extended backward implication

A. We can see that $A = 0$ does not have any implications, either direct implications or indirect implications. Now, let's derive extended backward implication of $(A, 0)$. Since A is an output of an AND gate, and $A = 0$, either B or C has to be 0, too. Consider the case when $B = 0$. Then $D = 0$, therefore $F = 0$ can also be deducted since D equals to the controlling value of the NAND gate. Next, consider the case when $C = 0$. Similarly, $E = 0$ and $F = 0$ can be deducted. Since $F = 0$ is achieved both by assuming $B = 0$ only and by assuming $C = 0$ only, $F = 0$ is an extended backward implication of $A = 0$.

To sum up, Static invariants are enforced by the circuit structure, and they do not need to be added as clauses explicitly (although adding some of them have been shown in [31] to be useful in speeding up SAT calls during CEC). For example, *Invariant* D (the outer-most constraint) in Fig. 2.4 is an example of a static invariant as it does not restrict search space any further. On the other hand, *Invariant* A, B, C are examples of inductive invariants and they define a smaller search space than that imposed by the circuit structure itself.

2.5 Induction-based Fix-point Calculation

2.5.1 Basics of Mathematical Induction

Mathematical induction is a method of mathematical proof that is typically used to show that a given statement is true for all natural numbers. The simplest and most common form of mathematical induction consists of two steps:

1), the basis (base case): showing that the statement holds when n is set to some valid value. Usually, $n = 0$ or $n = 1$;

and 2), the inductive step: showing that if the statement holds for some n, then the statement also holds when $n + 1$ is substituted for n.

One of the most classical examples for mathematical induction is the proof of the following statement for all natural numbers n[32]:

$$0 + 1 + 2 + \cdots + n = \frac{n(n+1)}{2},$$

It gives a formula for the sum of the natural numbers less than or equal to number n. The proof that the statement is true for all natural numbers n proceeds as follows. Let's call this statement $P(n)$.

First, we construct the basis:

Basis: Show that the statement holds for $n = 0$. $P(0)$ amounts to the statement:

$$0 = \frac{0 \cdot (0+1)}{2}.$$

In the left-hand side of the equation, the only term is 0, and so the left-hand side is simply equal to 0. In the right-hand side of the equation, $0?0 + 1)/2 = 0$. The two sides are equal, so the statement is true for $n = 0$. Thus it has been shown that $P(0)$ holds.

Then, we carry out the inductive step:

Inductive step: Show that if $P(n)$ holds, then also $P(n+1)$ holds. This can be done as follows.

Assume $P(n)$ holds (for some unspecified value of n). It must then be shown that $P(n+1)$ holds, that is:

$$(0+1+2+\cdots+n)+(n+1)=\frac{(n+1)((n+1)+1)}{2}$$

Using the induction hypothesis that P(n) holds, the left-hand side can be rewritten to:

$$\frac{n(n+1)}{2}+(n+1).$$

Algebraically:

$$\begin{aligned}\frac{n(n+1)}{2}+(n+1) &= \frac{n(n+1)+2(n+1)}{2} && (2.1)\\ &= \frac{(n+1)(n+2)}{2} && (2.2)\\ &= \frac{(n+1)((n+1)+1)}{2}. && (2.3)\end{aligned}$$

thereby showing that indeed $P(n+1)$ holds.

Since both the basis and the inductive step have been proved, it has now been proved by mathematical induction that $P(n)$ holds for all natural n. Q.E.D.

Another concept—strong induction (or sometimes called complete induction)—says that in the inductive step listed above, not only that the statement holds for $n=m$ can be assumed, but also that the statement is true for all n less than or equal to m. An example of strong induction to prove the statement "every natural number greater than 1 is a product of prime

numbers" with assumption that for a given $m > 1$ it holds for all smaller $n > 1$ is as follows[33]:

If m is prime then it is certainly a product of primes, and if not, then by definition it is a product: $m = n_1 n_2$, where neither of the factors is equal to 1; hence neither is equal to m, and so both are smaller than m. The induction hypothesis now applies to n_1 and n_2, so each one is a product of primes. Then m is a product of products of primes; i.e. a product of primes. Note both that the base case (m equal to 2) was never explicitly considered, and that the hypothesis that all smaller numbers than m are products of primes was used, since the factors of m are a priori unknown.

2.5.2 Induction-based Proof for Truthfulness of Invariants

The induction-based fix-point computation for finding true invariants can be executed based on an unrolled two-time-frame model as indicated in Fig. 2.3. When k is set to 2. We denote this model as M_{simple}. In the approach, all potential invariants from the invariant group are assumed to be true in the first time-frame, and the negation of one of the candidate invariants is imposed on the second time-frame to check if it hold or not. If not, the corresponding candidate is removed from the group. The approach needs to be iterated until a fix point is reached, where all invariants that remain in the group have been proved to be true.

Lemma 1. *For a miter circuit C_{miter}, if there exists a group of potential invariants I_{group} among flip-flops that satisfies (i) an input sequence I_{seq} can take the circuit from an unknown state to a state S_{init} where all invariants in I_{group} are true, and (ii) for all states that do not violate any invariants in I_{group}, all invariants in I_{group} remain true in the subsequent time-frames, then all invariants in I_{group} hold true in all reachable states.*

Proof. We prove this inductively. First, since S_{init} is reached by applying I_{seq} from an

unknown state, S_{init} is reachable. We are also given that all invariants in I_{group} are true in S_{init}. Next, since we are also given that any state that holds invariants in I_{group} continues to hold them in any next state that can be transitioned, starting from S_{init} that holds I_{group}, all subsequent states reachable after S_{init} with any input sequence will also hold them. Finally, since the set of reachable states forms a strongly connect component, there exists a path from S_{init} to all reachable states. Hence, all reachable states would hold I_{group}.

□

Chapter 3

Sufficiency-based Invariant Filtering Model I

In this chapter, we present a sufficiency-based filtering technique that target on filtering useful invariants out of initial potential invariant group towards restricting the search space to the area where equivalence property holds. First, random simulation is performed to generate a list of initial potential invariants. If the invariants involve internal signals, they are validated by the proposed SAT-based validation scheme. Next, the invariant filtering scheme based on a strong-induction based over-approximation model is applied to select a subset of invariants that are sufficient to prove equivalence property, or to prove that the initial potential invariants are not sufficient. For cases that the potential invariants are sufficient, experimental results show that the selected invariants are very effective in pruning enough portion of the don't-case space and prove the equivalence property.

The rest of the chapter is organized as follows. Section 3.1 introduces how we use random simulation to derive original list of potential invariants to represent potential relationships

among signals. Section 3.2 indicates the motivation of the filtering technique that we propose, following with detailed description of the technique in Section 3.3. Section 3.4 presents details in basic-induction-based fix-point calculation to prove set of true invariants. Section 3.5 describes the whole SEC framework that we propose. Section 3.6 presents experimental results and discusses how the proposed filtering technique is more efficient. Finally, we conclude the chapter in Section 3.7.

3.1 Base Case Invariants Generation

3.1.1 Basic Idea

After the miter circuit for two designs under verification is constructed, sequential logic simulation is conducted on the miter circuit which has two characteristics: i): simulation starts with unknown states; and ii): random input vectors are used during simulation.

The logic values of all signals during every simulation are recorded. Invariants are generated after simulation is finished, based on different values that have been explored. We are interested in the value combinations of signals that are missing from data recorded during simulation, which can be called missing pattern invariants. Missing pattern invariants can be used to represent all kinds of relationships that signals have, either on one signal or among multiple signals. And they can easily be converted to clauses. For example if a signal A is constant 0 during simulation, we say that $A == 1$ is missing, so a clause $\neg A$ can be used to represent this property in the CNF formula. And if a signal B always have the same value as signal C during simulation, we say that A is potentially equivalent to B. Therefore $(A, B) = (1, 0)$ and $(A, B) = (0, 1)$ missing, and so clauses $(\neg A \vee B)$ and $(A \vee \neg B)$ can be added to represent these two missing patterns.

Algorithm 1 Missing Pattern Invariants Selection

```
For specification of signal scale and maximum number of nodes in invariants, create space
for all possible signal value combinations, label all of them 0.
while simulation count has not reached the intended count do
  Logic simulate
  for all possible signal value combinations do
    if the values of the signals match the signal value combination then
      Change the label of this signal value combination to 1
    end if
  end for
end while
for all possible signal value combinations do
  if the label of it is 0 then
    This signal value combination is missing, convert it to a missing pattern invariant
  end if
end for
```

3.1.2 A Novel Implementation

In previously proposed techniques like in [11], missing pattern invariants based on random simulation are derived in such a way: a two-dimension window is first created after random simulation, with x dimension lists the signals and y dimension lists the input vector number. Each row in the database corresponds to a reachable signal assignment. After the table is created, missing pattern invariants are selected based on the values of the variables recorded in the table. For example, if a value combination $(A, B) = (0, 1)$ is not present in any row, this signal combination is treated as a missing pattern invariant. A potential problem of this approach is that: when there are too many simulation input vectors, the memory might not be enough to store the table. On the other hand, we observed that vector number doesn't really matter during invariant selection. For instance, whether the value combination $(A, B) = (0, 1)$ exists after the second vector is applied or after the fifth vector is applied makes no difference, the point is that $(A, B) = (0, 1)$ has been explored and thus can not

be a missing pattern invariant. Based on the observation, we propose a missing pattern invariant selection algorithm based on random simulation as indicated in Algorithm 1.

3.2 Motivation for Invariant Filtering

Consider Fig. 2.4, among Invariants $A,\ B,\ C,\ D$, we are only interested in inductive invariants $A,\ B,\ C$, since D is a static invariant and does not contribute to restricting the state space. Next, invariants like A and B are of our utmost interest because combining A and B alone is sufficient to restrict the state space to be inside the space where the equivalence property holds. Although a direct intuition of finding powerful invariants like A and B is to compute the search space that each invariant restricts first and then select an optimal group of them algorithmically, the computation of the space that each invariant restricts can be too complicated, making it very difficult to be put into practice. On the other hand, we observe that during SEC, whenever a SAT solver returns a satisfiable solution that indicates equivalence has not been proved, the assignments returned by the SAT solver can be extremely valuable since they represent one or more states that can violate the equivalence property. Therefore, invariants that can block out the satisfiable assignments have more potential in restricting state space towards the space where equivalence holds.

For example, suppose there are 3 invariants: I_1 denoted by $(\neg A \vee B)$, I_2 denoted by $(A \vee B)$ and I_3 denoted by $(\neg B \vee \neg C)$. In addition, the SAT solver returns a satisfiable solution with assignment 101 on signals $A,\ B,\ C$, respectively. Out of these three invariants, we say that I_1 has more value than the other two since it prevents the SAT solver from deriving this assignment by preventing A and B to be 1 and 0 simultaneously. For generality, suppose two other three-node invariants are present: I_4 denoted by $(\neg A \vee B \vee \neg C)$ and I_5 denoted by $(A \vee B \vee C)$, Out of these two invariants, we choose I_4 over I_5 for the same reason. Based

on this observation, we start with a simple algorithm to select invariants as indicated in Algorithm 2.

Algorithm 2 Simple algorithm

```
Initialize I_base = {set of all potential invariants} and I_filtered = ∅
while SAT solver returns SAT do
  Get the satisfiable assignments of all signals
  for all i ∈ I_base do
    if i violates the signal assignments then
      Add i into I_filtered and add i in the first time-frame
    end if
  end for
end while
return I_filtered
```

The problem of this simple algorithm lies in the fact that the selected invariants are applied before they are proved. Many of these invariants might be false in the first place. Or if they are true, the truthfulness may depend on some other invariants and can only be proved when these invariants are present. As a result, after proving the invariants, the invariants that are true may often be a small subset of the original set of potential invariants, which is insufficient to prove the equivalence property. Therefore, we need a better filtering technique. The following

3.3 The Proposed Filtering Technique

In this first technique, a miter circuit for the two circuits under verification is first created and unrolled for k time-frames. During selection, the invariants are only applied to the first time-frame and the goal is to select enough invariants that makes it impossible for any output-pair to be distinguished in any of the k time-frames. Consider the unrolled miter

circuit illustrated in Fig. 3.1. This is essentially to select those potential invariants that would make all the O_i signals 0, starting from any initial state of the unrolled instance.

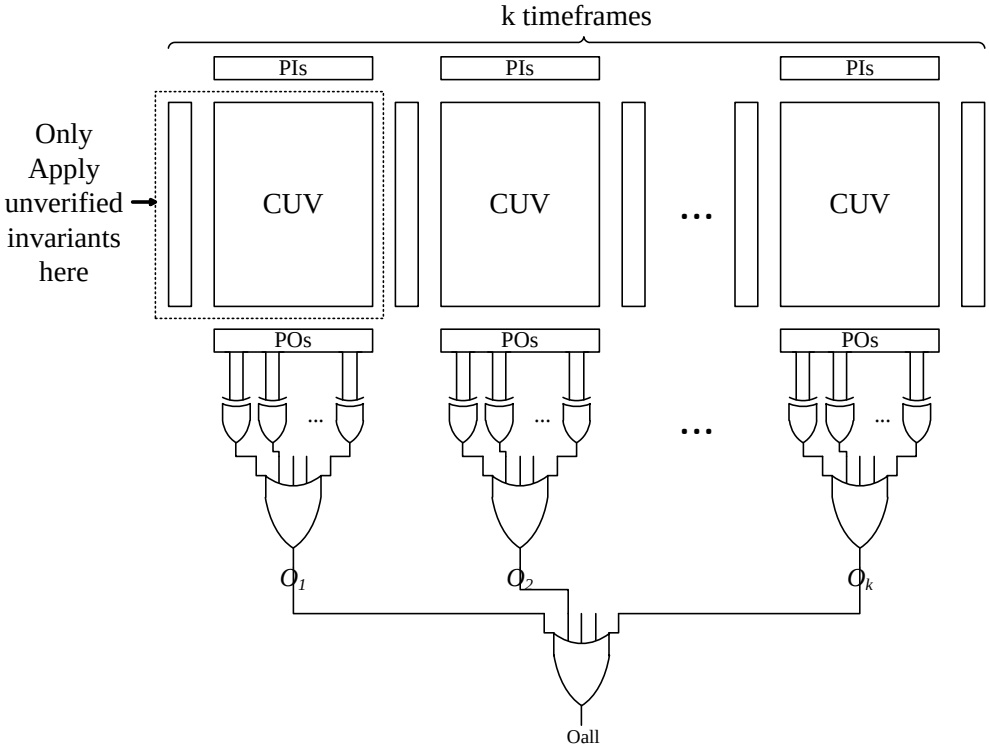

Figure 3.1: Filtering Model I

Let the unrolled miter circuit shown in Fig. 3.1 be denoted as M and the CNF formula converted by it with $O_1, O_2, ...O_{k-1}$ restricted to 0 and O_k restricted to 1 as f_M. Based on this model, we propose the invariant selecting scheme as Algorithm 3. To discuss this algorithm, we first discuss some key concepts.

Lemma 2. *For a sequential circuit, any state that can lead to a known unreachable state $s_{unreachable}$ is also unreachable.*

Proof. We prove this by contradiction. Suppose there exists a reachable state s_{init} that can

Algorithm 3 Invariant Filtering Scheme I

```
Initialize I_base = {set of all potential invariants} and I_filtered = ∅
Set currentAssumeTF = 1, currentVerifyTF = 2
while currentVerifyTF ≤ k do
  Force O_currentAssumeTF to 0 and O_currentVerifyTF to 1 in the CNF formula
  if SAT solver returns a satisfiable solution S then
    for all i ∈ I_base do
      if i violates the signal assignments then
        Add i into I_filtered and add i in the first time-frame
      end if
    end for
    if No invariant violates the signal assignment then
      if currentAssumeTF == k − 1 then
        return failure {Set of all invariants insufficient to prove equivalence}
      else
        currentAssumeTF++
        if currentVerifyTF == currentAssumeTF then
          Unrestrict O_currentVerifyTF in CNF formula
          currentVerifyTF++
        end if
      end if
    end if
  else
    Unrestrict O_currentVerifyTF in CNF formula
    currentVerifyTF++
  end if
end while
return I_filtered
```

lead to $s_{unreachable}$, $s_{unreachable}$ must also be reachable since reachable states form a terminally strongly connect component. However, this contradicts with the fact that $s_{unreachable}$ is known to be unreachable. □

In the invariant selection scheme listed in Algorithm 3, whenever there is a SAT solution for M, it means that $O_{currentVerifyTF} = 1$ must be satisfiable, indicating that there exists an initial state that can distinguish one output pair. We add *all* potential invariants to $I_{filtered}$ that can block each SAT solution. If the two circuits are indeed equivalent, according to

Lemma 2, the states leading up to *currentVerifyTF* must also be invalid. Theoretically we can select invariants according to signal assignments from time-frames 1 to *currentVerifyTF*. However, since we are only applying invariants on the first time-frame, we find that it is more effective to select invariants only according to the signal assignments on the first time-frame. Next, we increment currentVerifyTF whenever the SAT solver returns UNSAT. This allows us to include more potential invariants to falsify the formula deeper in M. In so doing, we will eventually include all those potential invariants which can set the Miter output to 1 in any of the k time-frames.

Proposition 1. *In the invariant selection scheme listed in Algorithm 3, a large portion of static invariants are avoided implicitly.*

Argument. In Algorithm 3, we are selecting invariants based on signal assignments which comply with the circuit structure. Therefore, static invariants that represent signal relationships based on circuit structure will never be selected to falsify the Satisfiable solution. However, we need to point out that since the initial state is unconstrained in our filtering model, some static invariants whose truthfulness is based on state space transition might still get selected. □

Proposition 1 is worth mentioning considering how difficult it is to compute two-node static implications[30], yet how easy a large portion of both two-node and multi-node static invariants can be avoided by our invariant selection scheme.

Definition 4. *For a group of true invariants I_{true}, we say that I_{true} is k-sufficient if adding these invariants to all k time-frames of M makes f_M UNSAT. This means that these invariants are sufficient to constrain the state space to prove the equivalence property.*

Definition 5. *For a group of potential invariants $I_{potential}$, we define them as k-sufficient if*

the true invariants $I_{true} \subseteq I_{potential}$ are k-sufficient. On the contrary, we define $I_{potential}$ as k-insufficient if $I_{true} \subseteq I_{potential}$ are not k-sufficient.

Theorem 2. *While solving f_M with constraints from I_{base} already added as on the first time-frame of M, if the SAT solver returns a satisfiable assignment S_M, then I_{base} is k-insufficient.*

Proof. We prove this by contradiction. Suppose I_{base} were *k-sufficient*, meaning that $I_{true} \subseteq I_{base}$ are *k-sufficient*. After constraining I_{base} in the first time-frame of M, I_{true} are also included. Because I_{true} are true invariants, they are also true on all successive time-frames in M. Because I_{true} is *k-sufficient*, SAT solver should return UNSAT after solving f_M, therefore contradicting with the premise that the instance was satisfiable with a satisfying assignment S_M. □

3.4 Fix-point Calculation for Proving Invariants

To find true invariants out of a potential invariant set, either it's the initial potential invariant set or the selected potential invariant set, we use a SAT-based method established on a basic-induction proof. First, we construct a two-time-frame unrolled circuit model denoted as M_{simple}, they we apply an iteration-based fix-point calculation described in Algorithm 4.

3.4.1 Unique State Constraint

Unique state constraint is defined under the notion that two consecutive states can not be the same. It's helpful in deriving more true invariants after the algorithm finishes, therefore is helpful in proving the equivalence property since the more true invariants there is, the more constraint there is. The way that this is implemented can be illustrated in Fig. 3.2.

Algorithm 4 Invariant Filtering Scheme I

```
Convert M_simple to a CNF formula
Convert unique state constraint [34] to clauses and add them to the CNF formula
Push invariants to be verified into I_group
Set needAnotherIteration to 1
while needAnotherIteration do
  Clear needAnotherIteration
  Apply all invariants in I_group to the first time-frame of M_simple
  for all invariants i ∈ I_group do
    Apply negation of i into the second time-frame of M_simple
    if An invariant that contain this invariant has been proved to be true during this
    iteration then
      Do nothing
    else
      if An invariant that this invariant contain has been dropped during this iteration
      then
        Remove i from I_group
      else
        Call SAT solver
        if SAT Solver returns SAT then
          Set needAnotherIteration
          Remove i from I_group
        end if
      end if
    end if
  end for
end while
return I_group
```

In Fig. 3.2, the corresponding state elements in each timeframe is connected by XOR gates first, these XOR gates are then connected by an OR gate. By restricting the value of the OR gate to 0, we are excluding the possibility that two states are the same during SAT call.

3.4.2 Accelerating Heuristics

We first denote that an invariant i_1 *contain* another invariant i_2 if *search space restricted by* $i_1 \supseteq$ *search space restricted by* i_2. For example, suppose i_1 is a two-node invariant on signals

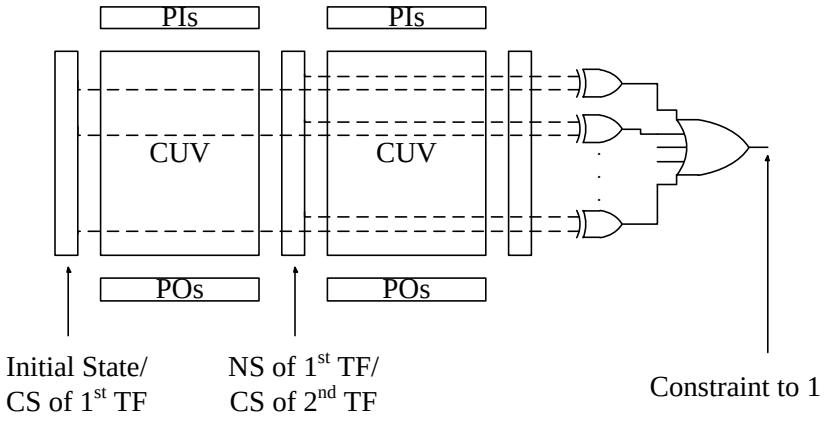

Figure 3.2: Unique State Constraint

A and B and is expressed as $(\neg A \vee B)$ and i_2 is a three-node invariant on signals A, B and C expressed as $\neg A \vee B \vee C$, it's obvious that i_1 *contain* i_2. Therefore, during one iteration in Algorithm 4, if i_1 has been verified to be true, the truthfulness of i_2 can be deducted directly. On the other hand, if i_2 is falsified, then i_1 can also be falsified directly. This reasoning saves a lot of time comparing to verifying every invariant during the assume-then-verify approach. The reason why we still need to verify i_2 when i_1 is already in the base case and i_1 contains i_2 is because if i_1 is falsified, there is still some chance for i_2 to be true, which might contribute to verifying equivalence property.

Another accelerating heuristics that we applied in Algorithm 4 is that during one iteration, when an invariant is falsified, theoretically the constraint of it needs to be removed in the assumption phase right away, and all invariants(including the invariants that have been verified in that iteration) need to be re-verified since the assumption have changed. However, the way we deal with the situation in Algorithm 4 is that whenever an invariant is falsified, its constraint is not removed in the assumption phase right away, and instead the rest of the invariants are verified before another iteration is carried out with only the constraint of true invariants verified in the last iteration added in the assumption phase. In this way, the

number of iterations needed in the assume-then-verify is greatly reduced and thus the time taken by the whole assume-then-verify process is much less.

3.5 SEC framework

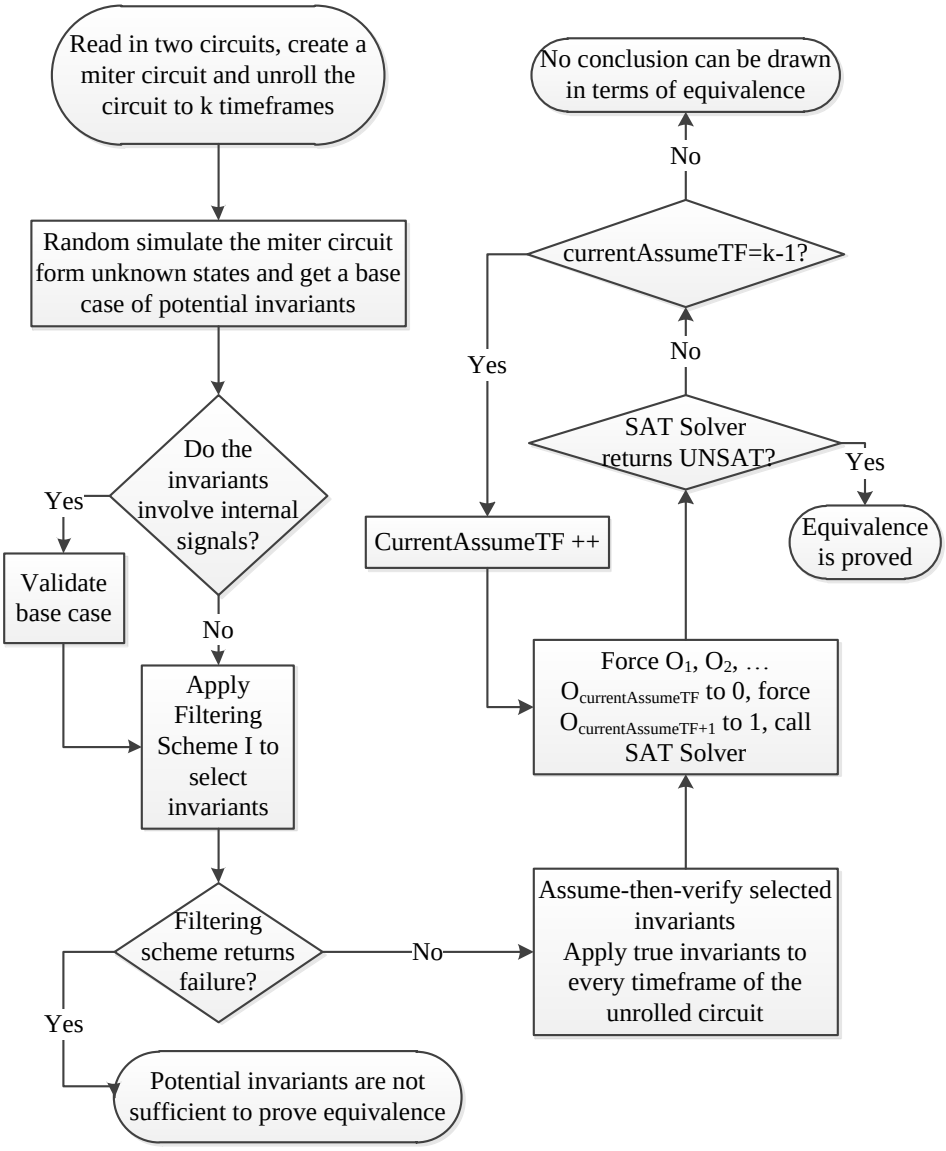

Figure 3.3: Proposed SEC Framework

We propose the entire sufficiency-based SEC framework that can be embedded with Filtering Technique I as illustrated in Fig. 4.4.

3.6 Experimental Results

The proposed SEC framework was implemented in C++ and the performance was evaluated on an Intel Pentium 4 3.4 GHz, 1GB of RAM, running Ubuntu 10.04. Experiments are carried on a suite of hard-to-verify SEC benchmarks generated by using drastically different state encodings (gray and one hot, namely) for ITC99 circuits. These SEC instances are hard as there are few or no internal equivalent points.

We derive invariants among flip-flops to illustrate the power of our filtering techniques. Due to memory issues that too many invariants may cause to the SAT solver, we limit the number of initial potential invariants to under 200,000. Therefore for all circuits, we derive 2-node, 3-node and 4-node invariants for b01, b02 and b06, 2-node and 3-node for b03, b04, b08, b09, b10 and b11, and only 2-node for the rest of the circuits including b05, b07, b11, b12 and b13.

The results are shown in Table 3.1. The first column lists the names of the miter circuits. The second column reports the number of flip-flops in the circuits, for example, 5/10 means there are 5 flip-flops in b01_gray and 10 flip-flops in b01_onehot. The third column shows the number of time-frames used in the filtering models we proposed. For b09 and b11, we used both the filtering technique. The fourth column shows the number of initial potential invariants extracted from random simulation. The fifth column shows the number of invariants after filtering, followed with the sixth column reporting the percentage of selected invariants over the original potential invariants. The seventh column shows if equivalence can be proved using initial set of potential invariants, while the eighth column shows whether equivalence

Table 3.1: Results evaluation for proposed filtering schemes

Benchmarks	# FFs	TFs	# Orig. Inv.	# Redu. Inv.	% of Inv. Slec.	P1[1]	P2[2]
b01_gray_onehot	5/10	20	14319	3707	25.9%	Y	Y
b02_gray_onehot	4/8	20	6325	1708	27.0%	Y	Y
b03_gray_onehot	30/31	20	118311	35847	30.3%	Y	Y
b04_gray_onehot	69/70	20	198500	53459	26.9%	Y	Y
b05_gray_onehot	32/34	20	3451	1697	49.2%	N	N
b06_gray_onehot	9/13	20	73341	9108	12.4%	Y	Y
b07_gray_onehot	51/53	20	12207	6010	49.2%	N	YN
b08_gray_onehot	21/23	20	45842	10813	23.6%	Y	Y
b09_gray_onehot	28/30	20	76249	25569	33.5%	Y	N
b10_gray_onehot	17/24	20	23105	7903	34.2%	N	N
b11_gray_onehot	35/40	20	184706	47395	25.7%	N	YN
b12_gray_onehot	199/220	20	239238	118598	49.6%	N	N
b13_gray_onehot	29/34	20	2610	1308	50.1%	N	N

[1] Sufficient to prove equivalence using initial potential invariants

[2] Sufficient to prove equivalence using selected potential invariants

can be proved by the reduced list of invariants selected by our filtering techniques. 'Y' means sufficient to prove and 'N' means insufficient, 'YN' means that the initial potential invariants actually passed the filter, but the filtered invariants failed to prove equivalence. We can see that out of all 7 instances that can be proved by using all initial potential invariants, our filtering approach can also prove 6 of them. Moreover, for circuits that cannot be proved by using initial set of potential invariants, our approach can detect this by not being able to select sufficient invariants. For b07 and b11, although the filtered invariants were not able to prove invariants, the fact that the original set of invariants passed the filter makes our

proposed technique a fail.

Table 3.2: Performance comparison of proposed SEC framework with/without Filtering Technique I

Benchmarks	T-filter(s)[3]	T-P1(s)[4]	T-P2(s)[5]	T-total1(s)[6]	T-total2(s)[7]	Speedup(x)
b01_gray_onehot	0.340	0.132	0.117	2.960	3.284	0.90
b02_gray_onehot	0.032	0.052	0.100	1.280	1.436	0.88
b03_gray_onehot	1.608	57.824	40.939	89.282	72.737	1.23
b04_gray_onehot	191.767	234.503	36.083	641.160	631.855	1.01
b05_gray_onehot	13.041	633.787	0	636.787	16.253	39.18
b06_gray_onehot	0.736	5.828	1.22	24.053	20.205	1.19
b07_gray_onehot	12.886	3608.407	1028.736	3616.098	1049.313	3.45
b08_gray_onehot	8.444	392.996	128.584	405.761	149.813	2.71
b09_gray_onehot	20.817	550.766	278.709	579.880	328.640	1.76
b10_gray_onehot	45.907	101.546	0	112.807	57.168	1.97
b11_gray_onehot	70.844	16454.805	0	16518.777	134.816	122.53
b12_gray_onehot	50.731	72000(TO)	72000(TO)	72000(TO)	72000(TO)	N/A
b13_gray_onehot	7.085	210.953	0	208.065	9.973	20.86

[3] Time taken by filtering schemes

[4] Time taken by proving equivalence using initial invariants

[5] Time taken by proving equivalence using selected invariants

[6] Total time taken by SEC framework without filter

[7] Total time taken by SEC framework with filter(s)

Table 3.2 shows the performance comparison of proposed SEC framework with/without filters. The first column shows the names of the benchmarks. The second column shows

the time taken by the filtering schemes that we applied. Columns 3 and 4 show the time for fix-point calculation of inductive invariants for initial invariants and filtered invariants, respectively. Note that a value of 0 means the initial invariants can not pass the filters, therefore the original set of invariants is insufficient to prove equivalence is directly concluded. The fifth and the sixth column show the total time taken by the SEC framework without filters and with filters. Finally, Column 7 shows the speedup of the total run time for SEC framework with filters over without filters.

For cases that equivalence can be proved by the original potential invariants, consider b08_gray_onehot: after random simulation, 45842 potential invariants are derived. After applying Filtering Technique I, only 23.6% out of them are selected. Fix-point calculation of the selected invariants takes only 32.7% of the time used compared with the original set of potential invariants — resulting a 2.7× speedup of overall runtime. Note that the selected invariants are still sufficient to prove equivalence. The results for other instances except for b09 show similar trends.

Next, consider the circuit b05_gray_onehot among those cases that equivalence could not be proved with the original set of potential invariants. After random simulation, 3451 potential invariants are obtained. If all of these 3451 potential invariants were true invariants, adding them to all time-frames of M would make the formula f_M UNSAT. However, it's almost always the case that only a subset of the initial potential invariants are true inductive invariants. Fix-point calculation of these inductive invariants and applying all true invariants in attempting to solve the SEC instance took 633.787 seconds, while our filtering technique took only 13.041 seconds to conclude that the set of potential invariants are insufficient to prove equivalence, saving the trouble to actually find the true inductive invariants and perform the SEC. A speedup of more than 39× was achieved in this case. The results are similar for other instances(except for b11) for which equivalence could not be proved.

For circuits such as b01_gray_onehot and b02_gray_onehot, the proposed approach takes more or less the same amount of time as the SEC framework without our filter. However, these circuits are small. For all other cases, the proposed SEC framework takes less time with filter than without filter.

3.7 Summary

We presented a novel filtering technique for processing potential invariants that select useful invariants towards proving the equivalence property of the miter circuit. The filter can work on all kinds of invariants, e.g., either two-node or multi-node, either among only flip-flops or among all internal signals. Experimental results show the power of the proposed approaches: For most cases, when the original set of potential invariants is able to prove equivalence, the proposed approaches can always select a smaller subset and still be able to prove equivalence; on the other hand, when the original set of potential invariant is insufficient to prove equivalence, the proposed filters can quickly prove this by being unable to select a sufficient set of invariants.

Chapter 4

Sufficiency-based Invariant Filtering Model II

In this chapter, we present a second sufficiency-based filtering technique that also target on filtering useful invariants out of initial potential invariant group towards restricting the search space to the area where equivalence property holds. This filtering technique can work on the same group of potential invariants derived by random simulation as Filtering Technique I: random simulation is performed to generate a list of initial potential invariants. Also if the invariants involve internal signals, they are validated by the proposed SAT-based validation scheme. What's different from Filtering Technique I is that this second filtering technique is based on a basic-induction over-approximation model to select a subset of invariants that are sufficient to prove equivalence property, or to prove that the initial potential invariants are not sufficient. The proposed filtering technique is highly flexible in use: it can both be combined with Filtering Technique I and be used individually. We present experimental results in two parts: Part I shows that when combining with Filtering Technique I, the filtering technique can solve tough cases that Filtering Technique I can not solve successfully

alone; Part II shows that the SEC framework with proposed filtering technique alone can solve problems more efficient than without filter.

Since we use the same SEC framework for this technique as Filtering Technique I, we will not repeatedly introduce how we use random simulation to derive original list of potential invariants to represent potential relationships among signals or details in basic-induction-based fix-point calculation to prove set of true invariants, which can be found in Section 3.1 and 3.4 separately. However, we will state the necessity of validity check if potential invariants involve internal signals and propose a simple SAT-based method for it in Section 4.1 since Part II of experimental results involves internal signals. Next, Section 4.2 indicates the motivation of the filtering technique that we propose, following with detailed description of the technique in Section 4.3. Section 4.4 describes the total SEC framework that we use. Section 4.5 presents experimental results to show the power of both the proposed filtering technique and the combination of the proposed filtering technique with Filtering Technique I. Finally, we conclude the chapter in Section 4.6.

4.1 Base Case Invariants Validity Check

Since potential invariants are derived from random simulation, there could be many potential invariants extracted as a result. Every candidate potential invariant extracted has the following characteristic: it has never been violated by any of the stimuli in the random simulation trace. Hence, if a potential invariant involves only state variables (flip-flops), then it is consistent with all states reached thus far by random simulation. And the fact that it is true in at least one such reachable state suffices for a valid base case in the inductive proof. On the other hand, if a potential invariant involves internal signals, the random trace is insufficient to serve as a base case, since we have not exhausted all possible input combi-

nations for a given state to ensure that the potential invariant is indeed true for all possible inputs within at least one known reachable state. Therefore, for this kind of invariant, we propose the validity Check scheme based on Fig. 4.1.

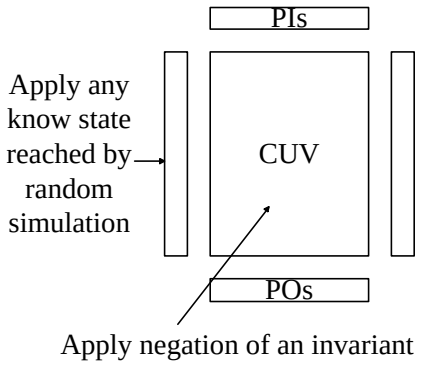

Figure 4.1: Base Case Validation Model

The scheme starts with converting a one-time-frame miter circuit to CNF formula. Then, we restrict the state of the one timeframe model with any known state reached by random simulation. Since random simulation starts with unknown states, any state reached by it is a reachable state. Next, for any invariant involving internal signals, we apply the negation of it to the model and call SAT solver. If SAT solver returns UNSAT, which means the potential invariant holds for all possible input vectors on this reachable state, the invariant passes the validity check. If SAT solver returns SAT, which means the invariant does not hold for at least one input vector, it does not pass the validity check and is dropped. After that, the clauses added for the negation of invariant is deleted and another invariant can then be evaluated.

For example suppose there is a two-node invariant on two signals(namely, A and B) and at least one of them is an internal signal. The invariant is expressed as $(\neg A \vee B)$, meaning that

$A = 1$ and $B = 0$ can not happen at the same time, the negation of it can be expressed as two one-node clauses: (A) and (B). Therefore, if (A) and (B) are added to the CNF formula and the SAT solver returns UNSAT, it means that under the current specific reachable state, there is no input vector that can make $A = 1$ and $B = 0$ at the same time, thus the invariant is valid. On the contrary, if the SAT solver returns SAT, it means that at least one input vector can violate the invariant by satisfying $A = 1$ and $B = 0$. For generality, for three other possible two-node invariants on A and B, the corresponding negation of them are expressed as follows:

Negation of $(A \vee \neg B)$: $(\neg A)$ and (B)

Negation of $(\neg A \vee B)$: (A) and $(\neg B)$

Negation of $(A \vee B)$: $(\neg A)$ and $(\neg B)$

4.2 Motivation of a more exact filtering model

Analyzing the experimental results, we can see that for b09, the initial potential invariants are sufficient to prove equivalence as indicated in P1, however, the invariants selected by Filtering Technique I are not as indicated in P2. Moreover, for b07 and b12, although the initial potential invariants turned out to be insufficient to prove equivalence(P1), Filtering Technique I still selected 'sufficient' invariants. The reason why these cases happen is because the setup of Filtering Model I is essentially a strong induction on invariants. Therefore it only guarantees that if the invariants hold for k time-frame are able to restrict state space such that equivalence property holds in f_M. However, during the fix-point calculation of inductive invariants from $I_{filtered}$, basic induction is used instead of strong induction to be more time-efficient. Therefore, the true invariants obtained by the fix-point calculation can be fewer such that they no longer are sufficient. Thus for these circuits, we need a second

filtering technique that is based on a model using basic induction only.

4.3 The proposed filtering technique

A general model that Filtering Technique II relies on is shown in Fig. 4.2. If the invariants only involve flip-flops, the model can be simplified to Fig. 4.3.

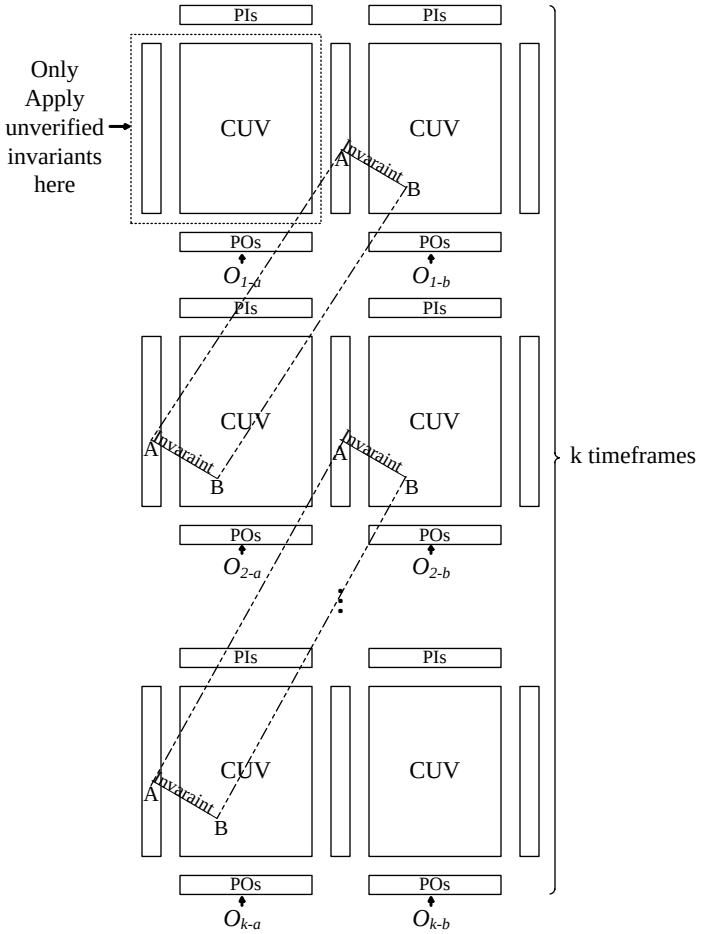

Figure 4.2: General Filtering Model II

In both general model and simplified model, 'k timeframes' does not mean actual timeframes,

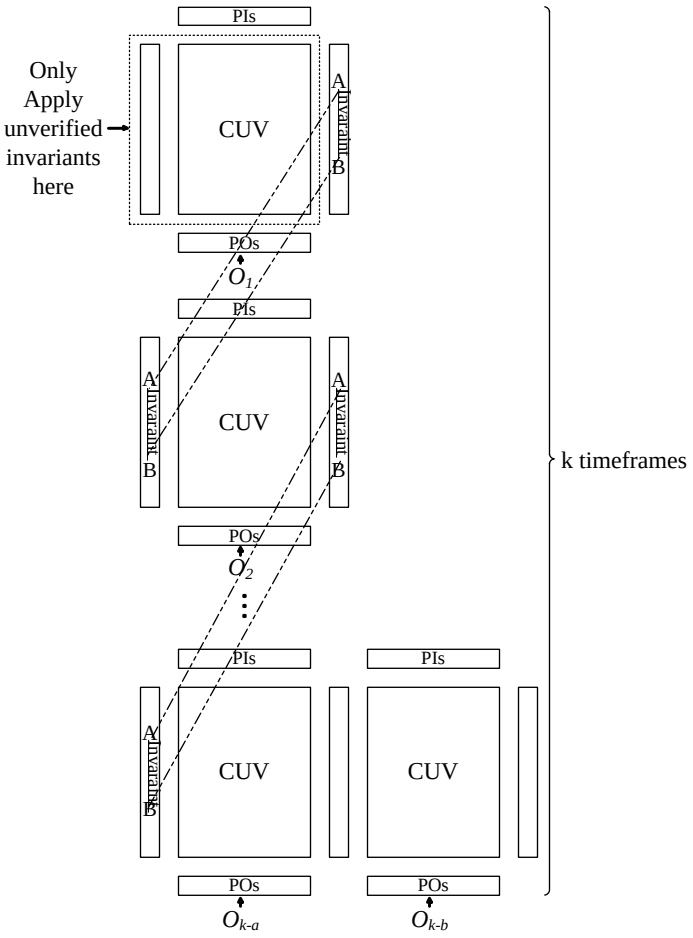

Figure 4.3: Simplified Filtering Model II

but rather time-frames that functionally count. We denote a timeframe as $l(1 \leq l \leq k) - a \ or \ b(first \ or \ second \ time - frame)$ for convenience. Comparing Fig. 4.2 to Filtering Model I, we break the state element connections after every two time-frames. We also insert a time-frame to pass only the constraints from their previous time-frames, thus enabling exact basic induction implicitly.

The connections between the inserted time-frames and their previous time-frames are denoted by dashed line in Fig. 4.2 and Fig. 4.3 to indicate the specific connection for transferring constraints. For example, let's denote a signal X at time-frame i as X_i; if an invariant

$(\neg A \vee \neg B)$ is applied on time-frame $1-a$, it means $(A_{1-a},\ B_{1-a}) = (1,\ 1)$ is impossible. To pass this constraint imposed by A_{1-b} and B_{1-b} to time-frame $2-a$, the following clauses (denoted by *transferring clauses*) needs to be added:

$$(\neg A_{2-a} \vee \neg B_{2-a} \vee A_{1-b})$$
$$(\neg A_{2-a} \vee \neg B_{2-a} \vee B_{1-b})$$

The above two clauses impose the following: whenever $(A_{1-b},\ B_{1-b})$ is $(0,\ 0)$, $(0,\ 1)$ or $(1,\ 0)$, $(A_{2-a},\ B_{2-a})$ also has to be one of the three combinations. Conversely, whenever $(A_{2-a},\ B_{2-a})$ is $(1,\ 1)$, $(A_{1-b},\ B_{1-b})$ also has to be $(1,\ 1)$. Therefore, if $(\neg A_{1-b}, \neg B_{1-b})$ is true, then $(\neg A_{2-a}, \neg B_{2-a})$ is implicitly added; if $(\neg A_{1-b}, \neg B_{1-b})$ is not true, then there is no constraint on $(A_{2-a},\ B_{2-a})$. To complete the discussion, *transferring clauses* corresponding to 3 other possible two node constraints are listed below:

$(\neg A \vee B)$:

$$(\neg A_{2-a} \vee B_{2-a} \vee A_{1-b})$$
$$(\neg A_{2-a} \vee B_{2-a} \vee \neg B_{1-b})$$

$(A \vee \neg B)$:

$$(A_{2-a} \vee \neg B_{2-a} \vee \neg A_{1-b})$$
$$(A_{2-a} \vee \neg B_{2-a} \vee B_{1-b})$$

$(A \vee B)$:

$$(A_{2-a} \vee B_{2-a} \vee \neg A_{1-b})$$
$$(A_{2-a} \vee B_{2-a} \vee \neg B_{1-b})$$

For generality, the *transferring clauses* for 2 examples out of all 8 possible three-node invariants on signals A, B and C are:

$(\neg A \vee \neg B \vee \neg C)$:

$$(\neg A_{2-a} \vee \neg B_{2-a} \vee \neg C_{2-a} \vee A_{1-b})$$
$$(\neg A_{2-a} \vee \neg B_{2-a} \vee \neg C_{2-a} \vee B_{1-b})$$
$$(\neg A_{2-a} \vee \vee \neg B_{2-a} \vee \neg C_{2-a} \vee C_{1-b})$$

$(\neg A \vee \neg B \vee C)$:

$$(\neg A_{2-a} \vee \neg B_{2-a} \vee C_{2-a} \vee A_{1-b})$$

$$(\neg A_{2-a} \vee \neg B_{2-a} \vee C_{2-a} \vee B_{1-b})$$

$$(\neg A_{2-a} \vee \vee \neg B_{2-a} \vee C_{2-a} \vee \neg C_{1-b})$$

Let the unrolled miter circuit shown in Fig. 4.2. be denoted as M' and the CNF formula converted by it with $O_{1-a}, O_{2-a}, ...O_{k-a}$ restricted to 0 and $O_{1-b}, O_{2-b}, ...O_{k-b}$ restricted to 0 restricted to 1 as $f_{M'}$.

Definition 6. *We define* APPLY-2 *as a function that adds an invariant i to M' directly in the first time-frame of M' and for all subsequent time-frames in M', add transferring clauses between time-frames $l - b$ and (l+1)$-a$.*

Based on this Filtering Model II, we propose the invariant selecting scheme indicated in Algorithm 5.

Note that in Algorithm 5, if the proposed technique is not combined with Filter technique I, during the first iteration of selecting invariants based on the SAT solution, because there is no *transferring clauses* added yet, there is no connection between any of the functional time-frames. Moreover, since we are only restricting O_{k-b} to 1, there is no point of selecting invariants based on satisfiable signal assignments on time-frames other than $k - a$, since blocking signal assignments on those time-frames does not necessarily contribute to proving equivalence. However, during the successive iterations of invariants selection, we select invariants based on signal assignments in all time-frames, since there are transferring clauses connecting functional time-frames and all functional time-frames now lead to the state on $k - b$ where equivalence does not hold. Therefore invariants that can violate signal assignments on any time-frame can contribute to restricting the search space to the space where equivalence property holds. Moreover, too many transferring clauses can be added during invariant selection. For example, in a 30 functional time-frame model, for every three-node

Algorithm 5 Invariant Filtering Scheme II

```
if combining with Filtering Scheme I then
  Add all invariants from I_filtered achieved by Algorithm 3 to time-frame 1 − a
end if
Force all O_{l−a}, 1 ≤ l ≤ k to 0 and all O_{k−b} to 1
Call SAT Solver
if not combining with Filtering Scheme I then
  Get the satisfiable assignments of signals on time-frame k − a
  for all i ∈ I_base do
    if i violates signal assignments on time-frame k − a then
      Add i into I_filtered
      APPLY-2 i to M_2
    end if
  end for
end if
while SAT Solver returns SAT do
  Get the satisfiable assignments of all signals
  for all i ∈ I_base do
    if i violates signal assignments in any time-frame then
      Add i into I_filtered
      APPLY-2 i to M_2
    end if
  end for
  if No invariant is selected then
    return failure {Potential invariants are not enough to prove equivalence}
  end if
  Call SAT Solver
end while
return I_filtered
```

invariant selected, 3 transferring clauses need to be added between every two successive functional time-frames, making the total number of transferring clauses added 87. Therefore, to save time during invariant selection, we need to control the number of iterations of invariant selection to achieve as few SAT calls as possible, and selecting invariants according to signal assignments on all time-frames enables us to do that.

Theorem 3. *While solving $f_{M'}$ with invariants from I_{base} already added on M' (via APPLY-2 function), if the SAT solver still returns a satisfiable solution $S_{M'}$, then I_{base} is 2-insufficient.*

Proof. We prove this by contradiction: Suppose I_{base} is *2-sufficient*. Therefore, $I_{true} \subseteq I_{base}$ is *2-sufficient*. After adding I_{base} to M' via the APPLY-2 function, I_{true} are also added, since I_{true} is a subset of I_{base}. Because I_{true} contains true invariants, they are passed to all time-frames in M' including time-frames that have only connections with previous time-frames by *transferring clauses* of I_{base}. Considering the last two time-frames of M', because I_{true} is *2-sufficient* and has been added to the circuit model, the SAT solver should return UNSAT for $f_{M'}$, contradicting the premise that the formula was satisfiable. □

The proposed filtering model essentially does basic-induction-based assume-then-verify from the first functional time-frame to the last functional-timeframe without explicitly assuming and verifying single invariants. Consider Fig. 4.2. Suppose a group of invariants I_{group} is applied on time-frame $1-a$, a subset of them should still hold on time-frame $1-b$, since the current state elements of time-frame $1-b$ are connected with the next state elements of time-frame $1-a$ directly. We denote them as a set of invariants I_{1-hold}, and $I_{1-hold} \subseteq I_{group}$. Because of the *transferring clauses* between time-frame $1-b$ and $2-a$, all invariants in I_{1-hold} also hold on time-frame $2-b$. Without I_{1-hold}, there is no other constraint on time-frame $2-a$. Therefore, the group of invariants that still hold on time-frame $2-b$ $I_{2-hold} \subseteq I_{1-hold}$ is equivalent with the group of invariants that hold after the second iteration in basic-induction-based assume-then-verify process on I_{group}. To extend the idea, the group of invariants that hold on time-frame $k-a$ represents the group of invariants that hold after $(k-1)^t h$ iteration of assume-then-verify. Therefore, if there was a way to prove $I_{(k-1)-hold} \equiv I_{k-hold}$, a fix-point have already been reached. And if equivalence property holds on time-frame $k-b$ when O_{k-a} is restricted to 0, equivalence property is proved directly without the need of an explicit assume-then-verify process to derive true invariants.

4.4 SEC framework

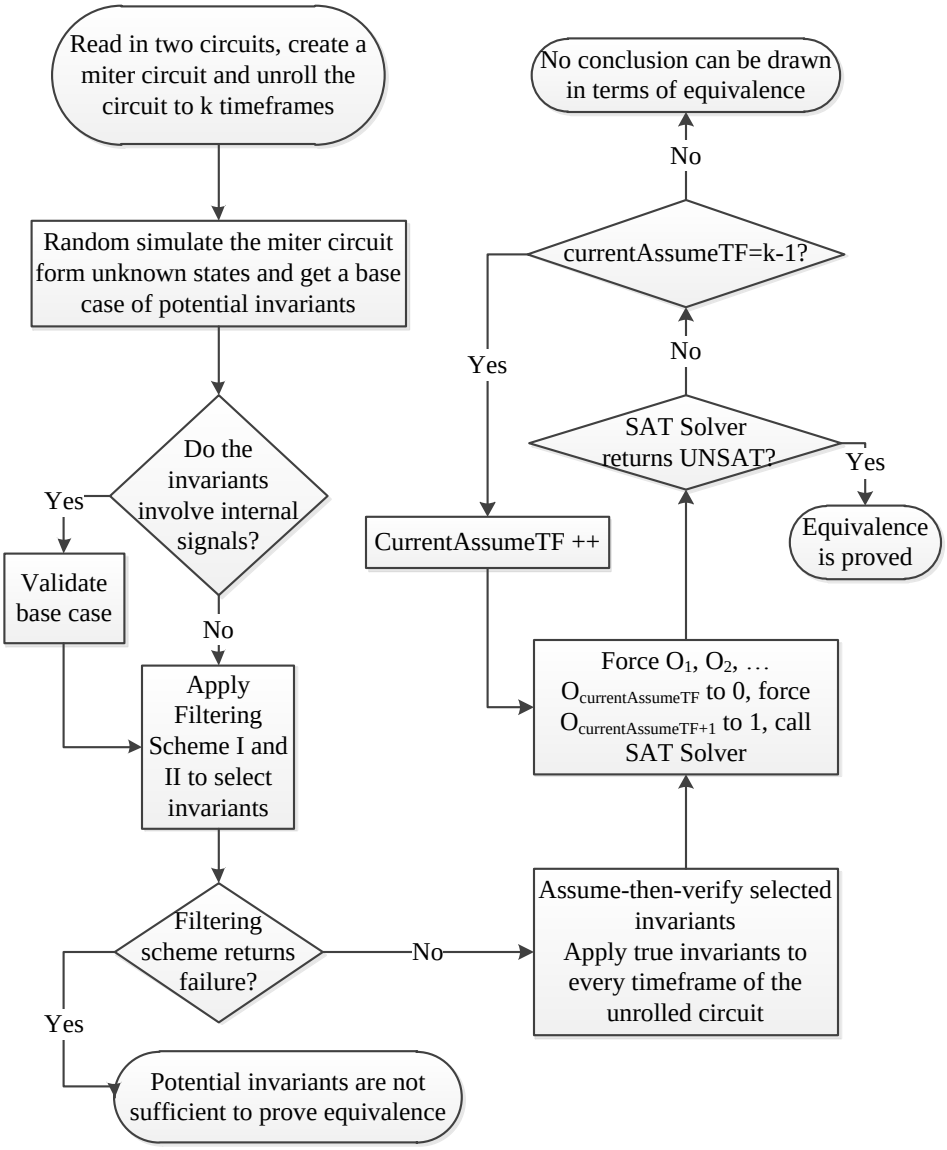

Figure 4.4: Proposed SEC Framework

We propose the entire sufficiency-based SEC framework that can be embedded with either Filtering Technique II or a combination of Filtering Technique I and II as indicated in Fig.

4.4.

4.5 Experimental Results

The proposed SEC framework was implemented in C++ and the performance was evaluated on an Intel Pentium 4 3.4 GHz, 1GB of RAM, running Ubuntu 10.04. Experiments are carried on a suite of hard-to-verify SEC benchmarks generated by using drastically different state encodings (gray and one hot, namely) for ITC99 circuits. These SEC instances are hard as there are few or no internal equivalent points.

4.5.1 Part I

We first target on the three circuits b07, b09 and b12 that were failure cases in the previous chapter. We use the same experimental setup as in previous chapter: For b09, we derive 2-node and 3-node invariants among flip-flops; For b07 and b12, we derive only 2-node invariants.

Table 4.1: Results evaluation for combining Filtering Technique I & II

Benchmarks	# FFs	TFs	# Inv. Orig.	# Inv. I	# Inv. I&II	P1 [8]	P2 [9]	P23 [10]
b07_gray_onehot	51/53	20/29	12207	6010	7291	N	YN	N
b09_gray_onehot	28/30	20/29	76249	25569	45451	Y	N	Y
b12_gray_onehot	199/220	20/29	239238	118598	119188	N	YN	N

[8] Sufficient to prove equivalence using initial potential invariants

[9] Sufficient to prove equivalence using invariants selected by Filtering Technique I

[10] Sufficient to prove equivalence using invariants selected by Filtering Technique I & II

Table 4.1 shows the results. The first column lists the names of the miter circuits. The second column reports the number of flip-flops in the circuits, for example, 5/10 means there are 5 flip-flops in b01_gray and 10 flip-flops in b01_onehot. The third column shows the number of time-frames used in the filtering models we proposed. The number before '/' indicates number of time-frames in the first filtering model and the second number is number of time-frames in the second filtering model. The fourth column shows the number of initial potential invariants extracted from random simulation. The fifth column shows the number of invariants selected after Filtering Technique I, followed with the sixth column reporting the number of selected invariants after the combination of Filtering Technique I & II. The seventh column shows if equivalence can be proved using initial set of potential invariants, while the eighth column and the ninth column show whether equivalence can be proved by the reduced list of invariants selected by Filtering technique I only and a combination of Filtering Technique I and II, respectively. 'Y' means sufficient to prove, 'N' means insufficient and 'YN' means that the initial potential invariants actually passed the filter, but the filtered invariants failed to prove equivalence. We can see that for b09 that can be proved by using all initial potential invariants, our filtering approach with combination of Filtering Technique I can also prove all of them. On the other hand, for b07 and b12 that cannot be proved by using initial set of potential invariants, our approach can detect that by not being able to select sufficient invariants to pass the filter(s). Therefore comparing to the failed results on all 3 cases by using only Filtering Technique I, the proposed approach is a win.

Table 4.2 shows the performance comparison of proposed SEC framework with/without filters. The first column shows the names of the benchmarks. The second column shows the time taken by the filtering schemes that we applied. Columns 3 and 4 show the time for fix-point calculation of inductive invariants for initial invariants and filtered invariants,

Table 4.2: Performance comparison of proposed SEC framework with/without filters

Benchmarks	T-filter(s)[11]	T-P1(s)[12]	T-P23(13)[8]	T-total1(s)[14]	T-total23(s)[15]	Speedup(×)
b07_gray_onehot	75.277	3608.407	0	3616.098	80.697	44.81
b09_gray_onehot	1331.171	550.766	620.555	579.880	1980.840	0.29
b12_gray_onehot	218.245	72000(TO)	0	72000(TO)	327.668	>219.73

[11] Time taken by both filtering schemes

[12] Time taken by fix-point calculation of inductive invariants using initial pot. invariants

[13] Time taken by fix-point calculation of inductive invariants using filtered pot. invariants

[14] Total time taken by SEC framework without filter

[15] Total time taken by SEC framework with filters

respectively. Note that a value of 0 means the initial invariants can not pass the filters, therefore the original set of invariants is insufficient to prove equivalence is directly concluded. The fifth and the sixth column show the total time taken by the SEC framework without filters and with filters. Finally, Column 7 shows the speedup of the total run time for SEC framework with filters over without filters.

From the results we can see that for b09_gray_onehot, Filtering Technique II took more time which causes the proposed approach to cost more overall. Also, although only 45451 out of 25569 invariants are selected for assume-then-verify, the time taken by the fix-point calculation of the selected invariants is still longer. We found that the reason lies in the fact that fix-point calculation of the selected invariants takes more iterations. Nevertheless, the fact that the selected 59.6% of the original invariants can still prove equivalence property still shows the power of the proposed approach. Moreover, for b07, there are 12207 initial potential invariants. Fix-point calculation of these inductive invariants and applying all true

invariants in attempting to solve the SEC instance took 633.787 seconds, while combining our filtering technique with Filtering Technique I took only 75.277 seconds to conclude that the set of potential invariants are insufficient to prove equivalence, saving the trouble to actually find the true inductive invariants and perform the SEC. Therefore, a speedup of more than 44× was achieved in this case. The result is similar for b12 for which equivalence also could not be proved, where a speed up of >219.73× is achieved for total runtime.

4.5.2 Part II

In this part, we show the power of the proposed filtering technique alone without combining with Filtering Technique I. We still target on the same group of ITC circuits and derive invariants among not only flip-flops, but also internal signals to show the flexibility of our proposed approach. However, since we now have a much larger number of signals to derive invariants, the number of invariants derived can easily explode. To control the number of initial potential invariants to a reasonable range, we derive only 2-node invariants among several smaller circuits.

Results are showed in Table 4.3. Similar to the results shown in Part I, we use the first two columns to show circuit information. The third column shows the number of functional time-frames used in the filtering model. Because the invariants involve internal signals, we use the general filtering model, so the actual time-frames used is two times the time-frame number shown. Column 4, 5 and 6 show the number of initial invariants, number of selected invariants and the proportion of selected invariants over initial invariants, respectively. The seventh column shows if equivalence can be proved using initial set of potential invariants, while the eighth column show whether equivalence can be proved by the reduced list of invariants selected by Filtering technique II. 'Y' means sufficient to prove and 'N' means

Table 4.3: Results evaluation for using proposed filtering scheme only

Benchmarks	# FFs	TFs	# Orig. Inv.	# Redu. Inv.	% of Inv. Slec.	P1[16]	P3[17]
b01_gray_onehot	5/10	20	3044	450	14.8%	U	U
b02_gray_onehot	4/8	20	897	272	30.3%	U	U
b06_gray_onehot	9/13	20	3037	458	15.1%	Y	Y
b09_gray_onehot	28/30	20	92097	43510	47.2%	Y	Y
b13_gray_onehot	29/34	20	76016	29874	39.3%	N	N

[16] Sufficient to prove equivalence using initial potential invariants

[17] Sufficient to prove equivalence using invariants selected by Filtering Technique II

insufficient. We can see that for both b06 and b09 that can be proved by using all initial potential invariants, our filtering approach can also prove all of them. On the other hand, for other circuits that cannot be proved by using initial set of potential invariants, our approach can detect that by not being able to select sufficient invariants to pass the filter.

Also similar to Part I, Table 4.4 shows the performance comparison of proposed SEC framework with/without filter. The first column shows the names of the benchmarks. The second column shows the time taken by the filtering scheme that we applied. Columns 3 and 4 show the time for fix-point calculation of inductive invariants for initial invariants and filtered invariants, respectively. The fifth and the sixth column show the total time taken by the SEC framework without and with filter. Finally, Column 7 shows the speedup of the total run time for SEC framework with filter over without filter.

We first note that the time for filtering is generally a small fraction of the time needed for fix-point calculation. For instance, in b06_gray_onehot, the filter took only about 1 second, while the fix-point calculation for the inductive invariants took nearly 12 seconds. Also fix-

Table 4.4: Performance comparison of proposed SEC framework with/without Filtering Technique II

Benchmarks	T-filter(s)[18]	T-P1(s)[19]	T-P3(s)[20]	T-total1(s)[21]	T-total3(s)[22]	Speedup(x)
b01_gray_onehot	0.084	28.07	0	32.83	4.844	6.78
b02_gray_onehot	0.036	3.984	0	5.132	1.184	4.33
b06_gray_onehot	0.916	11.749	2.761	16.133	8.061	2.00
b09_gray_onehot	222.392	16939.418	8545.917	17028.996	8857.887	1.92
b13_gray_onehot	31.464	TO(36000)	0	TO(36000)	82.445	>436.60

[18] Time taken by filtering schemes

[12] Time taken by fix-point calculation of inductive invariants using initial pot. invariants

[13] Time taken by fix-point calculation of inductive invariants using filtered pot. invariants

[21] Total time taken by SEC framework without filter

[22] Total time taken by SEC framework with filter

point calculation of selected invariants which is only 15.1% of the original potential invariants took only 32.7% of the time — resulting a 2× speedup in terms of overall runtime. Note that the selected invariants are still sufficient to prove equivalence. The results for b09 for which the potential invariants were sufficient to prove equivalence follow a similar trend.

Next, consider the circuit b13_gray_onehot among those cases that equivalence could not be proved with the original set of potential invariants. After random simulation, 76016 potential invariants are obtained. If all of these 76016 potential invariants were true invariants, adding them to all time-frames of M' would make the formula $f_{M'}$ UNSAT. However, it's almost always the case that only a subset of the initial potential invariants are true inductive invariants. On the other hand, since our filter iteratively adds any filtered constraints to the first time-frame only, those false invariants will not remain true in subsequent time-frames.

As a result, it allows us to conclude that the potential invariants are insufficient. Fix-point calculation of these inductive invariants and applying all true invariants in attempting to solve the SEC instance causes a timeout (1 hour), while our Filtering Technique I took only 31.436 seconds to conclude that the set of potential invariants are insufficient to prove equivalence, saving the trouble to actually find the true inductive invariants and perform the SEC. A speedup of more than 400× was achieved in this case. The results are similar for other instances for which equivalence could not be proved.

4.6 Summary

We presented a second novel filtering technique for processing potential invariants that select useful invariants towards proving the equivalence property of the miter circuit. It's highly flexible and can be both used alone and combined with Filtering Technique I that we proposed in the previous chapter. Comparing Filtering Technique I, this filter is based on a model that simulates basic-induction-based fix-point calculation more exactly. Therefore as shown in Part I of experimental results, for the cases that initial potential invariants are sufficient but the invariants selected by Filtering Technique I are not, using this filter after using Filtering Model I can supply with a little more invariants and make the selected invariants sufficient; Also for cases that initial potential invariants are not sufficient but they can still pass Filtering Model I, put this filter behind Filter I can block the initial invariants and prove the insufficiency of them directly. Experimental results Part II shows the power of the proposed filtering scheme alone, without combining it with Filter I.

Chapter 5

Conclusion and Future Work

We presented two novel filtering techniques for processing potential invariants that select useful invariants towards proving the equivalence property of the miter circuit. Both techniques are highly flexible and can work on either two-node or multi-node, either among only flip-flops or among all internal signals. Both of them can be used alone or be combined with each other. Experimental results show the power of the proposed approaches: when the original set of potential invariants is able to prove equivalence, the proposed approaches can always select a smaller subset and still be able to prove equivalence; on the other hand, when the original set of potential invariant is insufficient to prove equivalence, the proposed filters can quickly prove this by being unable to select a sufficient set of invariants. Overall, because the filtering techniques always take much less time than the time taken to actually prove the invariants, the SEC framework can save a lot of time when embedded with our filtering technique(s) and can still give us the same results as the SEC framework without any filter.

Our future work can be characterized into two directions. The first direction is to accelerate the filtering techniques that we propose. Currently we have two many clauses in the filters

which is caused by the reasons I: we are using too many time-frames in the filters and II: for Filtering Model II, there are too many transferring clauses. Since the complexity of a SAT problem is exponential with the number of clauses in the CNF formula, adding too many clauses to the filters cause SAT solver takes too much time to solve, especially for larger circuits. The second direction of future work is to find a way to prove a fix-point directly on Filtering Model II. As we analyzed in Chapter 4, Filtering Model II essentially simulate the exact process of basic-induction-based assume-then-verify. Therefore if a fix-point can be proved in a functional time-frame k where all the invariants that are true on time-frame $k - a$ remain true on time-frame $k - b$, and the equivalence property can be proved on this functional time-frame, the truthfulness of equivalence can be justified directly without the need of basic-induction-based assume-then-verify to find out exactly which invariants are true, thus the SEC framework can be further accelerated.

Bibliography

[1] A. Mishchenko, M. L. Case, R. K. Brayton, and S. Jang, "Scalable and scalably-verifiable sequential synthesis," in *Proceedings of International Conference on Computer-Aided Design*, 2008.

[2] J. Baumgartner, H. Mony, V. Paruthi, R. Kanzelman, and G. Janssen, "Scalable sequential equivalence checking across arbitrary design transformations," in *Proceedings of International Conference on Computer Design*, pp. 259–266. 2006.

[3] J.-H. Jiang and R. Brayton, "Retiming and resynthesis: A complexity perspective," in *IEEE Transactions on Computer-Aided Design*, Vol. 25 (12), Dec. 2006, pp. 2674–2686.

[4] C.A.J. van Eijk, "Sequential Equivalence Checking Based on Structural Similarities," in *IEEE Transactions on Computer-Aided Design*, vol. 19, pp. 814–819, Jul. 2000.

[5] J.-H. Jiang and W.-L. Hung, "Inductive equivalence checking under retiming and resynthesis," in *Proceedings of IEEE International Conference on Computer-Aided Design*, November 2007, pp. 326–333.

[6] F. Lu and K.-T. Cheng, "Sequential equivalence checking based on K-th invariants and circuit SAT solving," in *Proceedings of IEEE High Level Design Validation and Test Workshop*, Nov. 2005, pp. 45–52.

[7] A. Rosenmann and Z. Hanna, "Alignability equivalence of synchronous sequential circuits," in *Proceedings of IEEE High Level Design Validation and Test Workshop*, 2002.

[8] D. Kaiss, M. Skaba, Z. Hanna, and Z. Khasidashvili, "Industrial strength SAT-based alignability algorithm for hardware equivalence verification," in *Proceedings of IEEE Formal Methods in Computer-Aided Design*, November 2007, pp. 20–26.

[9] Z. Khasidashvili and Z. Hanna, "SAT-based methods for sequential hardware equivalence verification without synchronization," in *Electronic Notes in Theoretical Computer Science* 89(4), 2003.

[10] M. Syal and M. S. Hsiao, "VERISEC: VERIfying Equivalence of SEquential Circuits using SAT," In *Proceedings of IEEE High Level Design Validation and Test Workshop.* pp. 52–59(2005)

[11] N. Goel, M. S. Hsiao, N. Ramakrishnan, and M. J. Zaki, "Mining complex boolean expressions for sequential equivalence checking," in *Proceedings of the IEEE Asian Test Symposium*, December 2010.

[12] L. Zhao, M. J. Zaki, and N. Ramakrishnan, "Blosom: a framework for mining arbitrary boolean expressions," in *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* New York, NY, USA: ACM, 2006, pp. 827–832.

[13] S. Ray, A. Mishchenko, R. Brayton, "Incremental Sequential Equivalence Checking and Subgraph Isomorphism," in *Proceedings of IEEE International Workshop on Logic & Synthesis*, 2009.

[14] 2006. [Online]. Available: http://www.eecs.berkeley.edu/ alanmi/abc/

[15] M. L. Case, A. M. Robert, and K. Brayton, "Inductively Finding a Reachable State Space Over-Approximation," in *Proceedings of International Workshop on Logic & Synthesis*, 2006.

[16] F. Lu and T. Cheng, "IChecker: An efficient checker for inductive invariants," in *Proceedings of IEEE High Level Design Validation and Test Workshop*, pp. 176–180. 2006.

[17] C. Pixley, "A theory and implementation of sequential hardware equivalence," in *IEEE Transactions on Computer Aided-Design*, vol. 11(12), December 1992, pp. 1469–1478.

[18] V. Singhal, C. Pixley, A. Aziz, and R. K. Brayton, "Theory of safe replacements for sequential circuits," in *IEEE Transactions on Computer-Aided Design*, vol. 20(2), February 2001, pp. 249–265.

[19] M. N. Mneimneh and K. A. Sakallah, "Principles of sequential-equivalence verification," in *IEEE Design & Test of Computers*, Vol. 22(3), pp. 248–257, 2005.

[20] S. Cook, "The complexity of theorem proving procedures," in *Proceedings of ACM symposium on Theory of computing,* pages 151–158, 1971.

[21] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: engineering an efficient sat solver," in *Proceedings of Design Automation Conference*, 2001, pp. 530?35.

[22] [Online]. Available: http://www.princeton.edu/ chaff/zchaff.html

[23] N. Een and N. S rensson, "An extensible sat-solver [ver 1.2]," 2003.

[24] [Online]. Available: http://minisat.se/

[25] J. P. Marques-Silva and K. A. Sakallah, "GRASP: A New Search Algorithm for Satisfiability," in *Proceedings of International Conference on Computer-Aided Design*, pp. 220-227, 1996.

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

[27] L. Fang and M. S. Hsiao, "Boosting SAT solver performance via a new hybrid approach," in *Journal of Electronic Testing: Theory and Applications,* vol. 24, no. 1–3, June, 2008, pp. 285–296

[28] W. Kunz and D. K. Pradhan, "Recursive learning: A new implication technique for efficient solutions to cad problems test, verification, and optimization," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 9, pp. 1149–1158, September 1994.

[29] J. K. Zhao, E. Rudnick, and J. Patel, "Static logic implication with application to redundancy identification," in *Proceedings of IEEE VLSI Test Symposium*, April 1997, pp. 288–293.

[29] M. Syal, R. Arora, and M. Hsiao, "Extended forward implications and dual recurrence relations to identify sequentially untestable faults," in *Proceedings of IEEE International Conference on Computer Design: VLSI in Computers and Processors*, October 2005, pp. 453–460.

[30] J. K. Zhao, J. A. Newquist, and J. H. Patel, "A graph traversal based framework for sequential logic implication with an application to C-Cycle redundancy identification," in *International Conference on VLSI Design*, p. 163, 2001.

[31] R. Arora, M. S. Hsiao, "Using global structural relationships of signals to accelerate SAT-based combinational equivalence checking," in *Journal of Universal Computer Science*. vol. 10, no. 12, pp. 1597–1628, 2004.

[32] E. Knuth, E. Donald, "The Art of Computer Programming", Volume 1: Fundamental Algorithms (3rd ed.). Addison-Wesley. ISBN 0-201-89683-4. (Section 1.2.1: Mathematical Induction, pp. 11–21.) 1997

[33] Kolmogorov, N. Andrey, Sergei V. Fomin, "Introductory Real Analysis". Silverman, R. A. (trans., ed.). New York: Dover. ISBN 0-486-61226-0. (Section 3.8: Transfinite induction, pp. 28–29.) 1975

[34] M. Sheeran, S. Singh, and G. St almarck, "Checking safety properties using induction and a sat-solver," in *Proceedings of International Conference on Formal Methods in Computer-Aided Design*. London, UK: Springer-Verlag, 2000, pp. 108–125.