SAT-BASED VERIFICATION FOR ANALOG AND MIXED-SIGNAL CIRCUITS

A Thesis

by

YUE DENG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2012

Major Subject: Computer Engineering

SAT-BASED VERIFICATION FOR ANALOG AND MIXED-SIGNAL CIRCUITS

A Thesis

by

YUE DENG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Peng Li
Committee Members,	Jose Silva-Martinez
	Duncan M. Walker
Head of Department,	Costas N. Georghiades

May 2012

Major Subject: Computer Engineering

ABSTRACT

SAT-based Verification for Analog and Mixed-signal Circuits. (May 2012)

Yue Deng, B.S., Xi'an Jiaotong University

Chair of Advisory Committee: Dr. Peng Li

The wide application of analog and mixed-signal (AMS) designs makes the verification of AMS circuits an important task. However, verification of AMS circuits remains as a significant challenge even though verification techniques for digital circuits design have been successfully applied in the semiconductor industry.

In this thesis, we propose two techniques for AMS verification targeting DC and transient verifications, respectively. The proposed techniques leverage a combination of circuit modeling, satisfiability (SAT) and circuit simulation techniques.

For DC verification, we first build bounded device models for transistors. The bounded models are conservative approximations to the accurate BSIM3/4 models. Then we formulate a circuit verification problem by gathering the circuit's KCL/KVL equations and the I-V characteristics which are constrained by the bounded models. A nonlinear SAT solver is then recursively applied to the problem formula to locate a candidate region which is guaranteed to enclose the actual DC equilibrium of the original circuit. In the end, a refinement technique is applied to reduce the size of candidate region to a desired resolution. To demonstrate the application of the proposed DC verification technique, we apply it to locate the DC equilibrium points for a set of ring oscillators. The experimental results show that the proposed DC verification technique is efficient in terms of runtime.

For transient verification, we perform reachability analysis to verify the dynamic property of a circuit. Our method combines circuit simulation SAT to take advantage of the efficiency of simulation and the soundness of SAT. The novelty of the proposed transient verification lies in the fact that a significant part of the reachable state space is discovered

via fast simulation while the full coverage of the reachable state space is guaranteed by the invoking of a few SAT runs. Furthermore, a box merging algorithm is presented to efficiently represent the reachable state space using grid boxes. The proposed technique is used to verify the startup condition of a tunnel diode oscillator and the phase-locking of a phase-locked loop (PLL). The experimental results demonstrate that the proposed transient verification technique can perform reachability analysis for reasonable complex circuits over a great number of time steps.

To my parents

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1117660, and the Semiconductor Research Corporation and Texas Analog Center of Excellence under Contract 2009-TJ-1836.

This research work would not have been possible without the patient guidance of my advisor, Dr. Peng Li. I appreciate his insightful advices and warm encouragement all through this research work.

Also, I would like to thank other members in the committee, Dr. Jose Silva-Martinez and Dr. Duncan M. Walker for their suggestion on my thesis.

Furthermore, special thanks goes to my colleague, Leyi Yin for his kindly helps on this work.

Last but not the least, thanks to my parents for their support and love.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

A. Background

An analog and mixed-signal circuit (AMS) is an integrated circuit which contains both digital and analog circuits on a single chip. AMS design is crucial for embedded system designs and microprocessors. It enables the embedded system to receive analog signals from the real world and to process the converted digital signals using digital circuits. Besides, AMS circuits' functionalities also include timing signal generation and biasing [1], etc. Due to the wide application of embedded systems and microprocessors, AMS circuits can be found in devices from consumer electronic products like smart phones to specific controllers in automobiles and airplanes. Clearly, it is of great importance to make sure that the AMS circuits meet the design specifications.

The process of verifying whether a design satisfies its specifications is referred to as *verification*. While verification techniques of digital circuit designs have been successfully applied in industry for decades, verification techniques for AMS designs are still far from maturity considering the size and the complexity of the problems they can handle. Part of the reasons why verification for AMS designs is much difficult is that analog circuits operate over a continuous state space and usually involve far more complex analog characteristics. Moreover, the behaviors of AMS circuits involve the interaction between the discrete domain and continuous domain.

In industry, verification is usually fulfilled by simulation techniques. Simulation techniques generate a number of simulation traces based on a model of the target AMS design. Each simulation trace is checked to see whether it violates the specifications or not. If vi-

---

The journal model is *IEEE Transactions on Automatic Control.*

olation happens for any of the traces, the design fails and needs to be rectified. Otherwise, the design is treated as correct. The major problem of the simulation method is that limited number of traces are theoretically unable to provide full coverage for the state space of the circuits' behaviors. Due to this inherent incompleteness nature, no guarantee but only a certain level of confidence can be obtained on the correctness of the designs using simulation techniques. Therefore, a more rigorous verification method for AMS design is strongly needed.

B. Formal verification for AMS circuits

In recent years, a lot of research works has been performed on the development of formal verification techniques for AMS designs. Formal verification techniques refer to the methods which mathematically specify and verify the correctness of a system against certain specifications. Unlike the conventional verification technique such as simulation, the entire range of input and parameter variations is implicitly considered in the formal verification. Thus if a formal verification method proves that a design meets its specifications, this diagnosis holds for all the input values and parameter values.

In [1] on formal verification techniques for AMS circuit, the techniques are categorized to two different fashions: state space exploration methods and theorem proving methods.

The state space exploration methods can be further divided to two groups: equivalence checking and model checking.

Equivalence checking compares the outputs of two different models for the same circuit design over a certain range of input and decides wether these two models are equivalent in behavior by measuring the difference between the outputs. The models being compared can be at the same level as well as at different levels, e.g., netlist v.s. netlist, netlist v.s.

behavioral, behavioral v.s. macromodel, etc. One major motivation of equivalence checking is to make difficult verification problem easier. For a reasonably large and complex AMS circuit, the task to verify the correctness of the circuit on transistor level can be very time-consuming. A solution to this problem is to build simplified models for the circuit by extracting only characteristics of interest. The simplified models are then used in the verification process to represent the behaviors of the circuit. However, before the extracted simplified models are used, they must be validated that they're indeed equivalent to the transistor level model in some sense.

A semi-formal equivalence checking methodology for large AMS circuits is proposed in [2]. This work clearly defines the mapping between the behavioral domain and electrical domain and formulates the verification problem as an optimization problem.

Two extensions are proposed for an equivalence checking method for analog circuits with strong nonlinear characteristics in [3]. The first extension introduced is new eigenvalue mapping methods built upon observability and structural information. The second extension is reachability analysis which prevents the occurrence of false negative by constraining the search space.

Model checking refers to the group of techniques that verify wether a model meet its specifications or not. In model checking, both the model of the AMS circuit to be verified and the property to be checked must be described in some sort of mathematical formula. Usually, model checking is used to check the dynamic behavior of an AMS circuit of which the model is a kind of transition system. The state space of the model of design is explored by the model checking techniques to decide whether a given property is satisfied or not.

In [4, 5], state space exploration is performed by converting continuous dynamics to approximated discrete model. State space exploration can also be achieved by reachability analysis techniques originated from the research on verification of hybrid system [6–8]. In these reachability analysis techniques, the state space is over-approximated using geo-

metrical representation such as polytopes or zonotopes. A reachability analyisis technique targeted for phase-locked loops (PLLs) is developed recently [9]. One of the key ideas in the approach is to over-approximate the switching times of the charge pump and perform reachability analysis using linear continuous models with uncertain parameters.

Theorem proving methods, also called proof-based methods, construct mathematical proof that a model satisfies its specifications using a certain set of inference rules. Theorem proving methods are powerful while expertise-intensive and time-consuming.

Formal verification method using an automatic theorem prover, MetiTarski, is introduced in [10]. In this work, a closed form of the behavioral model of the circuit is generated and combined with the properties of interest which are expressed by a set of inequalities. The combined constraints are then proved using MetiTarski.

There are other sort of verification techniques which are not included in the survey [1] mentioned earlier. Some verification technique is specifically developed to verify properties for a certain set of circuits which share some common features. For example, by taking advantage of the monotonic property of MOSFET devices, a specific technique is developed to verify the start-up conditions for ring oscillators via finding all its DC solutions [11].

Satisfiability (SAT) based verification has become an active topic in the CAD community along with the dramatic improvement of SAT solver technology. SAT solver solves the decision problem of whether a given formula can be evaluated to true. The exhaustiveness feature of the underlying search algorithm of SAT solvers makes them natural tools for formal verification. Recent advances in SAT-based formal verification of digital hardware designs can be found in the survey [12].

fSPICE, a formal verification tool, is described in the work [13]. *fSPICE* first captures the nonlinear behaviors of transistors in the circuit by conservative interval-based representation. The constraints for the circuit are then formulated and gathered like in a SPICE style simulation problem. In the end, *fSPICE* finds the solutions using an exhaustive search

scheme with the help of a linear SAT solver. Abstract refinement techniques is also introduced in the work to improve the runtime efficiency of the tool.

A formal approach is developed in [14] to find all the DC operating points, if any, for analog circuits. This work develops a MATLAB circuit modeling system to orchestrate a set of public tools including a SAT solver, HySAT [15, 16]. The developers of HySAT names the successor of HySAT by iSAT [17], which is the SAT solver adopted by this presented thesis. Different from the SAT solver employed in [13], which handles only linear constraints, iSAT can handle both linear and nonlinear constraints through a tight integration of DPLL-style SAT solving framework with interval-based arithmetic constraint propagation technique. This feature enables us to conveniently represent the nonlinear behavior of the circuit using nonlinear functions instead of to approximate the nonlinear behavior using various linearization methods. In order to emphasize the benefit from the nonlinear feature, we refer to the SAT solver, iSAT, and its underlying algorithm as *nonlinear SAT* (NLN-SAT). Chapter II will provide a detail introduction to the NLN-SAT technique.

This work is largely motivated by the initial success of *fSPICE* [13] on SAT-based formal analog verification as well as the recent advancements of iSAT [15, 17] on nonlinear SAT solving techniques.

C. Overview of the presented work

In this thesis, we developed a set of two verification methods for AMS circuits with the help of circuit macro-modeling methods and SAT solving technology. The first method, *DC* verification technique, determines the existence and location of all DC equilibrium points. The second method, a transient verification method, verifies the *dynamic* properties of AMS circuits via reachability analysis which leverages the efficiency of circuit simulation and the soundness of SAT at the same time. Though the two methods are different in many aspects,

they have the same flow as shown in Figure 1.

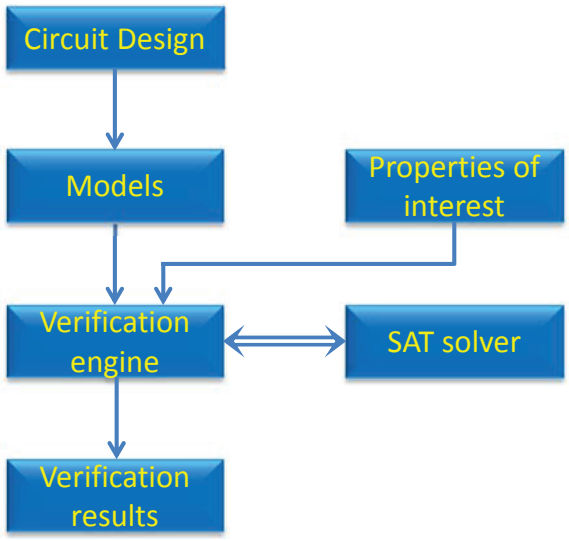


Fig. 1. General flow of SAT-based verification

The first method, DC verification technique, inspects circuits' behaviors at transistor level. However, modern transistor models such as BISM3/4 have a large number (i.e. thousands) of complex nonlinear equations, which cannot be practically handled by the NLN-SAT technique [15, 17]. To address this challenge, we propose to introduce an intermediate device modeling layer wherein conservative level-one like models are extracted to bound the exact device characteristics. Then, NLN-SAT is efficiently applied to the circuit whose devices are represented using simple bound models. Finally, the approximate solutions computed by the previous step is refined locally using accurate BSIM model data. The approach is applied to locate all DC equilibrium points for ring oscillators.

The second method, reachability analysis technique, checks the dynamic behaviors of AMS circuits by iteratively calculating the next reachable space starting from a given initial range. For AMS circuits with complex nonlinear behaviors, it is envisioned that modeling abstraction is required to render the reachability analysis practical. Techniques such as [2,3]

may be used to build conservative behavioral models to account for factors such as modeling error and parameter variations for a large AMS circuit. NLN-SAT can then be applied to the behaviorial models to yield conservative check of dynamic design properties. With the help of modeling abstraction, acceleration techniques are still desired. To this end, we propose a *simulation-assisted SAT* approach that simultaneously exploit the efficiency of simulation and the conservativeness of SAT technique. Simulation-assisted SAT approach can dramatically reduce the number of calld NLN-SAT calls, leading to large verification speedups. Moveover, in order to flexibly model arbitrary nonlinear dynamics and the resulting reachable state space, the reachable state space is represented using a collection of fixed-grid cubes. In this thesis, we demonstrate the application of this approach with two examples: 1) verifying start-up condition for a tunnel diode oscillator and 2) verifying locking time for a phased lock loop (PLL).

D. Organization of this thesis

The organization of the rest of this thesis goes as follows.

Chapter II offers an overview of SAT problems and the NLN-SAT technique. The cornerstone for most modern SAT solvers, DPLL algorithm, is described. Then the underlying algorithm for the NLN-SAT technique is provided.

Chapter III introduces our proposed verification technique for DC analysis. After explaining the importance of locating DC equilibrium points, the modeling method and problem formulation are described. The algorithm of our proposed DC verification technique is then explained in detail. The technique is demonstrated in the end by an experiment on a set of ring oscillators.

Chapter IV is devoted to our proposed transient verification method for dynamical properties checking. The motivation of why reachability analysis technique is selected

to fulfill transient verification is discussed at first. The fixed-grid-boxes representation of reachable space is explained along with its benefits and limitations. A subroutine which merges the reachable space is introduced to ease the limitations while keeping the benefits. After that, detail description of simulation-assisted SAT for reachability analysis is provided. In the end, two important experiments are used to demonstrate the application of our transient verification method. First, we apply our method to verify the start-up condition for a tunnel diode oscillator. Second, a more challenging experiment is performed in which the locking time of a charge-pump PLL is investigated.

Chapter V presents a brief summary of the research work in the end of this thesis.

CHAPTER II

INTRODUCTION TO SAT

Considering this work is largely empowered by the SAT solver technology, it is decided that the first main chapter (i.e. this chapter) provides an overview about the definition of SAT problems, and the underlying principle of SAT solvers.

A. SAT problems

A satisfiability (SAT) problem is a decision problem: given a formula ϕ, answering the question that wether there exists a variable assignment which can lead the formula ϕ to be true. If such a variable assignment exists, we call it a *satisfiable assignment* which makes the formula ϕ *satisfiable*. Otherwise, the formula ϕ is *unsatifiable*. A SAT problem is a NP-complete problem.

B. NLN-SAT technique

In this work, the SAT solver we used is *iSAT* [15, 17]. As mentioned in Chapter I, we refer to the solver as well as its underlying technique as NLN-SAT in this thesis.

NLN-SAT is a tight integration of the *Davis-Putnam-Logemann-Loveland* (DPLL) algorithm and the *interval constraint propagation* (ICP) technique.

DPLL algorithm, a complete backtracking-based searching algorithm, is the fundamental framework for most modern SAT solvers. The DPLL algorithm finds a satisfiable assignment ρ, if exists, to a given *boolean* formula ϕ in conjunctive normal form (CNF). A CNF formula is a collection of clauses connected by boolean *AND* operator while each clause is a set of literals connected by boolean *OR* operator. A literal is either a boolean variable or negation of a boolean variable. The algorithm flow of DPLL is shown in Algo-

rithm 1.

The first step is to preprocess the input boolean formula. After the formula passes the preprocess step, the algorithm arrives at the next step, *decision* step. In the decision step, the algorithm will first find if there exists any variable without assignment. If no such variable exists, then an satisfiable assignment has been found. Otherwise, the algorithm will select an unassigned variable and randomly assign it with a truth value, either *TRUE* or *FALSE*. Decision step is followed by *deduction* step. In the deduction step, the algorithm locates each *unit clause* and makes an assignment to let the unit clause to be true. If there is no unit clause left, then the algorithm will go back to the decision step. A unit clause is a clause having all but one of its literals assigned with values and the existing assignments make all the literals except the unassigned one false. It is clear that in order to make a unit clause to be true, which is a necessary condition making the entire formula ϕ to be satisfiable, a correct and unique assignment is needed to for the unassigned literal in the unit clause. After the implied assignment is made for the unit clause, the formula will be evaluated. If the evaluation result is unsatisfiable, e.g., a variable is assigned to be true and false at the same time, then the source of decision leading to the conflict will be located. If the union of conflict sources ever occurred covers the entire state space, the formula is found to be unsatisfiable. Otherwise, backtracking process will undo all the decision and deduction stemming from the conflict source. Besides, a conflict clause which is a negation of the conflict source will be added to the problem formula ϕ to prevent subsequent search from ending at the same conflict again. After the conflict clause is added to the problem formula, the algorithm will go back to decision step.

Based on interval arithmetic, interval constraint propagation (ICP) locates the intervals containing all solutions to the problem formula. For a given constraint ϕ over the real domain with the entire search space denoted by S, ICP technique can return a space $\{\rho|\rho \subseteq S\}$ and ρ contains all solutions to ϕ. A simple example will be sufficient to illustrate the

Algorithm 1 DPLL algorithm, input: boolean formula ϕ

```
1: if preprocess(φ) = UNSAT then
2:     return UNSAT;
3: end if
4: S = ∅; //conflict-source set
5: while unassigned variable exists do
6:     pick an unassigned variable and assign it a value; //decision()
7:     //deduction()
8:     while unit clause exists do
9:         pick a unit clause;
10:        make implied assignment for the unit clause;
11:        if evaluation(φ) = UNSAT then
12:            find the conflict-source s;
13:            S = S ∪ s;
14:            if S = the entire state space then
15:                return UNSAT;
16:            else
17:                undo all decision and deduction after s; //backtrack()
18:                φ = φ ∧ s̄;
19:                break;
20:            end if
21:        end if
22:    end while
23: end while
24: return SAT with satisfiable assignment
```

basic idea of ICP. The problem constraint is $a - b = c$ where $a \in [1, 2]$, $b \in [1, 3]$, and $c \in [-2, 2]$. Based on the transformed constraint $c = a - b$ and the interval of a and b, we can calculate the interval of the left-hand side of the equation, which is $[-2, 1]$. With this interval, the original interval $[-2, 2]$ of c can be contracted. We repeat this calculation for all of the variables until none of the intervals can be further contracted.

The NLN-SAT technique incorporates the ICP technique into the DPLL algorithm to tackle the SAT problem for formula over the real domain. In such context, the definition of a literal must be adjusted: each literal is an arithmetic constraint or a negation of an arithmetic constraint with variables over the real domain. The basic structure of the NLN-SAT algorithm is shown in Algorithm 2.

NLN-SAT takes the general structure of the standard DPLL algorithm as framework while customizing the decision and deduction processes accordingly to handle real variables. In the decision process, the NLN-SAT is no longer looking for unassigned boolean variables but real variables which have an interval with a length greater than a pre-defined threshold δ. If such variables exists, the NLN-SAT will select one of them and split its interval into two subintervals with the same length. The algorithm will then temporarily discard one of the subintervals and contract the interval of the selected variable to the other subinterval. After this, the ICP technique is applied to the formula ϕ. If the ICP routine terminates with no conflict, then the algorithm will jump back to the decision step. If conflict happens, e.g., the interval of a variable is contracted to be empty, the source of decision which leads to conflict will be located as in the standard DPLL algorithm. When the union of conflict sources covers the entire search space, the algorithm returns UNSAT. Otherwise, a backtrack routine will be called and the algorithm will go back to the decision process after adding a conflict clause to the formula ϕ.

It is worth to explain the termination conditions of the NLN-SAT algorithm. The NLN-SAT algorithm is essentially a branch-and-prune process. If a wrong decision is

Algorithm 2 NLN-SAT algorithm, input: formula ϕ over real and/or boolean domain

if preprocess(ϕ) = UNSAT **then**
  return UNSAT;
end if
$S = \varnothing$; //conflict-source set
while variable with interval length greater than δ exists **do**
  //decision()
  pick such a variable and divide its interval to two pieces with same length;
  randomly select one of the subintervals as new interval for that variable;
  //deduction()
  if ICP(ϕ) leads to UNSAT **then**
    find the conflict-source s;
    $S = S \cup s$;
    if $S =$ the entire state space **then**
      return UNSAT;
    else
      undo all decision and deduction after s; //backtrack()
      $\phi = \phi \wedge \bar{s}$;
    end if
  end if
end while
return INCONCLUSIVE with solution

made, e.g., a subinterval containing no solution is selected, the deduction process will recognize it and prune the corresponding subinterval by adding a conflict clause to the problem formula. If the entire search space is pruned, there is no satisifable assignment. On the other side, if the formula is satisfiable, the branch-and-prune process can repeat infinitely because no matter how small the remaining space is left there is still half of this remaining space with no solution. Therefore, in order to enforce a termination, the NLN-SAT algorithm needs an appropriate termination mechanism: when the interval length of a variable is smaller than a certain threshold δ this variable will no longer be considered in the decision process. Clearly, when no variable has an interval with a length greater than the threshold δ, the algorithm terminates and the current intervals of each variable are returned as result. The answer given by the solver is not an exact point solution but a space containing the point solution. In this work, we still refer to the resulted space as solution because with a small enough termination threshold δ the resulted space can be safely treated as a point.

Last and most importantly, the NLN-SAT technique we use in this work can provide *guarantee on unsatisfiability* [15] which means that if the solver returns an UNSAT result the problem formula indeed has no solution. As discussed later in the Chapter III and the IV, this feature is made use of by this work to guarantee the *conservativeness* of the solution found by our proposed methods.

CHAPTER III

DC VERIFICATION

A. Context

In the DC analysis of any circuit, a fundamental job is to identify the *DC operating points*. A DC operating point is a steady state of the circuit with constant input, i.e., the value of input does not change with time. DC operating point is very important in circuit analysis. Transient simulation takes the DC operating point as initial state of the circuit. Small signal analysis such AC analysis linearizes the circuit on the DC operating point to approximate the nonlinear circuit behavior. By sweeping the input to a circuit and finding its DC operating points, the transfer relations of the circuit can be obtained. Moreover, DC operating points also relates to circuits' properties of interest. For example, a Schmitt trigger circuit has multiple DC operating points, each of which represents a distinct state of the circuit. On the other side, a ring oscillator must have no operating point.

Since the DC operating points are so important, we provides its definition in a mathematical context. For any given circuit, we represent its states of voltages and currents by a vector $\vec{x}(t)$, and the input signal to the circuit by function $\vec{u}(t)$. Then the circuit's behavior can be described by a differential equation as follow

$$\frac{\mathrm{d}}{\mathrm{d}t}\vec{x}(t) = f(\vec{x}(t), \vec{u}(t)) \tag{3.1}$$

Assume $\vec{x_s}(t)$ is a solution to the Equation 3.1 when the input is constant, i.e., $\vec{u}(t) \equiv \vec{u}(0)$. Then $\vec{x_s}(\infty)$ is called a *DC equilibrium point* of the circuit. A circuit with DC equilibrium points does not necessarily have DC operating point because only *stable* equilibrium point is operating point (a circuit with constant input will settle only to stable equilibrium point).

A stable DC equilibrium point is defined as follow. For any $\vec{x_a}(t)$ which satisfies

$$\begin{cases} \dfrac{\mathrm{d}}{\mathrm{d}t}\vec{x_a}(t) = f(\vec{x_a}(t), \vec{u}(t)), \quad \vec{u}(t) \equiv \vec{u}(0) \\ \|\vec{x_a}(0) - \vec{x_s}(\infty)\| < \varepsilon, \qquad\qquad\qquad \varepsilon > 0 \end{cases}, \tag{3.2}$$

if

$$\lim_{t \to +\infty} \vec{x_a}(t) = \vec{x_s}(\infty) \tag{3.3}$$

then $\vec{x_s}(\infty)$ is a stable equilibrium. A intuitive version of the definition is as follow: if all traces starting from the neighborhood of $\vec{x_s}(\infty)$ will eventually converge to $\vec{x_s}(\infty)$, then $\vec{x_s}(\infty)$ is a stable DC equilibrium point.

The above definitions of equilibrium and stability show that in order to identify a DC operating point two steps must be performed. The first step is to locate the equilibrium points by solving the constraints of the circuit. The second step is to determine the stability for each located equilibrium point. Since this presented work focuses on the application of SAT solver in verification methodology, we implements only the first step: finding all DC equilibria. A implementation of the second step can be found in the paper [14].

B. Bounded transistor model

In order to formulate the constraint (Equation 3.1) for the circuit, we should at first decide the models we use to capture the behavior of the devices. Accurate transistor-level models such as BSIM3 [18], which is among the standard models in the semiconductor industry, are too complex to be practically handled by the NLN-SAT solver. To address this issue, we introduce an approximate yet conservative level-one style *bounded* device model which is in the following form

$$\begin{cases} I_{ds} & \geq & Lowerbound(V_{gs}, V_{ds}, V_{sb}) \\ I_{ds} & \leq & Upperbound(V_{gs}, V_{ds}, V_{sb}) \end{cases}. \tag{3.4}$$

The bounded device model (Equations 3.4) is much simpler than the BSIM3/4 models but it's guaranteed to bound the exact I-V characteristics of the devices. For any assignment of value to (V_{gs}, V_{ds}, V_{sb}), the corresponding current value I_{ds} calculated based upon BSIM3/4 model is bounded by *LowerBound*(V_{gs}, V_{ds}, V_{sb}) and *UpperBound*(V_{gs}, V_{ds}, V_{sb}).

The bounded model is built through curve fitting based on BSIM3/4 simulation data. The level one model [19] is modified to serve as fitting template. Specifically, we replace the formula of level one model in the cutoff region with the following equation $I_{ds} = k_1 e^{k_2 V_{gs}}$ to capture the behavior in subthreshold region more accurately. Moreover, although in this work we built bounded model for each transistor with fixed device parameters, the modeling method can be applied straightforwardly to take process variations into account. For example, if we want to model the effect of the *gate width* on the performance of the transistors, we only need to add the gate width W to the model (Equations 3.4) as a new independent variables, i.e.,

$$\begin{cases} I_{ds} & \geq & Lowerbound(V_{gs}, V_{ds}, V_{sb}, W) \\ I_{ds} & \leq & Upperbound(V_{gs}, V_{ds}, V_{sb}, W) \end{cases} . \tag{3.5}$$

Specifically, take the modeling of the upper bound for example. The fitting template of *UpperBound*(V_{gs}, V_{ds}, W) is as follow

$$\begin{cases} Wk_1 e^{k_2 V_{gs}}, & V_{gs} \leq V_{th} \\ Wk_3(1+\lambda V_{ds})(V_{gs} - V_{th} - \dfrac{V_{ds}}{2})V_{ds} + b, & V_{gs} > V_{th} \wedge V_{ds} < V_{gs} - V_{th} \\ W\dfrac{k_3}{2}(1+\lambda V_{ds})(V_{gs} - V_{th})^2 + b, & V_{gs} > V_{th} \wedge V_{ds} \geq V_{gs} - V_{th} \end{cases} . \tag{3.6}$$

For simplicity, we ignore V_{sb} in this example. In Equation 3.6, W, V_{gs} and V_{ds} are independent variables while λ, k_1, k_2, k_3, and b are optimization variables. To obtain the

upper bound model, an optimization problem is formulated as follow

$$\begin{aligned} min \quad & UpperBound(V_{gs}, V_{ds}, V_{sb}, W) - BSIM3(V_{gs}, V_{ds}, V_{sb}, W) \\ & UpperBound(V_{gs}, V_{ds}, V_{sb}, W) \geq BSIM3(V_{gs}, V_{ds}, V_{sb}, W) \end{aligned} \tag{3.7}$$

After an optimization tool is applied to solve Equation 3.7, the modeling of the upper bound is complete.

As shown by Equation 3.4, we only model the I-V characteristics of the devices in this work (*I* stands for the drain current and *V* stands for the gate voltages). However, the above discussion about how process variations can be included in the model shows that other effects of the devices like the nonlinear Q-V characteristics (i.e. the charge voltage relations) can be handled in a similar way.

C. Problem formulation

After choosing the bounded device model to represent the behavior of devices, we can formulate the constraint 3.1 as follows. Given a circuit *ckt*, similar to SPICE simulation problem, we first gather its Kirchhoff's current law (KCL) equations and Kirchhoff's voltage law (KVL) equations. In this thesis, we simply refer to the KCL and KVL equations as $KCL(\vec{I})$ and $KVL(\vec{V})$ where the vectors $\vec{I}$ and $\vec{V}$ represent the current and the voltage variables of the circuit respectively. For each transistor in the circuit, we use the bounded model to represent its I-V characteristics. All these constraints, together, conservatively represent the behavior of the circuit. We denote this set of constraints by problem formulae ϕ, which is as follow

$$\begin{cases} KCL(\vec{I}) \\ KVL(\vec{V}) \\ \vec{I} <= UpperBound(\vec{V}) \\ \vec{I} >= LowerBound(\vec{V}) \end{cases} \tag{3.8}$$

Note that the solution to ϕ is *not* a solution to *ckt* because ϕ is not equivalent to but a conservative approximation of *ckt*. However, the solution to ϕ is guaranteed to bound the solution to *ckt*. With this property, we first apply the NLN-SAT solver to find the solution to ϕ and then refine it to desired resolution using the technique introduced in [13]. Assume that the circuit we're studying is a trivial circuit in which there're only one current variable and one voltage variable to be determined. In this trivial case, the vectors $\vec{I}$ and $\vec{V}$ transform to two scalar variables I and V. For this trivial circuit, the transformation from the actual circuit problem *ckt* to its conservative approximation ϕ is given by the Figure 2.

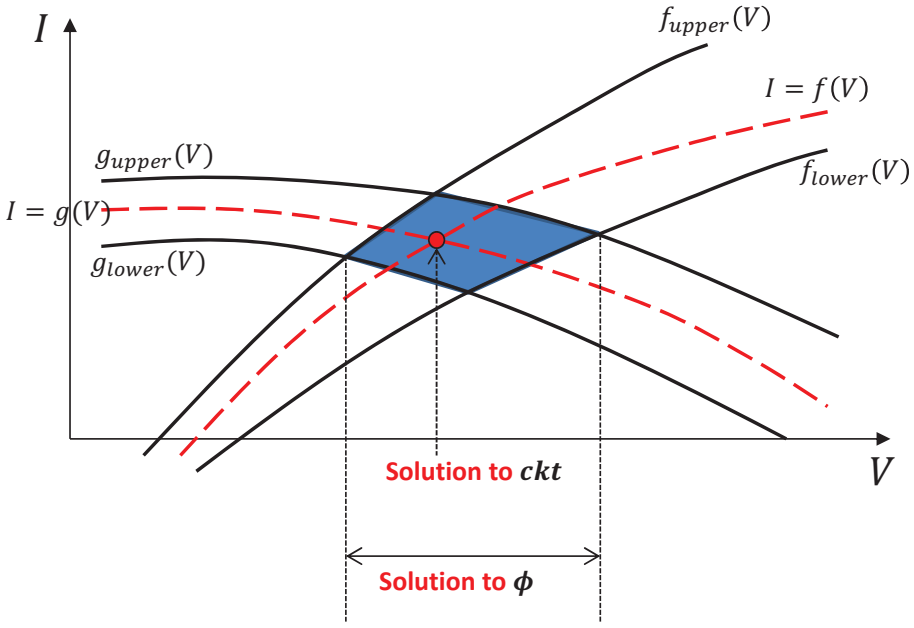


Fig. 2. Formulate DC problem with bounded models

As shown by the Figure 2, the trivial circuit *ckt* is constrained by the following equations

$$\begin{cases} I = f(V) \\ I = g(V) \end{cases}, \tag{3.9}$$

where the solution to *ckt* is represented by the intersection *point* of the two curves, which

correspond to $I = f(V)$ and $I = g(V)$ respectively.

With the bounded model, we approximate the circuit *ckt* by the formula ϕ via the following inequalities

$$\begin{cases} I <= f_{upper}(V) \\ I >= f_{lower}(V) \\ I <= g_{upper}(V) \\ I >= g_{lower}(V) \end{cases}, \tag{3.10}$$

where the solution is no longer a point but a region confined by the bounded model curves: $f_{upper}(V)$, $f_{lower}(V)$, $g_{upper}(V)$, and $g_{lower}(V)$. In Figure 2, the solution to ϕ is marked by blue color.

D. DC verification algorithm

Our DC verification method is a two-layer approach. In the first layer, we use a NLN-SAT solver to find, if any, all the solutions to the problem formulae ϕ. Still take the Figure 2 to illustrate how the first step goes. Of the two variables shown in Figure 2, it is clearly that the voltage variable V is the only independent variable. Therefore, we only need to determine the interval of variable V in which its value can make the formula ϕ to be true. This interval of interest is marked as *solution to* ϕ in Figure 2. For simplicity, we refer to the interval of interest as the solution interval in this section.

Here is the procedure (Algorithm 3) how the solution interval is determined.

First, we feed the formula ϕ to the NLN-SAT solver. Since any value within the interval is a solution to ϕ, the solver will return a value x_0 of V which can be anywhere within the solution interval. Then we construct an interval $[x_0 - \Delta, x_0 + \Delta]$ using the value x_0 as the mid-point. Next, We add a guidance constraint $V \notin [x_0 - \Delta, x_0 + \Delta]$ to formula ϕ. The augmented formula ϕ is then fed to the solver again. Clearly, the guidance constraint will force the solver to find solution only beyond the interval $[x_0 - \Delta, x_0 + \Delta]$. If a new

Algorithm 3 DC verification algorithm

construct problem formulae ϕ;
$S = \varnothing$; //solution set
call NLN-SAT solver;
while $solution \neq$ UNSAT **do**
    construct interval $[solution - \Delta, solution + \Delta]$
    $S = S \cup [solution - \Delta, solution + \Delta]$;
    $\phi = \phi \wedge (i,v) \notin [solution - \Delta, solution + \Delta]$;
    call NLN-SAT solver;
end while
if $S \neq \varnothing$ **then**
    refine S;
end if
return S

value x_1 is returned by the solver, we repeat the construction of guidance interval using x_1 as mid-point. Similarly the guidance interval is added to ϕ before the solver is calld again. On the other side, if the solver finds that there is no solution to the given constraints, we can be sure that the union of the intervals like $[x_0 - \Delta, x_0 + \Delta]$ is a super-set of the solution interval due to the guarantee on unsatisfiability from the solver. In this thesis, we refer to the union of the intervals like $[x_0 - \Delta, x_0 + \Delta]$ as *candidate region*. Δ is an experimental parameter which trades off the runtime with the over-approximation. With a big Δ, we can quickly determine a candidate region with a few calls to the solver. However, the size of the candidate region will be relatively large. With a small Δ, more calls are needed to determine a candidate region while the size of candidate region is relatively small. In our experiment, we choose a relatively large size to achieve faster runtime.

In the second layer, we apply the refinement technique introduced in [13] to reduce the size of candidate region to desired resolution. Since both the search methods in the first and second layer are conservative, we can be sure that there is definitely no solution outside the solution set S and therefore S is guaranteed to bound all DC solutions.

E. Experimental results

We apply our two-layer DC verification method to locate all DC equilibrium points for a set of ring oscillators[1]. In order to verify that the oscillator won't be trapped in a steady state, stability check is needed for each located equilibrium point. As mentioned before, this work focuses on only the first job: the identification of equilibrium points.

Fig. 3 shows the schematic of a three-stage ring oscillator. For simplicity, all PMOS transistors have the same size and so do all NMOS transistors. Besides, the carrier mobility of the NMOS transistors are triple as great as the carrier mobility of the PMOS transistors.

[1]Experiment environment: 4-core Intel CPU Q9450, 8 GB memory, Ubuntu.

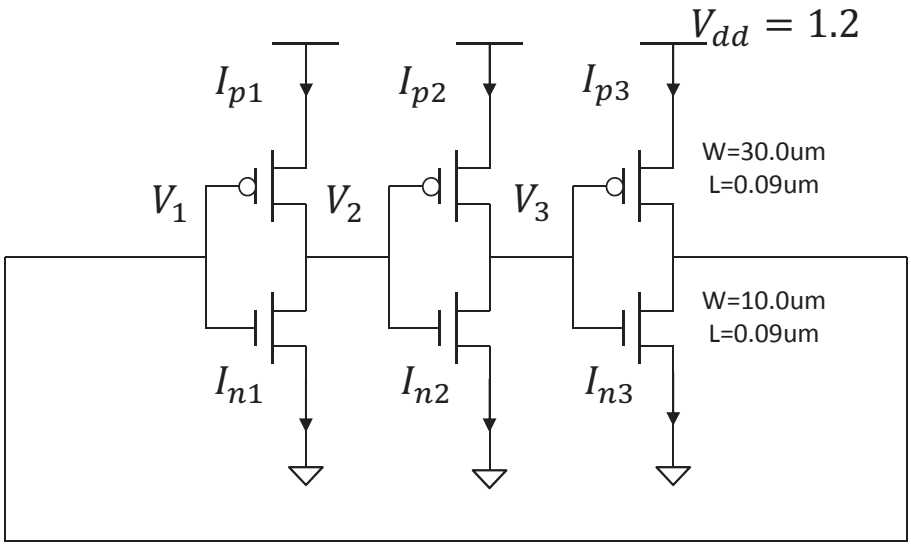


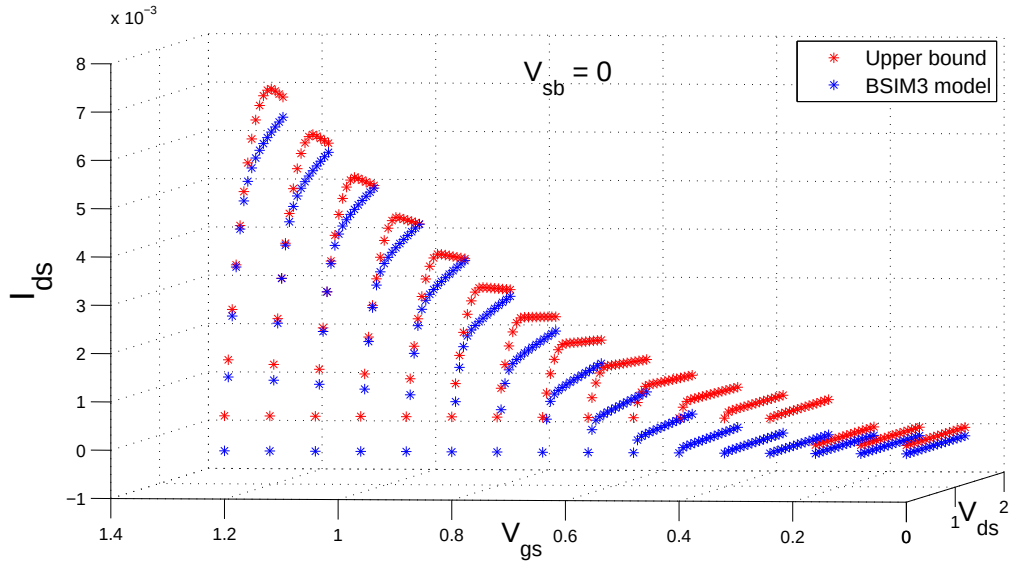
Fig. 3. Schematic of a 3-stage ring oscillator

For each transistor, we build its bounded device model by fitting its BSIM3 DC simulation results. Take the upper bound $I_{ds} = UpperBound(V_{gs}, V_{ds}, V_{sb})$ for example. For any fixed value val of V_{sb}, the upper bound uniquely corresponds to a surface in the 3-dimensional space (V_{gs}, V_{ds}, I_{ds}). In our experiment, for both the NMOS and PMOS transistors, $V_{sb} = 0$. The upper bound and the lower bound of the NMOS transistors in this experiment are shown in Figure 4.

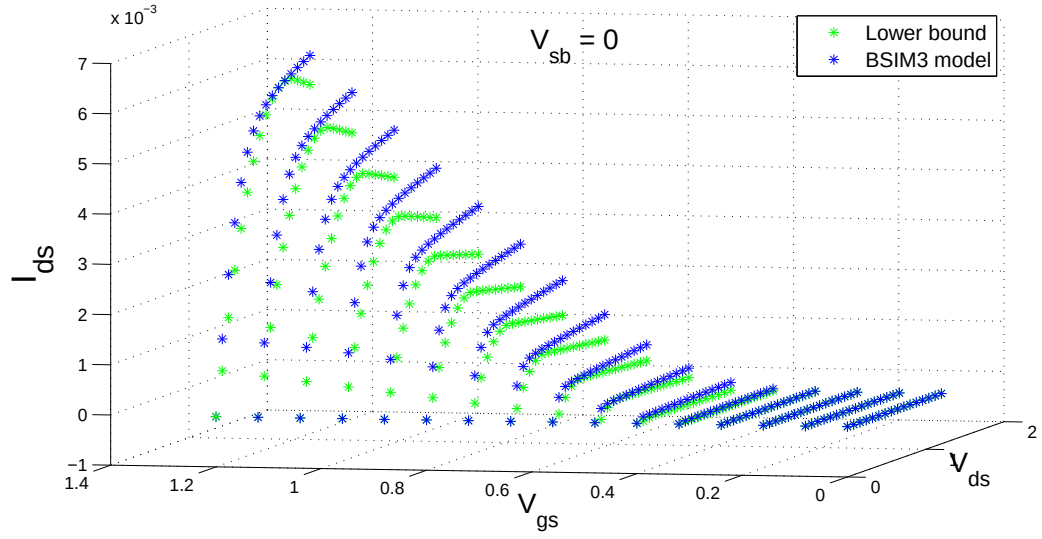
As for the problem formulation, the KCL constraints $I_{p1} = I_{n1}, I_{p2} = I_{n2}, I_{p3} = I_{n3}$ are first gathered. Then, IV characteristics described by the bounded models for each transistor are added. The target resolution of refinement is set to 0.001. For the ring oscillators with *odd* number of stages, our method returns a single solution

$$V_1 \in [0.599, 0.6], V_2 \in [0.599, 0.6], V_3 \in [0.599, 0.6], \ldots \tag{3.11}$$

while for the ring oscillators with even number of stages, our method find three set of



(a) Upper bound



(b) Lower bound

Fig. 4. Bounded models of the NMOS transistors

solutions. The solutions are as follow

$$\begin{cases} V_1 \in [0.599, 0.6], V_2 \in [0.599, 0.6], V_3 \in [0.599, 0.6], \ldots \\ V_1 \in [0, 0.001], V_2 \in [1.199, 1.2], V_3 \in [0, 0.001], \ldots \\ V_1 \in [1.199, 1.2], V_2 \in [0, 0.001], V_3 \in [1.199, 1.2], \ldots \end{cases} . \tag{3.12}$$

From the device parameters shown in the Figure 3, we know that gate width of the PMOS transistors are triple as large as the gate width of the NMOS transistors. On the other side, as mentioned earlier, the carrier mobility of the PMOS transistors are $1/3$ of the carrier mobility of the NMOS transistors. Considering these two facts, it is expected that there is an equilibrium point around $V_{dd}/2$, i.e., 0.6. Thus the solution for ring oscillator with odd number of stages is as expected. Besides, the HSPICE DC simulations for the ring oscillators with odd number of stages indicate that the DC equilibrium point indeed resides within the located region. There're three solutions for the ring oscillators with even number of stages. We had explained that the first one is as expected. And considering the inverting function of the inverters, the rest two solutions for the ring oscillator with even number of stages are also as expected.

Our experiment also compares the runtime of our two-layer method with the runtime of [13]. In the first layer of our method, the SAT solver is called to quickly find a small candidate region by solving the problem formulated with the bounded device model. In the second layer, we apply the refinement technique introduced in [13] on the small candidate region to get solution of desired resolution to the ring oscillators. In the second group of experiments, the method of [13] is directly applied on the entire state space to search for solutions of ring oscillators. For ring oscillators with up to 16 stages, the runtime results are shown in Table I. Our method has considerable advantage in terms of runtime. The reason is that the first layer can save a great amount of runtime for the second layer by restricting its initial search space.

Table I. Runtime of the ring oscillators verification

#stages	The proposed 2-layer method [s]	[13] method [s]	speedup
11	36.73	51.85	1.4
12	110.76	129.09	1.2
13	64.89	368.93	5.7
14	86.85	1226	14.1
15	134.74	4072	30
16	118.58	17223	145

CHAPTER IV

TRANSIENT VERIFICATION

A. Context

In the transient analysis of a circuit, the time dependent voltage and/or current response to a given input is the object. In the mathematical context, the object of transient analysis is to find the solution $\vec{x}(t)$ for the differential Equation 3.1 to a given input $\vec{u}(t)$. However, differential equations solving can be quite difficult or even impractical when the corresponding circuit is of reasonable large size and complex behaviors. In practise, numerical integration methods are used by transient simulators to approximate the continuous analytical solution $\vec{x}(t)$ with a set of discrete values $\underline{\vec{x}}(t_0)$, $\underline{\vec{x}}(t_1)$, $\underline{\vec{x}}(t_2)$, ..., where each $\underline{\vec{x}}(t_k)$ is an approximation to the $\vec{x}(t_k)$.

There are various numerical integration methods such as forward Euler (FE), backward Euler (BE), trapezoidal method (TR), etc [20]. Take BE for example. Assuming the state of the circuit at time t_k is known to us, BE method approximates the Equation 3.1 as follow

$$\frac{\underline{\vec{x}}(t_{k+1}) - \vec{x}(t_k)}{t_{k+1} - t_k} = f(\underline{\vec{x}}(t_{k+1}), \vec{u}(t_{k+1})) \tag{4.1}$$

where $\vec{x}(t_k)$ is the state of the circuit at time t_k and $\vec{u}(t_{k+1})$ is the input value at time t_{k+1}. The solution $\underline{\vec{x}}(t_{k+1})$ is an approximation to the circuit's state at time t_{k+1}.

In transient simulation, starting from an given initial condition (either an operating point or a certain state set by the user) the BE method is iteratively applied to calculate the next reachable state of the circuit. In the end, an approximation of the trajectory of the circuit behavior starting from the given initial condition is obtained.

In the verification of a circuit, we usually concern its dynamic behaviors stemming

from a *range* of initial condition. When simulation method is used to fulfill verification, a sophisticated scheme must be first designed to get samples in the initial range of interest. Then a number of simulation runs are performed for each sampling initial conditions. Simulation is incomplete in nature while it is relatively cheaper than the SAT-based method which is complete. With this observation, we propose a transient verification method combining simulation and SAT. The flow of our proposed transient verification method will be described in detail later in this chapter.

As mentioned in Chapter I, modeling checking which explores the state space of circuit behavior is an important class of methods in AMS verification. In terms of state space exploration, there are a variety of choices [6, 13]. One choice is the *unroll* strategy in which the circuit Equations 4.1 at each discrete time point are combined and solved all together [13]. Another choice is the *reachability analysis* in which the circuit Equations 4.1 are solved one by one [6]. In transient analysis, long simulation time is usually necessary to observe the dynamical behavior of a circuit to determine whether the properties of interest hold or not. This fact sets strict limitation to the application of the unroll strategy. When thousands or more time points are involved, the scalability issue is likely to make the unroll strategy impractical even for a simple circuit. Therefore, in this work, we implement our transient verification technique in the reachability analysis fashion. The rest of this chapter is organized as follow. First, the method of representing the state space in the reachability analysis is introduced. Then, the algorithm of our transient verification method is provided. In the end, two experiments are presented to demonstrate the application of our method.

B. State space representation and related issues

In the reachability analysis, starting from a given initial space, the algorithm iteratively calculates the next reachable space. In this work, we use the fixed-grid boxes to represent

the reachable space in the reachability analysis. An illustration is given by the Figure 5.

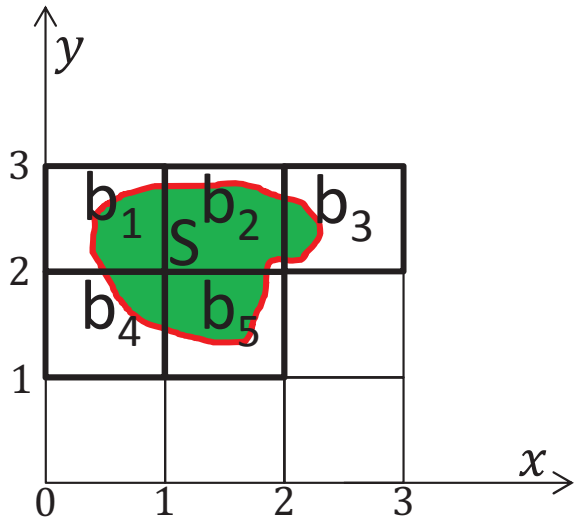


Fig. 5. An example of the fixed-grid representation

For simplicity, Figure 5 only shows a 2-dimensional state space (x, y). In this figure, the current reachable space S is circled by the red line and marked by green color. With the fixed-grid boxes, we shall use the union of five boxes to represent the space S as follow

$$\begin{aligned} &(x, y) \in b_1 \vee \\ &(x, y) \in b_2 \vee \\ &(x, y) \in b_3 \vee \\ &(x, y) \in b_4 \vee \\ &(x, y) \in b_5 \end{aligned} \tag{4.2}$$

Clearly, while the fixed-grid representation provides conservativeness, it does introduce over-approximation as any other geometrical state space representation methods. As we can see in Figure 5, there are spaces not belong to S in each of the five boxes and these space is the over-approximation introduced by the fixed-grid representation. The over-approximation will accumulate step by step. Without careful handling, the over-

approximation might lead to a situation that no firm conclusion can be drawn because the reachable space is too conservative. In this work, we increase the *resolution* of the grid to alleviate the accumulation of the over-approximation.

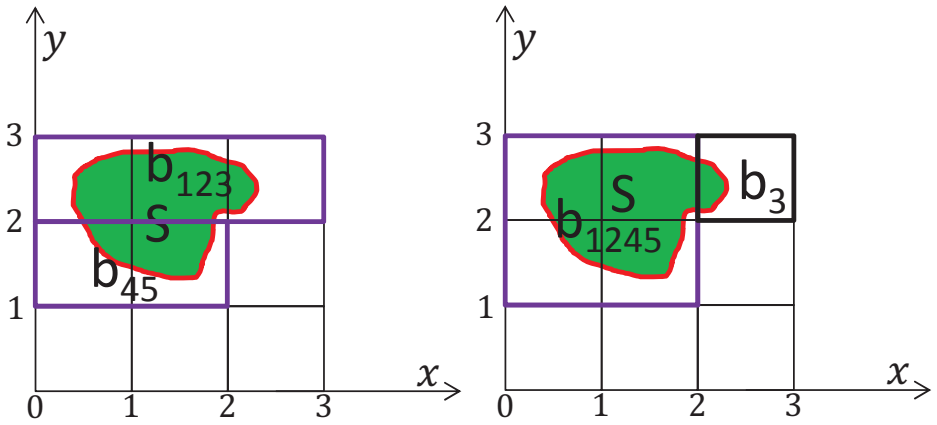
A higher resolution, though reduces the over-approximation, has side effects. As it will be discussed later in this chapter, the number of boxes used to represent the reachable space closely relates to the size of constraints and the number of times the SAT solver is called. And the latter two factors strongly affect the runtime efficiency of the verification technique. In short, the more the boxes are used to represent the reachable space, the greater the runtime is. Therefore, reducing the box number is imperative. In our work, we develop an box merging algorithm to achieve this goal.

In order to explain the merge algorithm, it is necessary to explain how a box is actually represented in a constraint. For the reachable space shown in Figure 5, the constraint (Equation 4.2) is coded as follow

$$\begin{aligned}
&(x \in [0,1] \wedge y \in [2,3]) \vee \quad &&//(x,y) \in b_1 \\
&(x \in [1,2] \wedge y \in [2,3]) \vee \quad &&//(x,y) \in b_2 \\
&(x \in [2,3] \wedge y \in [2,3]) \vee \quad &&//(x,y) \in b_3 \\
&(x \in [0,1] \wedge y \in [1,2]) \vee \quad &&//(x,y) \in b_4 \\
&(x \in [1,2] \wedge y \in [1,2]) \quad &&//(x,y) \in b_5
\end{aligned} \tag{4.3}$$

where the intervals specify the range for each dimension of the boxes. With this representation, we can vary the size of a box even though the resolution of the grid is fixed. Figure 6 shows two different merge results from the same set of boxes in Figure 5.

In Figure 6 (a), the boxes are merged horizontally. Boxes b_1, b_2, and b_3 are merged to form a larger box b_{123} while boxes b_4 and b_5 are combined to form another larger box b_{45}. The two larger boxes resulted from the merges are outlined by purple boundaries in



(a) (b)

Fig. 6. Merged boxes

Figure 6 (a). The merges can make the constraint (Equation 4.3) to be simplified as follow

$$\begin{aligned} &(x \in [0,3] \wedge y \in [2,3]) \vee \quad //(x,y) \in b_{123} \\ &(x \in [0,2] \wedge y \in [1,2]) \qquad //(x,y) \in b_{45} \end{aligned} . \tag{4.4}$$

Figure 6 (b) shows a different merge result in which boxes b_1, b_2, b_4, and b_5 are merged to form a large square box b_{1245} while box b_3 is left alone. The resulted constraint is as follow

$$\begin{aligned} &(x \in [0,2] \wedge y \in [1,3]) \vee \quad //(x,y) \in b_{1245} \\ &(x \in [2,3] \wedge y \in [2,3]) \qquad //(x,y) \in b_3 \end{aligned} . \tag{4.5}$$

The flow of our box merging algorithm is shown in Algorithm 4. There are two list of boxes. One stores the unmerged boxes, and the other stores the merged boxes. We refer to the lists as L_{merged} and $L_{ummerged}$ respectively. Initially, $L_{ummerged}$ stores the set of boxes to be merged and L_{merged} is empty. The algorithm visits each box b_i in $L_{ummerged}$ one by one and find if b_i can be merged to any of the boxes in the L_{merged}. If b_i can be merged, then the corresponding box in L_{merged} will be updated to the box resulted from the merge. If not, b_i is inserted into L_{merged}. After all box in $L_{ummerged}$ is visited, the algorithm compares the number of boxes in L_{merged} and in $L_{ummerged}$. If the numbers are the same, which means

that the last run of visit does not lead to any merge and so the algorithm stops. Otherwise, $L_{ummerged}$ dumps all its content and takes all the content of L_{merged}, after which L_{merged} is cleared to be an empty list again. Then another run of visit to the boxes in $L_{ummerged}$ will be performed. This process repeats until the size of L_{merged} equals the size of $L_{ummerged}$ after a run of visit.

Algorithm 4 merge($L_{ummerged}$)

```
1:  while TRUE do
2:    L_merged = ∅;
3:    for each box b_i ∈ L_ummerged do
4:      if there exists a box b'_j ∈ L_ummerged that can be merged with b_i then
5:        b'_j = b'_j + b_i; //merge
6:      else
7:        insert b_i into L_merged;
8:      end if
9:    end for
10:   if sizeof(L_merged) == sizeof(L_ummerged) then
11:     break;
12:   else
13:     L_ummerged = L_merged;
14:   end if
15: end while
16: return L_merged
```

Note, the ordering of the boxes in the list $L_{ummerged}$ can affect the result of the merge. An example has been shown in Figure 6. In Figure 6 (a), a possible ordering of the five

boxes in the list $L_{ummerged}$ is [b_1, b_2, b_3, b_4, b_5], where b_1 is the head and b_5 is the tail. Initially, the list L_{merged} is empty. In list $L_{ummerged}$, box b_1 is the first box to be visited by the merge algorithm. Since L_{merged} is empty when b_1 is visited, b_1 must be inserted into L_{merged} because no merge can happen. The second box being visited is box b_2. When the algorithm traverses the L_{merged}, it will finds that there is a box b_1 in L_{merged} which can be merged with b_2. Then in the list L_{merged}, the box b_1 is replaced by a larger box b_{12} resulted from the merge of b_1 and b_2. After one visit to each box in the $L_{ummerged}$, the algorithm shall get the merge result shown in the Figure 6 (a). For Figure 6 (b), a possible ordering of the boxes in the list $L_{ummerged}$ is [b_1, b_4, b_2, b_5, b_3], where b_1 is the head and b_5 is the tail. Similar to the previous example, box b_1 is the first one in the list $L_{ummerged}$ to be visited. Since the list L_{merged} is empty at this moment, b_1 is inserted to L_{merged}. The second box being visited is box b_4. Since box b_1 can be merged to b_4, in the list L_{merged} the box b_1 is replaced by a larger box b_{14}, which is the result from the merge of b_1 and b_4. Compared to the merge result of the previous example at the same stage (when the visit to the second box in $L_{ummerged}$ is completed), we can see that the difference of the merge result is due to the different orderings of the list $L_{ummerged}$ in these two examples.

Before moving to the next section, a short discussion on the possible benefits from the fixed-grid representation is provided. Compared to other geometrical state space representations, like zonotopes or polytopes, fixed-grid boxes are more flexible for reuses. With the fixed-grid representation, we can build up a look-up table of the causal relationship between each two boxes in the state space during the process of the reachability analysis. Before calculating the next reachable state space, we can first look up the table using the boxes representing the current reachable space as index. The boxes found during the look-up process, if exists, can be directly taken as the next reachable space without the need to solve the circuit equations. The look-up table can even be set as share information to facilitate the parallel computation of the reachable space. Though these features of the fixed-grid

representation aren't made used of in this work, the author believes that it is worthwhile to mention them to the readers.

C. Algorithm of reachability analysis

Reachability analysis is performed by iteratively applying a propagation algorithm which calculates the next reachable space based on the given current reachable state. In reachability analysis, we formulate the circuit equations as follow. First, for a given circuit, we formulate its behavior with the Equation 3.1. Then we approximate this equation with certain numerical integration method. In this work, BE method is adopted and so the behavior of the circuit is approximated using the Equation 4.1. Essentially, Equation 4.1 represents the mapping from the current state $\vec{x}(t_k)$ to the next state $\underline{\vec{x}}(t_{k+1})$. For simplicity, from now on we refer to $\vec{x}(t_k)$ as x_k and $\underline{\vec{x}}(t_{k+1})$ as x_{k+1} in this thesis. Then Equation 4.1 can be represented in the following form

$$mapping(x_k, x_{k+1}) = 0 \tag{4.6}$$

The rest of this section will first introduce the basic propagation algorithm which powered by NLN-SAT only. Then the simulation-assisted propagation algorithm which combines the strength of simulation and SAT will be described.

1. Basic propagation algorithm

Algorithm 5 shows the procedure of the basic propagation algorithm. The input is a set of boxes, S_k, representing the current reachable space. The object of the algorithm is to find the next reachable space S_{k+1} which is also represented by a set of boxes.

Initially, S_{k+1} is set to be an empty set. The input box set S_k is processed and updated using the merge algorithm. After the merge procedure, there will be much fewer boxes in

Algorithm 5 Basic propagation algorithm (input: S_k)

1: $S_{k+1} = \varnothing$;
2: $S_k = \text{merge}(S_k)$;
3: **for** each box $b_i \in S_k$ **do**
4: $Next = \varnothing$;
5: construct problem formula Q;
6: // Q: $mapping(x_k, x_{k+1}) = 0 \wedge x_k \in b_i \wedge x_{k+1} \notin Next$
7: call NLN-SAT solver;
8: **while** $solution \neq$ UNSAT **do**
9: find box, where $solution \in box$;
10: $Next = Next \cup box$;
11: $Next = \text{merge}(Next)$;
12: update formula Q;
13: call NLN-SAT on formula Q;
14: **end while**
15: $S_{k+1} = S_{k+1} \cup Next$;
16: **end for**
17: **return** S_{k+1}

S_k while it still represents the same reachable space. Then the algorithm uses a for-loop to calculate the next reachable state space $Next$ for *each* box b_i in S_k. The union of $Next$ for each b_i is the resulted reachable space we're looking for. Within the for-loop, a problem formula Q is set up by combining the circuit Equation 4.6 with the constraints on current and next reachable space. The constraint $x_{k+1} \notin Next$ serves as a guide for the solver by blocking the already located next reachable space from being searched again. At first, $Next$ is empty. After the problem is formulated, an inner while-loop is used to iteratively call the NLN-SAT solver until all reachable space stemming from b_i is located.

Within the while-loop, the algorithm selects the box which contains the solution returned by the solver. Although this selection does introduce over-approximation, it preserves the conservativeness of the solution and leads to less iteration of the while-loop. Each time after a reachable box is located and inserted into $Next$, the merge procedure will be called on $Next$ to reduce its number of boxes. The problem formula Q with the updated version of $Next$ is then fed to the NLN-SAT solver again.

Figure 7 illustrates how many times the NLN-SAT solver is called in the basic propagation algorithm. For simplicity, the figure only shows a 2-dimensional state space (x, y). The current reachable space is represented by a single box b_1. After one time step, b_1 propagates to b'_1, b'_2, and b'_3. Although it isn't necessary that the *actual* next reachable space entirely covers all the space in those boxes b'_i ($i = 1$ to 3), we nevertheless use the union of those boxes to represent the next reachable space since in the fixed-grid method the boxes like b'_1 represent the highest resolution we can get.

For the example illustrated in Figure 7, the Algorithm 5 calculates the next reachable space for b_1 as follow. First, the merge routine finds that no merge can be performed. Thus, S_k contains a single box b_1. Since the reachable space is propagated to b'_i ($i = 1$ to 3), the solution returned by the solver can come from any of these three boxes. Assume that the first solution the solver finds is within the box b'_1. After the first solution is returned, boxes

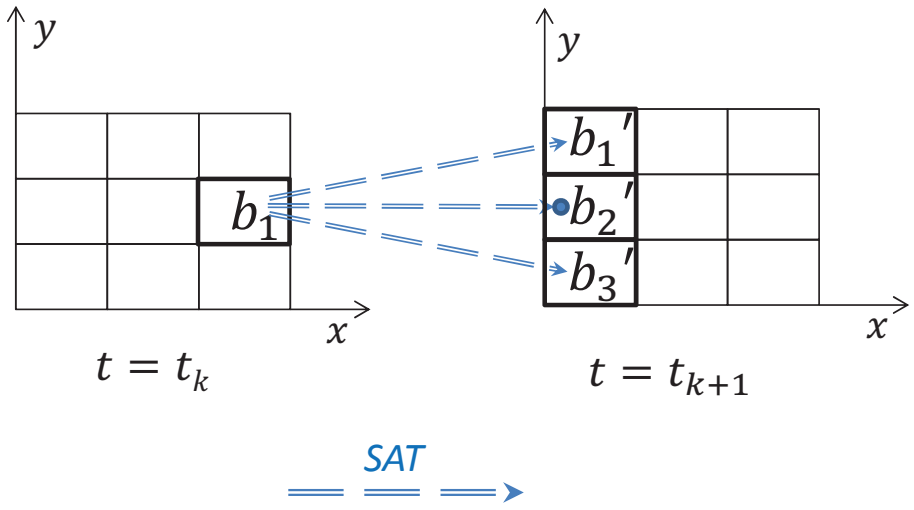


Fig. 7. Propagation of the reachable space

b_1' will be inserted into the box set $Next$ which stands for the reachable space stemming from b_1. The constraint $x_{k+1} \notin Next$ will prevent the solver from searching the space in b_1' again. Therefore, further calls to the solver will only return solution within the rest two boxes b_i' ($i = 2$ to 3). Clearly, the next two calls to the solver will mark the remaining two boxes as reachable space one by one. In the end, the fourth and last call will return a *UNSAT* result which means no solution can be found in the space outside of the identified boxes b_i' ($i = 1$ to 3). Considering the guarantee on unsatisfiability from the solver, $b_1' \cup b_2' \cup b_3'$ does conservatively represent the next reachable space stemming from b_1.

2. Simulation-assisted propagation algorithm

Although the NLN-SAT technique is rather powerful considering that it can handle SAT problems with boolean combination of a large number of constraints over the real domain, it is still less efficient than transient simulation on the job of finding the solution. However, the NLN-SAT technique can be used to guarantee the conservativeness of the reachable space which is the benefit simulation can't provide. Therefore, we combine transient simu-

lation with the NLN-SAT technique to build an reachability analysis method with both efficiency and completeness. We implement the transient simulation method through a C++ implementation of the Equation 4.6 which takes the value of x_k as the input and returns the value of x_{k+1} as the output.

The proposed reachability method works as follow. At first, we uniformly sample the current state space to get a set of sampling points. Then, for each sampling point, we calculate its projection point in the next time point using simulation. After a projection point is located, the box containing the point is regarded as the next reachable space. It is true that different sampling points can lead to projection points in the same box which means that not all the sampling points provide no information. However, considering that the simulation method covers most part of the next reachable space very efficiently, integrating circuit simulation based sampling into the verification flow is beneficial. After all the sampling points are processed via simulation, the NLN-SAT methods is called to find the unidentified reachable space in the same way as is in the Algorithm 5.

In general, the simulation method is in charge of quickly covering the major part of the next reachable space. The rest part is then located by a few calls to the NLN-SAT solver. The pseudo code of this simulation-assisted propagation algorithm is shown in Algorithm 6. Besides, a few techniques such as analytical bound calculation and constraint simplification are also implemented to increase the runtime efficiency of the reachability analysis.

D. Experiment on a tunnel diode oscillator

We first apply our transient verification method on a tunnel diode oscillator to verify its start-up condition. The rest of this section is organized as follow. First, the behaviors of the circuit are provided along with the problem formulation in the first subsection. Then,

Algorithm 6 Simulation-assisted propagation algorithm (input: S_k)

$S_{k+1} = \varnothing$;
$S_k =$ merge(S_k);
for each box $b_i \in S_k$ **do**
  $Next = \varnothing$;
  //sampling and simulation
  $SP = sampling(b_i)$; //SP is the set of sampling points
  for each point $p \in SP$ **do**
    x_{k+1} = simulation(p);
    find box, where $x_{k+1} \in box$;
    $Next = Next \cup box$;
  end for
  //call for NLN-SAT
  $Next =$ merge($Next$);
  construct problem formula Q;
  // Q: $mapping(x_k, x_{k+1}) = 0 \wedge x_k \in b_i \wedge x_{k+1} \notin Next$
  call NLN-SAT solver;
  while $solution \neq$ UNSAT **do**
    find box, where $solution \in box$;
    $Next = Next \cup box$;
    $Next =$ merge($Next$);
    update formula Q;
    call NLN-SAT on formula Q;
  end while
  $S_{k+1} = S_{k+1} \cup Next$;
end for
return S_{k+1}

the experimental setup, result and analysis are presented in the second subsection.

1. Model of the tunnel diode oscillator

The schematic of the tunnel diode oscillator is shown in Figure 8.

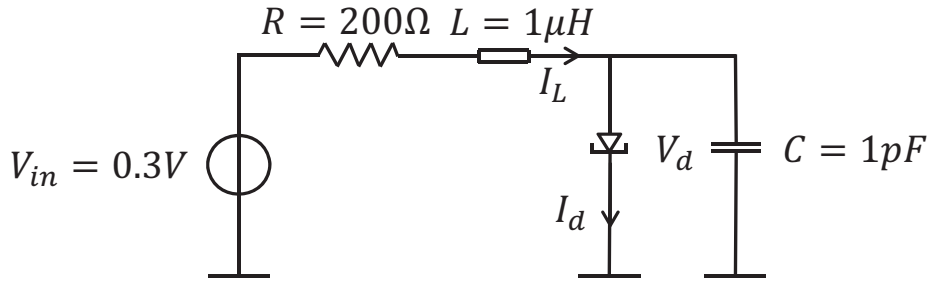


Fig. 8. Schematic of a tunnel diode oscillator

There're two independent state variables in the behavior space of the tunnel diode oscillator. The first one is the voltage V_d across the capacitor. And the other is the current I_L flowing through the inductor. The state equations which characterize the behaviors of the circuit are as follow

$$\begin{cases} I_L = I_d + C \cdot \dot{V}_d \\ V_{in} = R \cdot I_L + L \cdot \dot{I}_L + \dot{V}_d \end{cases} \tag{4.7}$$

Equations 4.7 are approximated using BE method in this work.

The voltage to current characteristic of the diode is captured by the following piece-wise nonlinear function

$$I_d(V_d) = \begin{cases} 6.01V_d^3 - 0.992V_d^2 + 0.0545V_d, & v_d \leq 0.055 \\ 0.0692V_d^3 - 0.0421V_d^2 + 0.004V_d + 8.96 \cdot 10^{-4}, & 0.055 \leq v_d \leq 0.35 \\ 0.263V_d^3 - 0.277V_d^2 + 0.0968V_d - 0.0112, & v_d \leq 0.35 \end{cases} , \tag{4.8}$$

where its curve is shown by Figure 9.

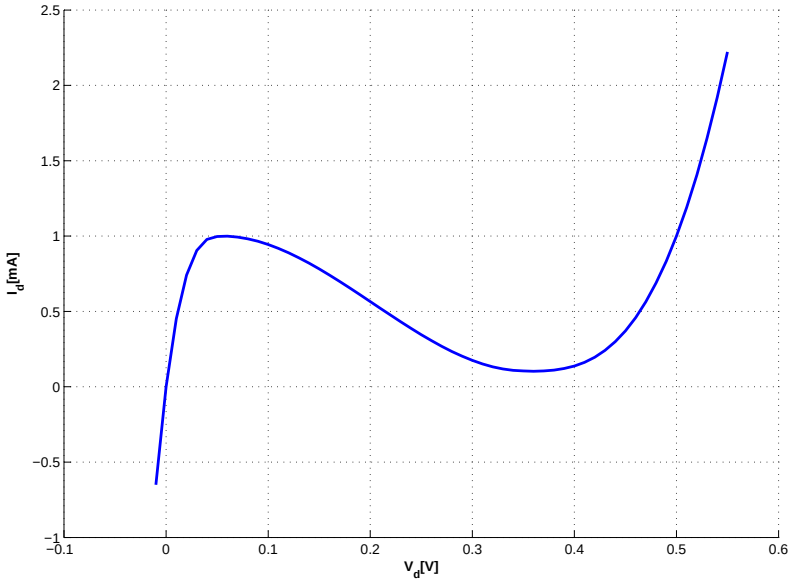


Fig. 9. Voltage-to-current characteristic of the diode

In our reachability analysis, the behavior of the tunnel diode oscillator is represented by the conjunction of the Equation 4.8 and the BE approximation of Equation 4.7.

2. Experimental setup and results

The parameters of the circuit are as shown in the Figure 8: $R = 200\Omega$, $L = 1\mu$H, $C = 1$pF, and $V_{in} = 0.3$V. The time step of reachability analysis is $\Delta t = 0.2$ns. The resolution of fixed-grid representation: $\Delta I_L = 0.001$mA and $\Delta V_d = 0.001$V. The range of initial condition of interest is $I_L = 0.6$mA and $V_d \in [0.42\text{V}, 0.52\text{V}]$. The object of our experiment is to verify that starting from any value within the given initial range, the tunnel diode oscillator can actually generate oscillation. The result of a 70-step reachability analysis is shown in Figure 10.

Figure 10 verifies that the tunnel diode oscillator can guarantee to generate oscillation starting from any state within the given initial range.

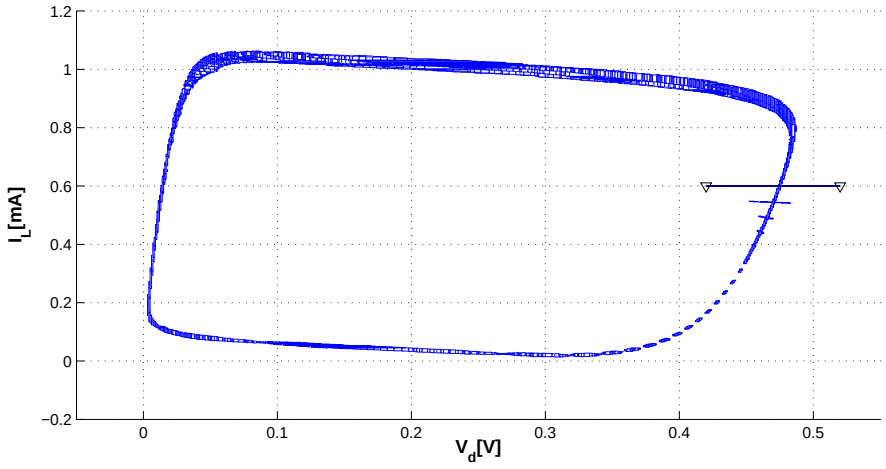


Fig. 10. Reachable space of the voltage-to-current characteristic of the diode

In the experiment, we observe that the sampling rate affects the runtime and the number of calls to the simulation routine and the SAT solver. We perform 7 runs of the 70-step reachability analysis, each with a different sampling rate. The runtime result is shown in Table II.

Table II. Runtime of the tunnel diode oscillator verification with various sampling rates

Sampling interval	#sim	sim time[s]	#SAT	SAT time[s]	sim time + SAT time [s]
1/3	0.9M	19.4	27808	6293.5	6312.9
1/4	1.6M	33.9	24128	4130.3	4164.2
1/5	2.5M	51.3	21450	3254.7	3306.0
1/6	3.4M	72.5	22905	3709.6	3782.1
1/7	4.6M	97.2	22448	3219.4	3316.6
1/20	34.5M	725.7	22292	3335.3	4061.0
1/91	703.1M	14966	21709	3254.7	18220.7

Note the definition of the sampling interval in Table II is as follow. For a given sampling interval p and the resolution Δx for the variable x being sampled, the interval between each two adjacent samples is $p \times \Delta x$. Table II shows the general trend that by increasing the sampling rate, the number of simulation runs increases while the number of SAT solver

calls decreases. Correspondingly, the runtime for simulation is increased while the runtime for SAT solver is decreased. With a number of experiments, it is possible to find out the best sampling interval which leads to the optimal runtime efficiency. In our experiment, the best sampling interval for the tunnel diode oscillator should be around $1/5$.

E. Experiment on a charge pump PLL

In the second experiment, we apply our transient verification technique to a more challenging circuit, a charge pump PLL. The object of the experiment is to check whether the PLL can achieve phase lock in a given period of time. Considering the complexity of the PLL circuit, it is envisioned that modeling abstraction (i.e. behavioral models in our case) is necessary to make the reachability analysis of PLL practical. In the rest of this section, the behavioral models of the PLL will be described in detail at first, followed by the experiment results and analysis.

1. Behavioral model of the PLL

Figure 11 shows the block diagram of the charge pump PLL. The reference signal *ref* is compared to the output signal *div* coming from a $1/N$ frequency divider by a phase frequency detector (PFD). The PFD generates its output signals based on the phase/frequency difference between *ref* and *div*. PFD's output controls the current i_{cp} running through the charge pump (CP) which in turns controls the input voltage v_1 of the voltage-controlled oscillator (VCO) through a loop filter. The output frequency f_v of VCO, which is directly controlled by v_1, is fed back to the PFD as signal *div* after going through the $1/N$ frequency divider. The feedback effect will finally set the frequency f_v to be around N times of the frequency of *ref*.

The PFD is implemented using two D flip-flops (DFFs) as shown in Figure 11. Signals

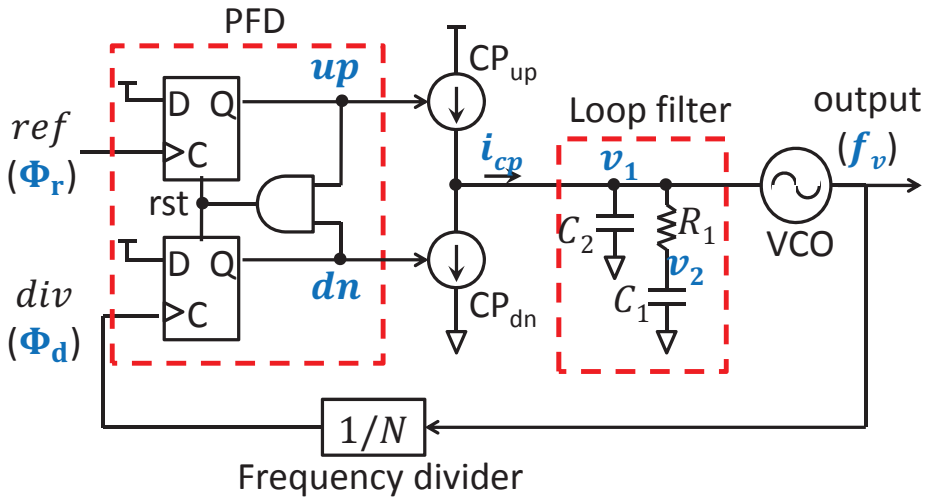


Fig. 11. Block diagram of a charge pump PLL

ref and *div* along with the current state of signals *up* and *dn* determines the next state of *up* and *dn*. Assuming the initial state of both *up* and *dn* are low (L). If the signal *ref* takes the lead and generates a rising edge at first, then the signal *up* will become high (H), which in turn makes the CP pumps current i_{cp} into the loop filter. The voltage v_1 will increase because of the injection of current i_{cp}. Finally, the increase of voltage v_1 will lead to the increase of output frequency f_v which makes the feedback signal *div* catch up with *ref*. When *div* eventually catches up and generates a rising edge of dn, the AND gate in PFD will output a reset signal to drag down both signals *up* and *dn* and in so doing the CP is turned off. Due to this reset mechanism, the state when both *ref* and *div* are H can be safely ignored. In this thesis, the next state of a variable x is denoted by x'. The transition rules just mentioned can be described by the propositional logic constraints as follow

$$(\Phi_r > 2\pi \wedge \Phi_d < 2\pi \wedge up = L \wedge dn = L)$$
$$\Rightarrow (\Phi_r' = \Phi_r - 2\pi \wedge \Phi_d' = \Phi_d \wedge up' = H \wedge dn' = L) \tag{4.9}$$
$$(\Phi_r < 2\pi \wedge \Phi_d > 2\pi \wedge up = H \wedge dn = L)$$
$$\Rightarrow (\Phi_r' = \Phi_r \wedge \Phi_d' = \Phi_d - 2\pi \wedge up' = L \wedge dn' = L) \tag{4.10}$$

where Equation 4.9 describes the condition when *ref* takes the lead at first and Equation 4.10 describes the condition when *div* catches up. Note the above two constraints does not fully describe transition rules of the PFD. Since the other constraints not listed are of the same form, we don't write them all here for simplicity.

The timing diagram below will help further illustrate the working principle of the PFD.

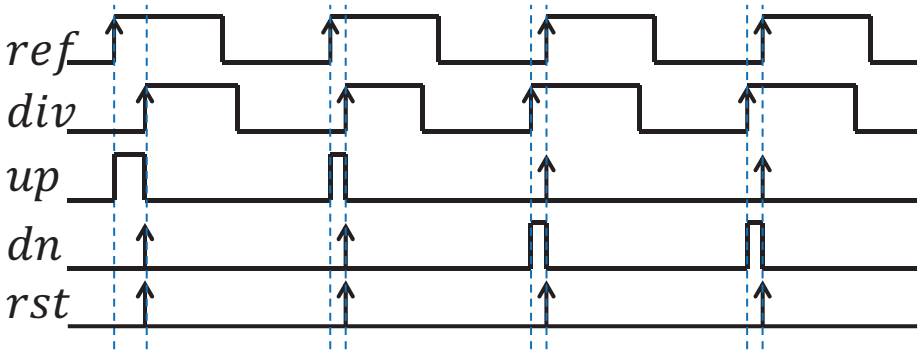


Fig. 12. Timing diagram of the PFD

The constraints controlling the CP are given by Equations 4.11. When the signal *up* is H, the CP pumps current i_{cp} into loop filter. When the signal *dn* is H, the CP pumps current i_{cp} out of the loop filter. When both the signals *up* and *dn* are L, the CP is turned off and no current is pumped in or out. As mentioned earlier, the duration of the state when *up* and *dn* are both H is so short that this state is not considered in the behaviors of the CP.

$$i_{cp} = \begin{cases} i_{up} & \text{if } up = H \wedge dn = L \\ 0 & \text{if } up = L \wedge dn = L \\ i_{dn} & \text{if } up = L \wedge dn = H \end{cases} . \tag{4.11}$$

The KVL and KCL constraints governing the loop filter are as follow

$$\begin{cases} C_2 \dfrac{\mathrm{d}v_1}{\mathrm{d}t} + C_1 \dfrac{\mathrm{d}v_2}{\mathrm{d}t} = i_{cp} \\ R_1 C_1 \dfrac{\mathrm{d}v_2}{\mathrm{d}t} + v_2 = v_1 \end{cases} . \tag{4.12}$$

Since the NLN-SAT technique does not take differential equation as input, we approximate the Equation 4.12 using BE method.

For the VCO, we use a piecewise nonlinear function (Equation 4.13) to capture its voltage-to-frequency behavior which is shown in Figure 13.

$$f_v(v_1) = \begin{cases} 0, & v_1 \leq 0.29 \\ 183.9v_1^2 - 104.4v_1 + 14.81, & 0.29 < v_1 \leq 0.50 \\ -121.1v_1^2 + 201.4v_1 - 61.84, & 0.50 < v_1 \leq 0.745 \\ -30.98v_1^2 + 66.96v_1 - 11.701, & 0.745 < v_1 \leq 1.0 \end{cases} . \tag{4.13}$$

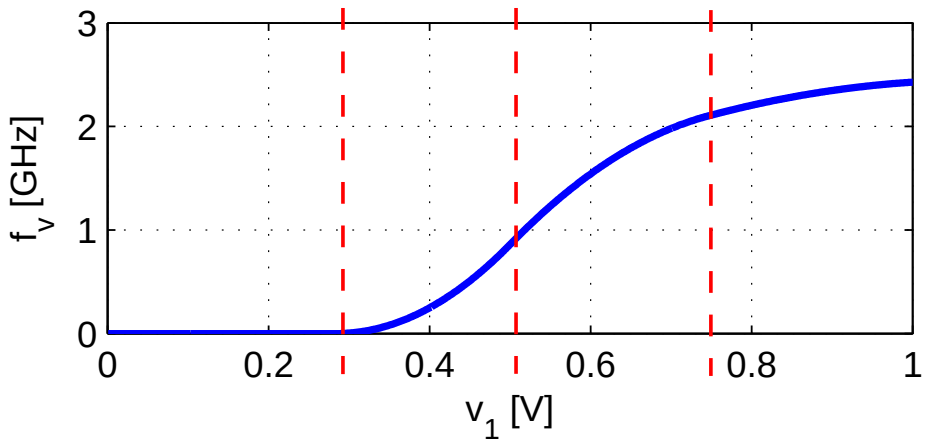


Fig. 13. Voltage-to-frequency characteristic of the VCO

It is worthwhile to mention that we don't model the internal delay for the VCO in this work, which means for any given voltage input v_1 the output frequency f_v *immediately* jumps to the steady state. However, it is not difficult to include the behavior of delay to the behavior model. One choice is to approximate the delay by adding a RC circuit at the output of the VCO.

Finally, considering the PLL as a whole, there are three independent continuous state variables, Φ_d , v_1, and v_2, and two independent discrete state variables, *up* and *dn*. The

current i_{cp} is directly controlled by the state of *up* and *dn* and thus can't be taken as an independent state. Φ_r is completely determined by its initial value $\Phi_r(0)$ and the frequency f_r of reference signal *ref*. Most importantly, as an input, Φ_r is irrelevant to the inherent characteristics of the PLL circuit. Therefore, Φ_r is also not regarded as an independent continuous state variable. The discrete state variables of PLL lead to three discrete state space $(L,\ L)$, $(L,\ H)$, and $(H,\ L)$. Within each discrete state space, there resides the continuous behaviors of the PLL. The hybrid automaton of the charge pump PLL is shown by Figure 14. Large number of switches among the discrete spaces, as is the case in PLL dynamical behavior, is one of the reasons that makes verification a difficult task for AMS circuit involving hybrid automaton because switch leads to more computational expense and over-approximation than non-switch propagation.

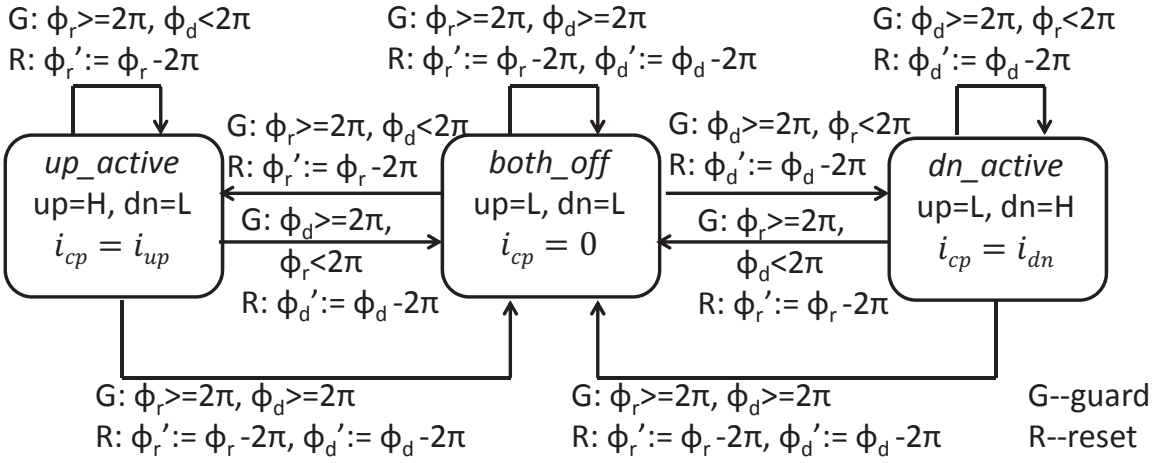


Fig. 14. Hybrid automaton of the charge pump PLL

2. Experimental setup and results

As mentioned in the beginning of this section, the object of this experiment[1] is to check whether the PLL can achieve phase lock in a certain period of time. In our experiment, the certain period of time is 2000 ns. The criteria of locking of phase is $\|\Phi_r - \Phi_d\| < 5\% \times 2\pi$. For convenience, the valid region of phase is normalized from [0,2π) to [0,1) in our experiment. Therefore, the criteria of phase locking must be modified to $\|\Phi_r - \Phi_d\| < 0.05$.

The parameters of the circuit are set as follow: reference frequency $f_r = 10$MHz, $C_1 = 2.5$pF, $C_2 = 0.6$pF, $R_1 = 160$kΩ, $N = 100$. The time step used in this reachability analysis is $\Delta t = 1$ns, $1/100$ of the reference period. The range of each independent continuous variable Φ_d, v_1, and v_2 is shown below in the Table III along with the valid combinations of the discrete state variable up and dn.

Table III. Range of the state variables

Variable	Valid location
v_1	[0,1]
v_2	[0,1]
ϕ_d	[0,1)
(*up*, *dn*)	(H,L), (L,L), (L,H)

The initial condition for the reachability analysis is set up as follow: $\Phi_r = 0.955$, $\Phi_d \in$ [0.8,0.9], $v_1 \in$ [0.47V, 0.48V], $v_2 \in$ [0.72V,0.73V], *up* $= L$, and *dn* $= L$. The resolution of the fixed-grid representation is $\Delta\Phi_d = 0.01$, $\Delta v_1 = 0.01$V, and $\Delta v_2 = 0.01$V. Note, since we don't include the internal delay effect of the VCO into the behavioral model, there is no power-on process of the VCO in our experiment. The result of 2000-step reachability

[1]Experiment environment: 4-core Intel CPU Q9450, 8 GB memory, Ubuntu.

analysis for the PLL is shown in Figure 15.

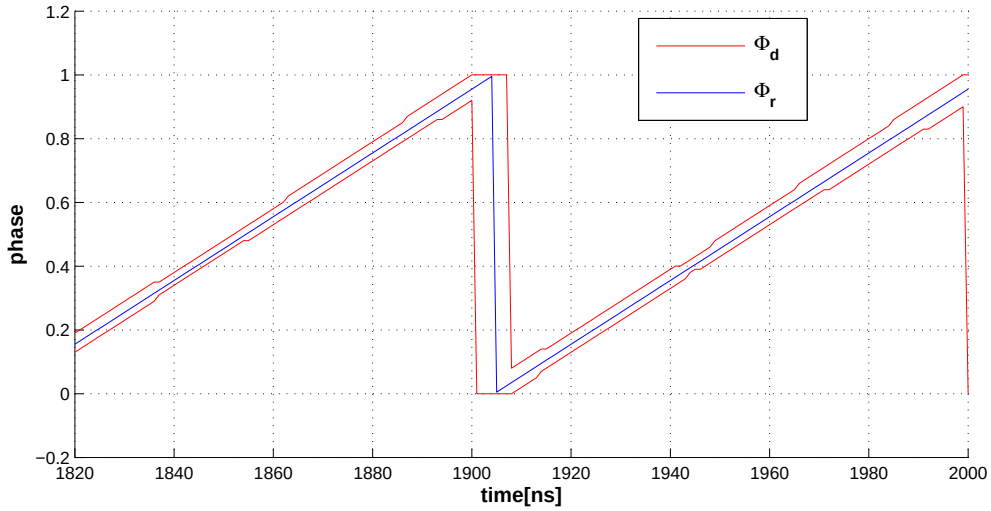


Fig. 15. Φ_r versus Φ_d

The result shows that apart from the discrete behavior region (i.e. where phase jumps from 1 to 0), the PLL successfully achieves lock of phase under the aforementioned criteria.

Similarly to the experiment on tunnel diode oscillator, we also vary the sampling rates during the simulation process to find the best combination of simulation and SAT. We perform 10 runs of the 2000-step reachability analysis, each with a different sampling rate. The runtime result is shown in the Table IV.

Note, the last row of Table IV does not finish the 2000-step reachability analysis. The data shown in the last row is the statistics at 588 steps. The definition of the the sampling interval is the same as in the experiment on the tunnel diode oscillator. The Table IV shows that the optimal sampling interval for the charge pump PLL is around $1/20$.

At last, we use an experiment to show the effect of the box merging technique. The experiment performs two runs of a 10-step reachability analysis with $\Delta t = 1ns$. The parameters and initial condition of the circuit are the same as before. In the first run, the

Table IV. Runtime of the PLL verification with various sampling rates

Sampling interval	#sim	sim time[s]	#SAT	SAT time[s]	sim time + SAT time [s]
1/2	14M	1.8	13237	2132.1	2133.9
1/3	19M	2.4	11827	1935.5	1937.9
1/4	27M	3.3	10733	1820.9	1824.2
1/5	35M	4.4	9387	1636.9	1641.3
1/6	48M	6.0	9113	1578.4	1584.4
1/7	62M	7.5	9514	1806.6	1814.1
1/8	76M	9.1	8679	1509.2	1518.3
1/9	113M	13.6	7325	1134.5	1148.1
1/20	481M	61.3	6516	952.4	1013.7
1/100	20938M	2543.0	2958	240.4	2783.4

program uses the merge technique while in the second run the program does not. It can be seen from the Figure 16 that without the help of merge technique, the increase of runtime will make it impractical to perform a reachability analysis over large number of time steps.

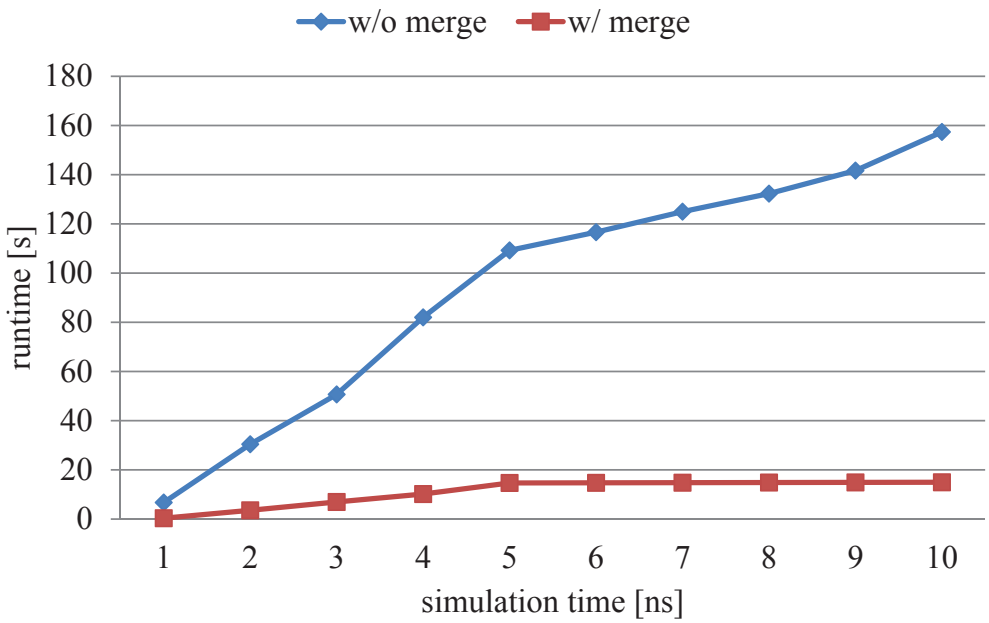


Fig. 16. Merge's effect on runtime

CHAPTER V

CONCLUSION

In this work, we propose two approaches for AMS verification based on circuit modeling, nonlinear SAT solver, and simulation techniques.. One is for DC verification, and the other is for transient verification.

For DC verification, we introduces a two-layer method. In the first layer, we formulate the circuit problem with the conservative bounded device models. Then we apply the SAT solver to find the solution of the formulated problem. In the second layer, a refinement technique developed in [13] is applied to reduce the size of the solutions previously found to desired resolution. We apply our method to find the DC equilibrium points for a set of ring oscillators. The experiment result shows that our method has considerable speedup over the previous work [13].

For transient verification, we perform reachability analysis by combining the simulation and SAT. The transient verification method first quickly finds most part of the next reachable space with simulation. Then SAT solver is applied to locate the rest unidentified part of the next reachable space. We also introduce a box merging algorithm to efficiently represent the state space using fix-grid boxes. We apply our method on a tunnel diode oscillator and a charge pump PLL. Both the experiments successfully verify the properties of interest. Besides, we also vary the sampling rate during the simulation process to find the optimal combination of simulation and SAT in terms of runtime.

REFERENCES

[1] M. H. Zaki, S. Tahar, and G. Bois, "Formal verification of analog and mixed signal designs: a survey," *Microelectronics Journal*, vol. 39, pp. 1395–1404, Dec. 2008.

[2] A. Singh and P. Li, "On behavioral model equivalence checking for large analog/mixed signal systems," in *IEEE/ACM ICCAD*, Nov. 2010, pp. 55–61.

[3] S. Steinhorst and L. Hedrich, "Advanced methods for equivalence checking of analog circuits with strong nonlinearities," *Formal Methods in System Design*, vol. 36, pp. 131–147, 2010.

[4] W. Hartong, L. Hedrich, and E. Barke, "Model checking algorithms for analog verification," in *IEEE/ACM DAC*, 2002, pp. 542–547.

[5] S. Little, D. Walter, K. Jones, C. J. Myers, and A. Sen, "Analog/mixed-signal circuit verification using models generated from simulation traces," *Int. J. Found. Comput. Sci.*, vol. 21, no. 2, pp. 191–210, 2010.

[6] G. Frehse, B.H. Krogh, and R.A. Rutenbar, "Verifying analog oscillator circuits using forward/backward abstraction refinement," in *IEEE/ACM DATE*, Mar. 2006, vol. 1, pp. 257–262.

[7] S. Gupta, B.H. Krogh, and R.A. Rutenbar, "Towards formal verification of analog designs," in *IEEE/ACM ICCCAD*, Nov. 2004, pp. 210–217.

[8] B.I. Silva and B.H. Krogh, "Formal verification of hybrid systems using CheckMate: a case study," in *American Control Conference*, 2000, vol. 3, pp. 1679–1683.

[9] M. Althoff, A. Rajhans, B. H. Krogh, S. Yaldiz, X. Li, and L. Pileggi, "Formal verification of phase-locked loops using reachability analysis and continuization," in *IEEE/ACM ICCAD*, Nov. 2011, pp. 659–666.

[10] W. Denman, B. Akbarpour, S. Tahar, M.H. Zaki, and L.C. Paulson, "Formal verification of analog designs using metitarski," in *Formal Methods in Computer-Aided Design*, Nov. 2009, pp. 93–100.

[11] M. R. Greenstreet and S. Yang, "Verifying start-up conditions for a ring oscillator," in *Proc. of the Great Lakes Symp. on VLSI*, May 2008, pp. 201–206.

[12] M. R Prasad, A. Biere, and A. Gupta, "A survey of recent advances in SAT-based formal verification," *International Journal on Software Tools for Technology Transfer*, vol. 7, no. 2, pp. 156–173, 2005.

[13] S.K. Tiwary, A. Gupta, J.R. Phillips, C. Pinello, and R. Zlatanovici, "First steps towards SAT-based formal analog verification," in *IEEE/ACM ICCAD*, Nov. 2009, pp. 1–8.

[14] M. H. Zaki, I. M. Mitchell, and M. R. Greenstreet, "DC operating points analysis – a formal approach," presented at the Formal Verification of Analog Circuits workshop, Grenoble, France, June 2009.

[15] M. Franzle, C. Herde, T. Teige, S. Ratschan, and T. Schubert, "Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure," *J. on Satisfiability, Boolean Modeling, and Computation*, vol. 1, pp. 209–236, 2007.

[16] A. Eggers, C. Herde, S. Kupferschmid, K. Scheibler, and T. Teige, "HySAT: A sounded model checker for hybrid systems," [Online]. Available: http://hysat.informatik.uni-oldenburg.de/.

[17] A. Eggers, C. Herde, S. Kupferschmid, K. Scheibler, and T. Teige, "iSAT: Tight integration of satisfiability & constraint solving," [Online]. Available: http://isat.gforge.avacs.org/.

[18] W. Liu, X. Jin, J. Chen, M-C. Jeng, and Z. Liu et al, "BSIM 3v3.2 MOSFET model users' manual," EECS Dept., Univ. of California, Berkeley, CA, Tech. Rep. No. UCB/ERL M98/51, 1998.

[19] "Star-Hspice Manual," Release 1998.2, Avant! Co., Fremont, CA, July 1998, ch. 16, pp. 1–7.

[20] T. L. Pillage, R. A. Rohrer, and C. Visweswariah, "Linear Transient Analysis I," in *Electronic Circuit & System Simulation Methods*, 1st ed. New York: McGraw-Hill, 1994, ch. 4, pp. 75–82.

VITA

Name: YUE DENG

Address: Department of Electrical and Computer Engineering,
Texas A&M University,
111 Wisenbaker Engineering Research Center,
TAMU 3259
College Station, TX 77843-3259

Email Address: dengyue43@gmail.com

Education: B.S., Automation, Xi'an Jiaotong University, 2009
M.S., Computer Engineering, Texas A&M University, 2012