Runtime Verification of Analog and Mixed Signal Designs

Zhiwei Wang

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science (Electrical & Computer Engineering)

at

Concordia University

Montréal, Québec, Canada

June 2009

Your file Votre référence
ISBN: 978-0-494-63186-7
Our file Notre référence
ISBN: 978-0-494-63186-7

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Zhiwei Wang**

Entitled: **Runtime Verification of Analog and Mixed Signal Designs**

and submitted in partial fulfilment of the requirements for the degree of

Master of Applied Science (Electrical & Computer Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Dr. Amir G. Aghdam

_____ Dr. Hon Fung Li

_____ Dr. Rabin Raut

_____ Dr. Sofiène Tahar

Approved by _____

Chair of the ECE Department

_____ 2009 _____

Dean of Engineering

ABSTRACT

Runtime Verification of Analog and Mixed Signal Designs

Zhiwei Wang

Analog and mixed signal (AMS) circuits play an important role in system on chip designs. They pose, however, many challenges in the verification of the overall system due to their complex behaviors and expensive consumption of simulation resources. Besides functionality, AMS systems also suffer from stochastic processes such as random noise which exhibits statistical properties. Among many developed verification techniques, runtime verification has been shown to be effective by experimenting finite executions instead of going through the whole state space. In this thesis, we propose a methodology for the verification of AMS designs using functional and statistical runtime verification. Functional runtime verification is used to check the functional behavior of the AMS design. A system of recurrence equation (SRE) is used to model the AMS design and construct a functional property monitor. This functional runtime verification is carried out in an *online* fashion. Statistical runtime verification is used to verify the statistical properties of the AMS design. Hypothesis test, which is a method to make statistical decisions about rejecting or accepting some statement about the information of a sample, is used to verify the statistical properties. We use Monte Carlo simulation for the hypothesis test and for evaluating its performance. The proposed methodology is applied to a phase lock loop based frequency synthesizer where several functional properties and stochastic noise properties are verified.

ACKNOWLEDGEMENTS

It has been an amazing experience to accomplish my Master's thesis in the Hardware Verification Group (HVG) at Concordia. It certainly would not have happened without the support and guidance of several people to whom I owe a great deal.

First of all, I would like to thank my supervisor, Dr. Sofiène Tahar. It is he who offered me the opportunity to join the group. He was fully supportive, understanding, involved and present during all the phases of my research. I have learned many things from him in regard to research, academia, and life in general.

Secondly, I would like to thank Dr. Mohamed Zaki. He introduced me to the topic of this thesis and guided me in the right direction. His wisdom, elegance of thought and extreme kindness were always very inspiring to me.

Next, let me thank all the members of HVG for their help and encouragement. Their friendship brought me a warm environment in the lab. I especially appreciate the collaboration with Naeem Abbasi, Rajeev Narayanan and William Denman. My work, from the first conference paper to my Master's thesis, has benefited a lot from their valuable advice and technical support.

Last but not least, this thesis could not have been done without the unconditional love from my parents who always give me their full support. In addition, I would like to give many thanks to Amy for her full understanding and spirit support during my research.

To my parents, grandfather, Lindai, and the memory of my grandmother

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

A/D	Analog to Digital Converter
AF	Analog Filter
AMS	Analog and Mixed Signal
AMT	Analog Monitoring Tool
ASIC	Application Specific Integrated Circuits
CDF	Cumulative Distribution Function
COMP	Comparator
CP	Charge Pump
CPU	central Processing Unit
CSL	Continuous Stochastic Logic
CT	Continuous Time
CTL	Computational Tree Logic
D/A	Digital to Analog Converter
DAE	Differential Algebraic Equations
DC	Direct Current
DE	Discrete Event
DIV	Divider
DJ	Deterministic Jitter
DSP	Digital Signal Processor
DT	Discrete Time
EDA	Electronic Design Automation
FL	Foundation Language
FPGA	Field-Programmable Gate Array
GPS	Global Positioning System
FSM	Finite State Machine
HDL	Hardware Description Language

HOL	Higher Order Logic
IP	Intellectual Property
LHA	Linear Hybrid Automata
LTL	Linear Temporal Logic
MaC	Monitoring and Checking
MEDL	Meta Event Definition Logic
MITL	Metric Interval Temporal Logic
MSA	Mixed Signal Assertions
OBE	Optional Branching Extension
PDF	Probability Density Function
PFD	Phase and Frequency Detector
PLL	Phase Locked Loop
PMF	Probability Mass Function
PSL	Property Specification Language
PVS	Prototype Verification System
RF	Radio Frequency
RJ	Random Jitter
SERE	Sequential Extension Regular Expressions
SoC	System on Chip
SRE	System of Recurrence Equation
STL	Signal Temporal Logic
VCO	Voltage Controlled Oscillator
VHDL	Very (High Speed Integrated Circuits) Hardware Description Language

Chapter 1

Introduction

1.1 Motivation

With the constant growth in integrated circuit technology, the number of transistors per chip has been doubling almost every two years according to Moore's Law [10] and that figure passed two billion by the end of 2008 [43]. As a consequence of high level integration, more complex functionalities can be realized in compact systems such as smart cell phones and portable game consoles. In modern design methodologies, instead of putting every transistor separately on the multi-million-transistor chip, functional components are integrated as building blocks in one chip. Because of this, System on Chip (SoC) architecture has prevailed for the last decade. It may contain digital, analog, mixed signal and radio frequency functional units in one chip. Although SoC designs have been driving the semiconductor industry, as shown in Figure 1.1, the growth of design productivity has been lagging behind the improvement in the number of transistors per chip by as much as 37% [49]. The rate of the first silicon success is one of the main reasons for this gap. Figure 1.2 shows that this rate dropped to 35% in 2003. As the performance of SoC continues to improve, AMS components are considered a bottleneck in improving the overall performance of the system and the factor for enhancement of the first silicon success

rate. 70% of re-spun designs contain functional bugs [9]. To overcome these obstacles, 70% of the total design effort in the semiconductor industry is now spent on verification. The failure of analog and mixed signal (AMS) components has been one of the major causes for the high design re-spun rate [9]. The design and verification of AMS systems became very important in recent years.

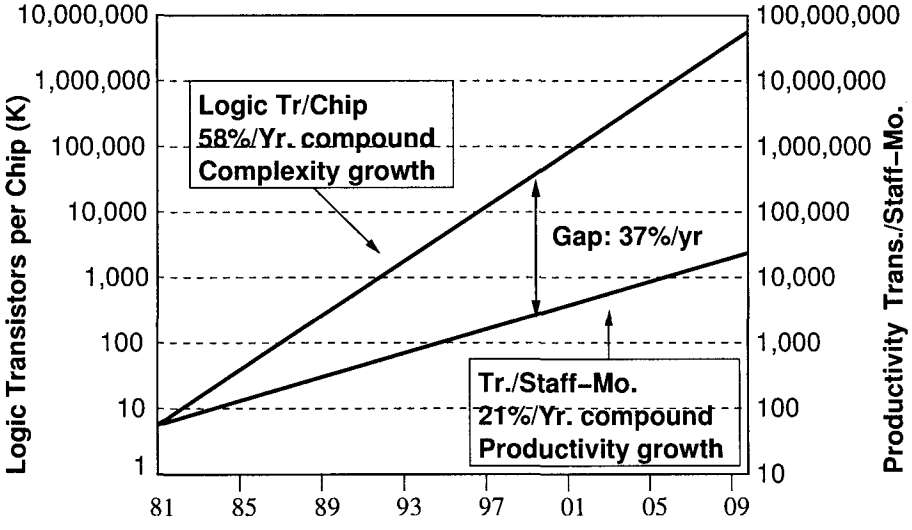


Figure 1.1: Design Productivity Gap [49]

We live in an analog world. The SoC system of the design operates in the real analog environment as well. The data is processed by the digital components inside the system. The class of components which connect the analog world to digital domain processors is called Analog and Mixed Signal (AMS). Examples of AMS designs include analog filters, frequency synthesizer, digital to analog converters (D/A), and analog to digital converters (A/D). They can be Intellectual Property (IP) cores or the interface between them.

Simulation is traditionally used to check whether a design exhibits the proper behavior as elicited by a series of tests. However, pure simulation approach is feasible only when the expected results can be done manually and the state space is not complete [50]. For more complex systems, especially AMS designs, pure simulation

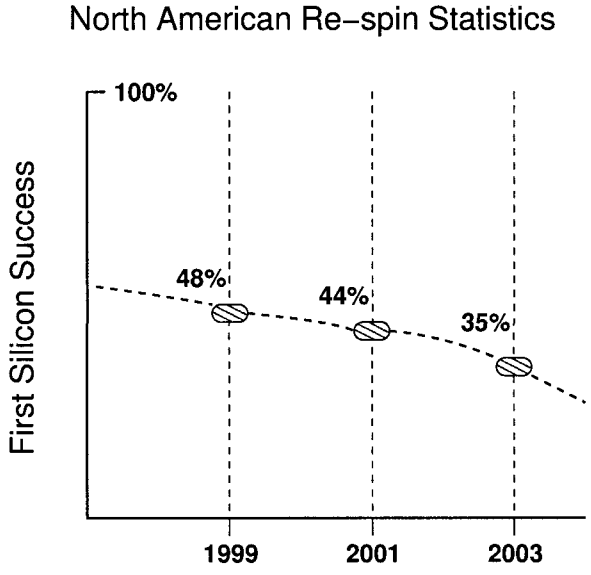


Figure 1.2: The Rate of First Silicon Success [49]

finds itself not competent for functional verification due to the coverage issue and long simulation time. With the evolution of hardware description languages (HDL) extensions tailored for AMS designs, such as VHDL-AMS [46], Verilog-AMS [45] and SystemC-AMS [44], the complex behaviors of an AMS system can be described at different abstraction levels. Recently, many Electronic Design Automation (EDA) companies have dedicated to the development of efficient AMS simulators. All these efforts have allowed the AMS modeling and specification to reach a new level together with the accuracy and efficiency. However, the development of verification of AMS systems has been lagging behind the design due to the limitations in platforms and different levels of the abstraction. A complete verification methodology integrating modeling and verification is desired.

In order to solve the coverage issue, formal verification has been advanced in recent years. Formal verification is a mathematical reasoning procedure to prove that an implementation satisfies its specification. The implementation can be a description of the design at any abstraction level, such as behavior level or Register Transfer Level (RTL). The specification refers to a correct description, or a

desired property, of the system to be checked. There are three main techniques of formal verification method [25]: theorem proving, equivalence checking, and model checking.

In theorem proving, the relationship between specification and implementation is defined as a theorem to be proven through a deductive procedure with a set of axioms and inference rules. Theorem proving, a proof based method, is able to perform various verification tasks at different abstraction levels. Although it is a powerful verification technique, the entire proof procedure requires considerable amount of manual effort. The two most common higher order logic provers are HOL [18] and PVS [39].

In industry, equivalence checking is widely used to compare two models of a system to check whether they are functionally equivalent. In digital domain, equivalence checking exhibits its efficiency for systems of moderate size. In an AMS system, the continuous signal representation makes the alternative real value model difficult to find. Hence, equivalence checking is not suitable for AMS designs.

Model checking, known as a state exploration method, is used to find out whether the model of the design satisfies a given temporal specification. The model refers to a space which contains all the possible states of the system. A state exploration algorithm is then applied to determine whether the model satisfies the property. If the property does not hold, a counterexample is reported at the state where the violation occurs. Model checking is an automatic technique. However, when scaling up to large circuits, it suffers inevitable problems such as state explosion which limits the computational resources in terms of memory and time. In the case of AMS designs, the situation is even worse because the analog signal in continuous domain can only be expressed using infinite states. Bounded model checking [8] technique has been adapted to cope with the state explosion issue. However, the price paid is the incomplete coverage issue due to the bounded approximation.

Runtime verification combining simulation and formal specification, which is

considered a semi-formal verification approach, has been adopted to complement formal methods. By evaluating finite execution traces instead of exploring them all, runtime verification avoids the state space explosion. The formal specification ensures the coverage of the verification. A survey on the formal verification of AMS designs using the above mentioned techniques can be found in [50].

The general motivation of this thesis is to present a complete methodology for the runtime verification of ASM designs. This methodology contains a new modeling technique, a functional runtime verification and a statistical runtime verification method. The modeling technique unifies the expression of analog and digital signal representation in order to simulate them in the same environment. The runtime online verification method constructs the monitor based on the formal specification of the system property using the same modeling technique . The statistical runtime verification method is capable of analyzing stochastic processes occurring in AMS designs. In following we will introduce the notions of runtime verification and its application in AMS designs followed by discussions of related works.

1.2 Runtime Verification

In recent years, runtime verification has been developed to bridge the gap between formal verification and traditional simulation methods. Initially serving in software verification, runtime verification can also be applied to hardware verification especially when formal methods and conventional simulation encounter practical obstacles. The most distinguishing feature of runtime verification is that the verification procedure is accomplished at runtime based on the simulation traces. The feature that no computational model is needed prior to the verification avoids the state space explosion problem [50]. Runtime verification deals with the detection of violation, as well as satisfaction, of the property. A monitor is used to detect the violation. The monitoring technique can be performed in two ways, namely,

online and *offline* monitoring. *Online* monitoring, which is used to check a current execution of a system when the simulation is running, is able to detect a property violation as soon as it occurs. On the other hand, *offline* monitoring operates on a set of recorded executions after the simulation is done.

Runtime verification can be grouped into functional runtime verification and statistical runtime verification in terms of different kinds of properties to be verified. In the following, we will introduce the functional and statistical runtime verification, respectively.

Functional Runtime Verification

Functional runtime verification is used to determine if the design satisfies a specified functional property [34]. The functional property refers to a specification that indicates the correct operation or how the system will function. For AMS designs, examples of functional properties include whether an oscillator oscillates or if a phase lock loop locks at the desired frequency.

Compared with model checking, functional runtime verification deals with finite executions instead of walking through large state space. This allows runtime verification to be applied to the AMS system whose entire system model is available at higher level of abstraction. In addition, the verification points are easy to be set up according to the finite traces. In model checking, the state exploration algorithm usually requires a complete generation of the state space before executing. This prevents online monitoring applicable for model checking. Runtime verification does not have this problem except that the requirement is needed. The advantage of functional runtime verification over conventional simulation approach is that runtime verification uses a formal specification for the property, which can be a trace or several individual observation points, instead of evaluating the inputs and output pairs in simulation. In addition, *online* monitoring technique allows the verification to terminate as soon as the violation is detected. This is important especially for

such AMS designs requiring long time in simulation. The formal specification also introduces more confidence in runtime verification than with simulation.

In functional runtime verification, the monitor is generated from a high level system specification, for which a high level modeling is needed. In this thesis, a high level modeling approach is introduced, and is then used to generate the monitor. Because of the consistency between the modeling and monitoring techniques, the *online* runtime verification of AMS design becomes feasible.

Statistical Runtime Verification

In this thesis, we propose a statistical runtime verification approach to investigate statistical properties of AMS systems. A statistical property refers to the property which deals with the stochastic behavior of the system and is analyzed using statistical methods. Examples of the statistical properties are mean, variance, and standard deviation. In AMS designs, the system suffers from different kinds of random noise such as thermal noise and jitter noise. We propose an approach for the verification of statistical properties using Monte Carlo simulation [33] and statistical hypothesis testing [28]. In statistical hypothesis testing, two hypotheses are made based on the property to be verified. The two hypotheses are exclusive to each other. The rejection of one leads to the acceptance of the other. In order to perform a hypothesis test, the distribution of the parameter of interest, or statistic, is expected to be known. Sometimes the distribution of the sample data is not known in advance. In such case, Monte Carlo simulation is applied to estimate the statistic model to perform the hypothesis test. Each decision made by hypothesis test has an associated confidence level. A 100% in confidence level is usually not realistic in random process. It is unlikely that the statistics estimated from a random sample is exactly equal the true value of the population parameter. The confidence interval is used to enclose the estimated value. The confidence level indicates the probability that the estimated value presents itself in such interval. The difference between the

bounds of the interval and the estimated value is called error margin. The assumed statistic model could result in some errors. The Monte Carlo method is then used to assess the performance of the hypothesis test conducted. For many properties of a stochastic process, it is usually acceptable to receive either a violation or satisfaction of the property with a bounded confidence level and error margin.

In next section, we will introduce the works related to functional runtime verification and statistical runtime verification, respectively.

1.3 Related Work

1.3.1 Functional Runtime Verification

Runtime verification originated in software verification initially. Recently, several notable efforts have been made to verify AMS designs using runtime verification. In one of the most prominent works [30], the authors present an offline methodology for monitoring the simulation of continuous signals. The monitoring technique was based on Signal Temporal Logic (STL) [30] which is an analog extension of Metric Interval Temporal Logic (MITL) [4]. The simulation and monitoring was conducted using Matlab/Simulink [32]. In [37], the authors synthesized the Property Specification Language (PSL) analog extension (STL/PSL) into an Analog Monitoring Tool (AMT). The tool is capable of both offline and incremental monitoring. In a recent case study [22], the authors investigated the verification of a DDR2 SDRAM memory using AMT in an offline mode. The approach mentioned synthesizes the property in terms of lower abstraction levels such as a finite state machine (FSM). The work we propose in this thesis applies to *online* monitoring, and presents a unified framework for both modeling and verification at a higher level of abstraction.

In [17], the authors propose an online monitoring technique. They used the linear hybrid automata (LHA) as a monitor to analyze the reachability of time domain features. A hybrid system analysis tool named HPAVer [16] was used to

verify the signal amplitude and jitter properties of an oscillator circuit. In order to avoid infinite memory required, necessary approximations were employed and the assumption of the existence of templates to build the monitor was also done. The computational expense of this technique is high because the work is based on formal verification and state space analysis rather than linear temporal logic (LTL) [14]. In general, it does not provide a generic way to obtain the monitors from the specification. In contrast, the approach we advance is capable of modeling and monitoring the AMS design and supports PSL as the specification language. An FPGA implementation of assertion based monitor is presented in [36]. The authors used PSL to generate an asynchronous monitor which is robust to process, temperature and voltage variations and suitable for ASIC designs. Nevertheless, the work in [36] is unable to support both analog and mixed circuits or sequential extended regular expression (SERE) [2] in PSL.

A more recent work [21] introduces a methodology to define mixed signal assertions (MSA) for verification by combining PSL and STL. In this work the specifications for digital and analog parts are translated into PSL and STL, respectively, as either precondition or postcondition. An MSA is then constructed by combining the precondition and postcondition with an implication. Assertion based verification or formal verification could then be carried out given the formalized properties. The authors applied the MSA to a first order delta-sigma converter and checked several properties. The work was validated within the MLDesigner [35] tool with an enhanced assertion monitoring library. In the research reported in this thesis, we use one single formalism, namely SRE (System of Recurrence Equation) [3], to express PSL properties for both analog and digital parts. Additionally, we offer a complete methodology including a simulator and an online monitor. In [3], an offline assertion based verification is introduced, where SREs are used to model the AMS design. Our work is different from [3] in two aspects. First, we use online monitoring to achieve verification. Secondly, we present a tool, named C-SRE [1], which

simulates AMS designs modeled with SREs, reads PSL properties in SRE notations and performs the online monitoring.

1.3.2 Statistical Runtime Verification

Statistical verification can be divided into three main categories: statistical theorem proving, statistical model checking and statistical runtime verification. Although several interesting advances have been made in statistical theorem proving, this technique is still in its infancy. The theorem for continuous random variables and random processes is needed to handle the analysis and verification of AMS and hybrid system [19]. The model checking method has been advanced first to complement general model checking. In [47], the authors present an independent model checking approach for verifying probabilistic properties of discrete event systems. The probabilistic properties were expressed using continuous stochastic logic (CSL) [5] formulas. These formulas were then verified through Monte Carlo simulations and statistical hypothesis testing. The verification procedure provides two parameters, α and β, which represent the probability of making a wrong decision in checking whether a formula is true or false. In a related work [48], the author presents a probabilistic model checking method to bound the probability of error, mentioned in [47], for the indifferent region (i.e., the region where both acceptance and rejection decisions can not be made). A symmetric polling system was studied to demonstrate the performance of the method. Following the statistical model checking approach in [47, 48], the authors in [7] applied this technique to a class of AMS circuits for the first time. The saturation property of a third order delta-sigma converter was verified both in time and frequency domains. However, the issues of state explosion and excessive computation time still prevail in statistical model checking.

Statistical runtime verification has also been investigated in the past. One of the most important works is [42] where the authors introduce a methodology

to verify quantitative and probabilistic properties in a real-time system at runtime. The quantitative specification was realized using Meta Event Definition Logic (MEDL) [23] which is based on LTL. The probabilistic properties are specified using time-bounded temporal operators and probabilistic operators. Statistical hypothesis testing technique was employed to evaluate the probabilistic properties and to make decisions about acceptance and rejection. Whenever the decision is made, a confidence level and error margin is provided. The monitor was implemented in a runtime verification tool termed MaC (Monitoring and Checking) [24] and performed in an online fashion. In this thesis, we present a methodology for statistical runtime verification for AMS designs.

1.4 Proposed Methodology

The general methodology for runtime verification of AMS designs is shown in Figure 1.3. The AMS design is modeled and then simulated using an AMS simulator. The properties of the AMS design are derived from the system specification. The satisfaction of the properties is checked based on the output of the simulator and the parameters of the design environment at runtime.

In this thesis, we employ the System of Recurrence Equation (SRE) [3] to model the AMS design. The simulation is done using an SRE based simulator. The properties are categorized into two classes: functional properties and statistical properties. A functional property describes the functional behavior of an AMS design. A statistical property on the other hand is used to describe stochastic or random behavior of the system. These properties are determined using statistical methods. During the runtime verification process, two different monitors are used for each kind of property. For functional property, an *online* monitor is constructed using SRE notation. For statistical property, a statistical monitor is designed to perform statistical runtime verification. The *online* monitor reports the violation or

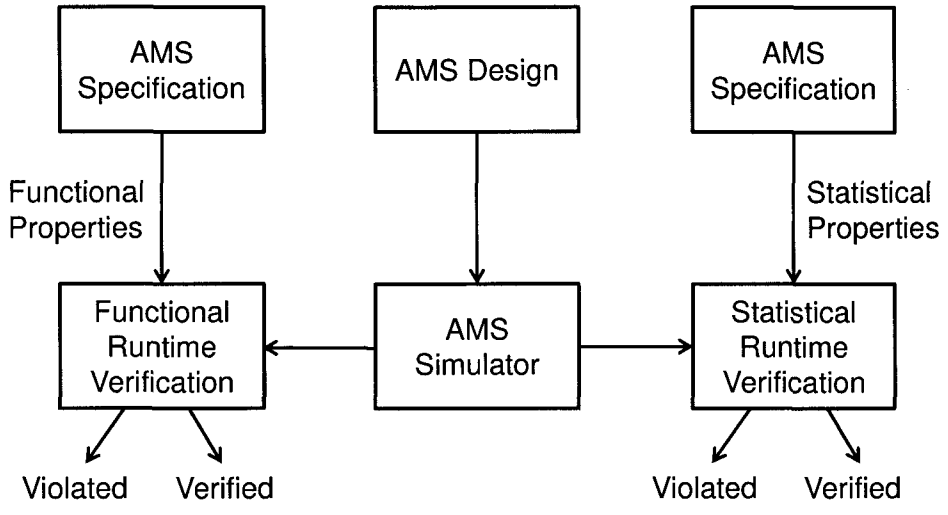


Figure 1.3: Runtime Verification Methodology

satisfaction and terminates the simulation as soon as it detects a violation. By doing this, the simulation resources are saved as the simulation for AMS design is usually very time consuming. The statistical monitor is capable of making the decision of the property satisfaction decision with the confidence level and error margin.

1.5 Phase Locked Loop

In this thesis, we will apply the proposed methodology on a phase locked loop (PLL) based frequency synthesizer as case study. A PLL is considered as a classical AMS system. Its theory was first developed by H. De Bellescize in 1932 [11]. The applications of a PLL can be found in many areas: in wireless communication, a PLL can act as a frequency synthesizer in radio frequency (RF) receivers to provide a desired frequency; for serial link and optical communications, it is used in data and signal recovery circuits; in microprocessors, it works as a clock multiplier unit. In general, PLL deals with clock or frequency for the system.

Frequency synthesizer is a basic building block in modern communication devices such as cellular phones and GPS systems. It is capable of generating a certain range of frequency. The PLL based frequency synthesizer shown in Figure 1.4 is the most widely used architecture. It is composed of a comparator (COMP), a phase and frequency detector (PFD), a charge pump (CP), an analog filter (AF), a voltage controlled oscillator (VCO) and a frequency divider (DIV). It consists of pure analog components (i.e., analog filter and VCO), pure digital components (i.e., PFD and divider) and mixed signal components (i.e., comparator and charge pump).

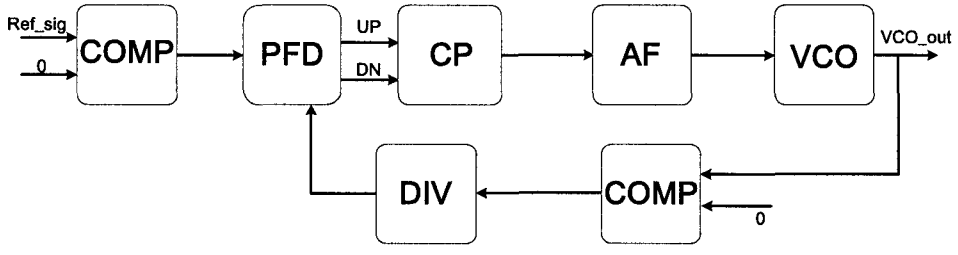


Figure 1.4: General PLL Architecture

The PLL operates on the principle of negative feedback control. The PFD detects the frequency and phase difference between the reference signal and VCO output. This difference produces appropriate voltage through the charge pump. The analog filter removes any high frequency noise of the voltage signal. The filtered voltage signal drives the VCO. The frequency of the VCO output signal is proportional to its input voltage. The VCO output is fed back to the PFD block through a frequency divider. The two comparator blocks convert sinusoid input to a square wave output of the same frequency and phase. Comparing the frequency and phase of the reference signal and VCO output feedback signal, PFD produces a new difference value and affect VCO output accordingly. This process continues until the phase and frequency of the VCO output coincide with the reference signal.

1.6 Thesis Contributions

In this thesis, a comprehensive runtime verification methodology for the Analog and Mixed Signal design is presented. The contributions of the thesis can be summarized as follows:

- We used the System of Recurrence Equations (SRE) to model the AMS design and to express PSL properties.
- We developed a runtime functional verification methodology for AMS designs. The proposed verification technique works in an *online* fashion and has the potential to save computational resources.
- We developed a statistical runtime verification methodology for AMS designs. The statistical properties are verified in an *offline* fashion, where the monitor reports the verification result along with a confidence level and error margin.
- We applied the whole runtime verification methodology to a Phase Locked Loop based frequency synthesizer as case study. The frequency synthesizer was simulated in a SRE based simulator. The functional properties were checked using online monitoring at runtime. The jitter noise properties were analyzed using the proposed statistical runtime verification.

1.7 Thesis Outline

The rest of the thesis is organized as follows: Preliminaries on SRE, PSL, and basic concepts on probability and statistics are described in Chapter 2. The SRE based simulator is also introduced in this chapter. Chapter 3 presents details of the runtime verification methodology for AMS designs including both functional and statistical runtime verification approaches. In Chapter 4, we describe the modeling and verification of the PLL based frequency synthesizer. Several interesting properties of

the PLL are checked and experimental results are described . Finally, Chapter 5 concludes the thesis and suggests avenues for future work.

Chapter 2

Preliminaries

In this chapter, we introduce the preliminary components that the runtime verification methodology is built on. They are SRE, PSL, SRE based simulator, and some basic concepts in probability and statistics as well as an introduction to Monte Carlo simulation.

2.1 The System of Recurrence Equations (SRE)

A recurrence equation or a difference equation is the discrete version of an analog differential equation [3]. A recurrence equation defines a relation between consecutive elements of a sequence. The notion of recurrence equation is extended to describe digital circuits using the normal form: generalized `If-formula` [3].

Definition 1 ***Generalized*** *`If-formula`* *The generalized* *`If-formula`* *is a class of symbolic expressions that extend recurrence equations to describe digital systems. Let i and n be natural numbers. Let $\mathbb{K}$ be a numerical domain in ($\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$ or $\mathbb{B}$), a generalized* *`If-formula`* *is one of the following:*

- *A variable $X_i(n)$ or a constant C that take value in $\mathbb{K}$*

- *Any arithmetic operation* $\diamond \in \{+, -, \times, \div\}$ *between variables* $X_i(n)$ *that take values in* $\mathbb{K}$

- *A logical formula: any expression constructed using a set of variables* $X_i(n) \in \mathbb{K}$ *and logical operators: not,and, or, xor, nor, . . . etc.*

- *A comparison formula: any expression constructed using a set of variables* $X_i(n) \in \mathbb{K}$ *and comparison operators* $\alpha \in \{=, \neq, <, \leq, >, \geq\}$

- *An expression IF(X ,Y ,Z), where X is a logical formula or a comparison formula and Y ,Z are any generalized* `If-formula`*. Here,IF(X ,Y ,Z)* : $B \times \mathbb{K} \times \mathbb{K} \longrightarrow \mathbb{K}$ *satisfies the axioms:*

  1. *IF(True,X ,Y)=X*
  2. *IF(False, X ,Y)=Y*

  The System of Recurrence Equations is defined as follows [3]:

Definition 2 ***The System of Recurrence Equations (SRE)***

Consider a set of variables $X_i(n) \in \mathbb{K}, i \in V = \{1, ..., k\}, n \in \mathbb{Z}$*, an SRE is a system of the form:*

$$X_i(n) = f_i(X_j(n - \gamma)), (j, \gamma) \in \epsilon_i, \forall n \in \mathbb{Z} \tag{2.1}$$

where $f_i(X_j(n - \gamma)$ *is a generalized* `If-formula`*. The set* ϵ_i *is a finite non empty subset of* $1, \ldots, k \times \mathbb{N}$*. The integer* γ *is called the delay.*

2.2 Property Specification Language: PSL

The Property Specification Language (PSL) is a language for the formal specification of hardware [2]. It is used to describe properties that are required to hold in the design under verification. PSL provides a means to write specifications that are

both easy to read and mathematically precise. It is intended to be used to define a functional specification on one hand and as input to functional verification tools on the other hand. Thus a PSL specification is an executable documentation of a hardware design. PSL is also an extension of the standard temporal logics LTL and CTL [2].

PSL consists of four layers: Boolean, temporal, verification and modeling layer [13]. The Boolean layer provides the Boolean expression to temporal layer. The temporal layer is the heart of PSL where complex temporal relations between signals can be expressed. The verification layer is used to tell the verification tools what to do with the behavior of the design inputs and to model auxiliary hardware that is not part of the design, but is needed for verification [2]. The modeling layer provides a means to model behavior of design inputs and to declare and give behavior to auxiliary signals and variables. Only the Boolean and temporal layers are used in our methodology for AMS runtime verification.

The temporal layer enhances LTL with regular expressions [12] and is used to describe the relationships between Boolean expressions of the Boolean layer over time. Instead of using Boolean expressions, the basic properties are employed in our methodology. The temporal layer is composed of the Foundation Language (FL) [13] and the Optional Branching Extension (OBE) [13]. The FL is used to describe properties of single traces, while OBE is used to express properties according to multiple traces. The Foundation Language is composed of two styles: LTL (Linear Temporal Logic) and SERE (Sequential Extended Regular Expression) [2]. In this thesis, we concentrate on the FL properties. Definition 1 shows the syntax of an SERE expression.

Definition 3 ***Syntax of Sequential Extended Regular Expressions (SEREs)*** *[3]*

- *if b is a Boolean expression, then b is a SERE*

- *if r is a SERE, then r[*] is a SERE (finite consecutive repetitions)*
- *if r_1 and r_2 are SEREs, then the following are SEREs:*
  - *the consecutive concatenation of two sequences, $r_1;r_2$*
  - *one-state overlapping concatenation $r_1 : r_2$*
  - *disjunction of sequence $r_1|r_2$*
  - *overlapping sequences $r_1\&r_2$*
  - *length-matching sequence $r_1\&\&r_2$*

2.3 C-SRE Simulator

The proposed modeling technique and *online* monitoring are implemented in a tool named C-SRE. Figure 2.1 shows the C-SRE simulator framework.

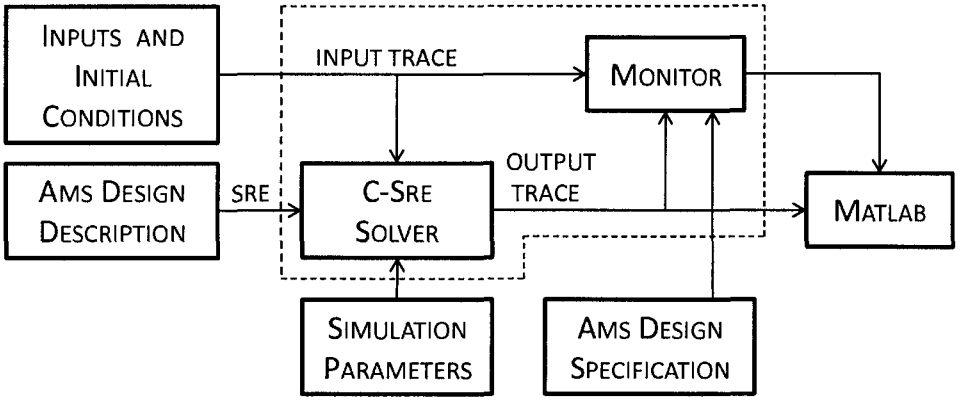


Figure 2.1: C-SRE Simulator Framework

The C-SRE tool solves a system of recurrence equations describing the behavior of an analog and mixed signal system. There are four main inputs to the tool. They are: (1) The AMS design behavior described using continuous-time (CT), discrete-time (DT) and discrete-events (DE) SRE notations; (2) PSL property monitors expressed in C language; (3) Various inputs and initial conditions to

the design; and (4) Simulation parameters such as minimum and maximum time step sizes, and simulation duration etc. The tool output contains the results of executing the monitor in an online fashion, along with various supporting signal traces for easy visualization of the results.

The C-SRE solver is the core of the simulator. It guarantees that the CT, DT or DE SREs are executed at an appropriate instant of time to simulate the correct transient behavior of the circuit. The scheduling algorithm is explained below: Let T_{CT}, T_{DT} and T_{DE} be the continuous time, discrete time, and discrete event time steps, respectively. If we assume that T_{CT} is always the smallest time step taken during the simulation, we can achieve both a desired time resolution and accuracy. T_{DT} is uniformly spaced in time and is known in advance. The size of T_{CT} and T_{DE} is determined dynamically during the simulation. In an AMS design, the continuous-time, discrete-time and discrete-event processes may interact with each other. The discrete-time part of the design only interacts at intervals of T_{DT} with the other parts. The simulation time advances by following four rules given below:

- If $T_{CT} = T_{DT}$ and $T_{CT} = T_{DE}$ then update the DE and DT SREs
- If $T_{CT} = T_{DT}$ and $T_{CT} < T_{DE}$ then update the DT SREs
- If $T_{CT} < T_{DT}$ and $T_{CT} = T_{DE}$ then update the DE SREs
- If $T_{CT} < T_{DT}$ and $T_{CT} < T_{DE}$ then update the CT SREs

where $t_{CT} = t_{CT} + T_{CT}, t_{DT} = t_{DT} + T_{DT}, t_{DE} = t_{DE} + T_{DE}$ and $T_{current} = MIN(t_{DT}, t_{CT}, t_{DE})$. Figure 2.2 illustrates sample time points at which continuous-time (circle), discrete-time (triangle), and discrete-event (square) SREs have to be executed so as to simulate the correct behavior of the system. The numbers in the figure show the sequence of operations. The discrete time steps (triangle) are equally spaced where as the continuous time (circle) and discrete event (square) time steps are determined dynamically during the simulation. The simulation starts with

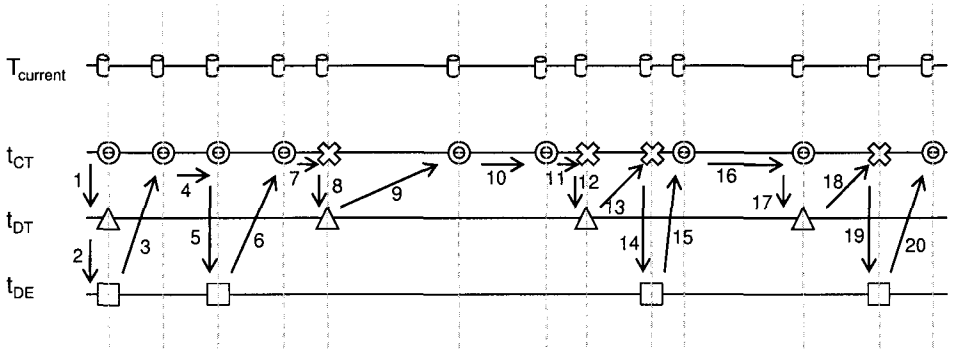


Figure 2.2: Timing Diagram

initialization and then proceeds guided by the scheduling algorithm. It terminates when the current simulation time ($T_{current}$) either exceeds or becomes equal to the maximum simulation time. The algorithm described above guarantees that SREs execute in proper sequence in order to simulate the correct behavior of the circuit. For a detailed description of the C-SRE simulator, please refer to [1].

2.4 Basic Concepts in Probability and Random Process

The basic definitions and concepts in probability and random process are briefly reviewed in this section. These concepts are essential for the understanding of statistical hypothesis testing method.

2.4.1 Random Variables

A random variable is a variable such that we do not know what specific value it will take on. We do know, however, the possible values it can assume and the probabilities of those values. There are two kinds of random variables: discrete random variables and continuous random variables. A *discrete random variable* can take on values from a finite or countably infinite set of numbers. Discrete

random variable arises in many applications involving counting. Most commonly used discrete random variable is the bernoulli random variable which is used to model the coin toss experiment. A *continuous random variable* can take on values from an interval of real numbers such as voltage, current or noise. Normal or gaussian random variable is the most commonly encountered contiuous random variable in both manmade and natural phenomena.

2.4.2 Distribution Functions

Distribution Functions for Discrete Random Variables

Let X be a discrete random variable and suppose that the possible value it can assume are $x_1, x_2, x_3, ...$ Suppose that these values have probabilities given by

$$f(x_i) = P(X = x_i) \qquad i = 1, 2, 3, ... \tag{2.2}$$

$f(x_i)$ is called the probability mass function (PMF). The cumulative distribution function, or CDF, for a discrete random variable X is defined as given by

$$F(x) = P(X \leq x) \tag{2.3}$$

where x is any real number, i.e. $-\infty < x < \infty$. The CDF for a given value a can be obtained from the PMF by

$$F(a) = P(X \leq a) = \sum_{x_i \leq a} f(x_i) \qquad i = 1, 2, 3, ... \tag{2.4}$$

Distribution Functions for Continuous Random Variables

The cumulative distribution function (CDF) for a continuous random variable X is defined by

$$F(x) = P(X \leq x) = P(-\infty < X \leq x) = \int_{-\infty}^{x} f(t)dt \tag{2.5}$$

It is a positive and monotonically increasing bounded function. The probability that a random variable X lies between the interval (a, b) is given by

$$P(a < X < b) = \int_{a}^{b} f(t)dt \tag{2.6}$$

The probability density function (PDF) for a continuous random variable is given by

$$f(x) = \frac{dF(x)}{dx} \tag{2.7}$$

and

$$f(x) \geq 0 \tag{2.8}$$

$$\int_{-\infty}^{\infty} f(x)dx = 1 \tag{2.9}$$

$F(x)$ is a positive bounded function.

2.4.3 Statistics

In statistics, we are interested in observing the behavior of a large group of objects and drawing conclusions based on our observation. The entire group is usually called a population. It can be finite, such as the final exam score of 500 students, or infinite, for example a study of the fairness of a particular coin, the population of all possible sequences of tosses of the coin is infinite. In practice, instead of investigating the entire group, which is difficult or impossible to do, we examine a small part of the population, which is usually called a sample.

2.5 Monte Carlo Simulation

Monte Carlo method originated in the 1940's [33]. It refers to a method of solving problems using random variables. It is widely used in the estimation of phenomena involving stochastic processes. One of the most important components of Monte Carlo simulation is the random number generator which generates random numbers without bias. The random numbers generated by computer softwares function are not truly random. They are generated based on a deterministic algorithm [27] and are sometimes called pseudo random numbers. The basic idea behind the Monte Carlo method is to sample the model of the true population of interest. This is followed by calculating the statistics of interest. The sampling and calculation procedure is repeated for M trials. The investigation of the distribution characteristics of the statistics is carried out based on those M experiments. When the Monte Carlo method is applied in hypothesis testing, we sample from a distribution which is known or assumed. The Monte Carlo hypothesis testing algorithm used for statistical runtime verification is described in detail in Chapter 3.

Chapter 3

Runtime Verification Methodology

In this chapter, we present the proposed runtime verification methodology for AMS designs. It consists of SRE modeling, functional runtime verification methodology and statistical runtime verification.

3.1 Overall Methodology

The proposed methodology contains three stages: the modeling stage, the functional runtime verification stage, and the statistical runtime verification stage. As shown in Figure 3.1, in the modeling stage the AMS design is modeled using SRE notations and delivered to the C-SRE simulator. In the functional verification stage, the functional properties derived from the AMS system specifications are verified using *online* runtime verification. The statistical properties which summarize the stochastic behavior of the AMS design is performed in statistical verification stage. The AMS system with stochastic process is modeled using SRE and simulated in C-SRE simulator as well. The functional runtime verification is performed using *online* monitoring technique and the monitor is implemented in C programming language incorporated with the C-SRE simulator. The statistical runtime verification is carried out in an *offline* fashion as all the information from the simulation

trace is needed. The statistical monitor is implemented in Matlab [32] environment.

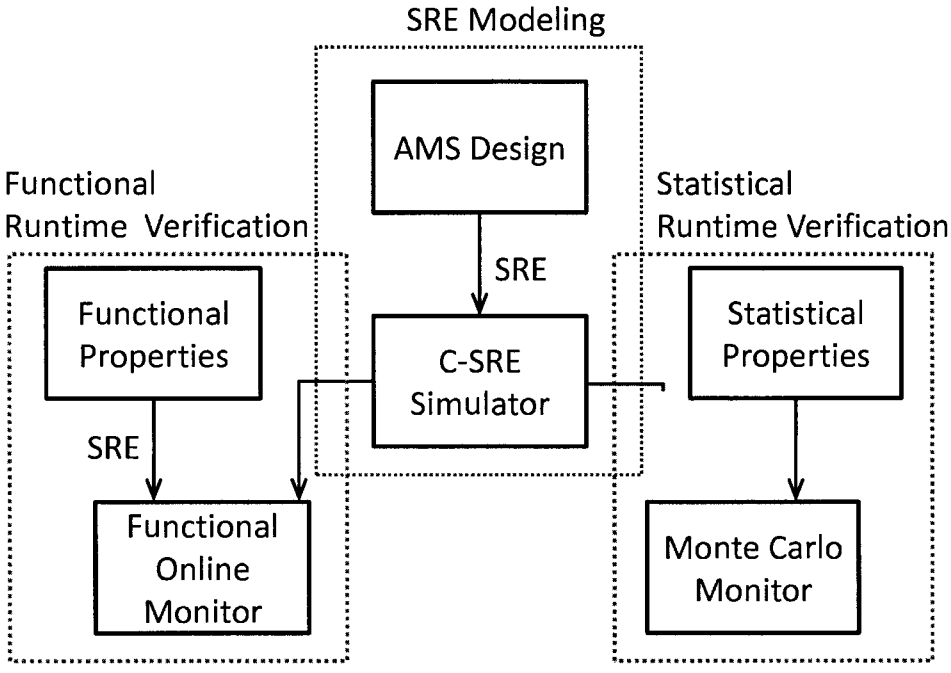


Figure 3.1: Overall Methodology

In the following sections, we describe the SRE modeling, the online runtime verification and the statistical verification, respectively.

3.2 SRE Modeling of AMS Design

The modeling stage is the first step of the proposed runtime verification methodology. The AMS system is modeled using a system of recurrence equations (SRE). In this section, we describe how AMS systems can be modeled using SRE.

SRE is used to model the system at a high level abstraction. The SRE modeling procedure of AMS designs usually begins with the mathematical model of the system. An AMS design is usually a complex mixture of pure analog, pure digital or mixed signal components. For digital blocks, the SRE can be generated based

on their logic function. The logic function can be expressed as difference equations which are already SRE (Definition 2). For analog components, we have two potential options to generate SRE. First, we can write the recurrence equation based on time domain differential algebraic equations (DAEs) through a discretization (i.e., in discrete form). However, most analog components are expressed using transfer function in frequency domain. The second option allows the use of the transfer function. We use Impulse-Invariant z transformation [6], which is a frequency transformation of discrete-time signal, to find z domain [38] approximation of the s domain transfer function of an analog component. Then, we apply the inverse of z transform to the z domain transfer function to generate time domain difference equation and convert it into equivalent SRE model. For mixed signal components, the input and output relation can be expressed using SRE notations. At the end of modeling we are left with a system of recurrence equations (SREs) which accurately describe the behavior of the system.

The SREs are then the input to the C-SRE simulator introduced. The sequence of the SREs has to be exactly the same as that of the original system. The outputs of each component of the AMS design can be conveniently plotted using a graphic interface such as Matlab [32]. The detailed examples of modeling the AMS design are illustrated in the case study of Chapter 4.

3.3 Runtime Verification with Online Monitoring

In functional runtime verification stage, the functional properties of AMS system are verified in an *online* fashion. We first use PSL to formulate the property of the AMS system. The PSL expression is then translated into SRE notation to construct a monitor. The consistency of the monitor and the modeling allows us to achieve *online* monitoring. In this section, we first describe how to convert PSL properties into SRE notations, followed by the functional runtime verification methodology.

3.3.1 Writing PSL using SREs

As an assertion language, PSL contains four layers [12]: Boolean, temporal, verification and modeling layer. The verification layer provides the communication and interaction between the property and the verification tool. The modeling layer is used to define the verification environment for the tool. The Boolean layer constructs the basic expressions for the property. The temporal layer, where the temporal relations between the signals are expressed, is the heart of the PSL. In this thesis, we consider the Boolean layer and the temporal layer only. In following, we will describe how to write PSL properties using SRE in terms of the Boolean layer and the temporal layer, respectively.

Boolean Layer

The Boolean layer specifies expressions of the design and associated signals which evaluate in a single cycle. The evaluation result is either true or false. In PSL the analog description to a Boolean variable is an inequation which is built using signals and registers of the AMS design [3]. This expression is defined as the *Basic Property* [3]:

Definition 4 ***Basic Property***

Let x be the name of an AMS signal (or register), a basic property p is a logical formula defined as follows: $p = x \diamond y$, where $\diamond \in \{<, \leq, >, \geq, =, \neq\}$ and y is a value, a name of a signal (or a register) in the design or an arithmetic function built using the design signals.

The Boolean expression can be written in SRE according to the logic it involves. Suppose that `a` and `b` are both basic properties, the expressions `a -> b` and `a <-> b` produce Boolean results, true or false. The expression `a -> b` can be expressed in SRE as

```
IF{a, IF{b, true, false}, true}
```

and the SRE expression of `a <-> b` is given by

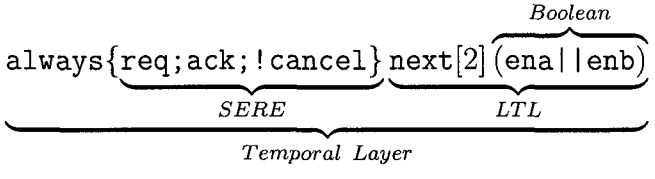
```
IF{a, IF{b, true, false}, IF{b, false, true}}
```

The two SREs employ the nested form of the generalized `If-formula`.

Temporal Layer

The Temporal Layer is used to specify temporal chains of events of Boolean expressions. In AMS design, these Boolean expressions are replaced with basic properties. The temporal layer consists of the Foundation Language (FL) and the Optional Branching Extension (OBE). FL is a linear temporal logic which embeds a customized version of Regular Expression, called the Sequential Extended Regular Expression (SERE). OBE is a class of the computational tree logic (CTL) [15] language. FL and OBE cannot be mixed in PSL property. In this thesis, we focus on SERE and FL expressions only.

For example, the PSL property shown below is in the temporal layer. It contains SERE and Boolean expression as well as LTL style property. In order to translate such complex PSL into SRE, we have to translate each sub-properties and join them together based on the temporal operator.



The SERE concatenation operator (;) constructs an SERE that is the concatenation of two other SEREs. The property `req;ack` holds tightly on a path if and only if there is a future cycle n, such that `req` holds tightly on the path up to and including the n^{th} cycle and `ack` holds tightly on the path starting at the $n+1^{th}$ cycle. In order to write SERE expression in SRE, cycle should be the whole time

cycle space (i.e., for all n). The SRE notation for the property `req;ack` can be given by

$$\texttt{IF\{}\prod_{i=0}^{n}\texttt{req(n), IF\{ack(n+1), true, false\}, false\}}$$

FL properties describe single or multi-cycle behavior built from Boolean expressions, sequential expressions, and subordinate properties. The most basic FL Property is a Boolean expression. An FL Property enclosed in parentheses (), as opposite to SERE that is identified using curly braces {}. The compound FL properties can be converted into SRE notation by nesting the operands of the FL property according to the temporal logic operator. In the example above, `next[2](ena||enb)` is an FL property. The *next* operator is in weak semantics which means that the Boolean expression is not required to happen at the next second cycle on the path. The property holds in two cases: (1) the path ends before the second cycle ; (2) the operand `(ena||enb)`, which is a basic FL property, holds at the next second cycle. Suppose that the path length is denoted by N and the current cycle is denoted by n. The SRE of this FL property can be described as

$$\texttt{IF\{n+2>N, true, IF\{ena(n+2)=1}\vee\texttt{enb(n+2)=1, true, false\}\}}$$

The SRE notation can be implemented using if-else statement in any programming language such C/C++, Matlab or VHDL. The transformation from PSL to SRE allows us to achieve the *online* runtime verification for AMS designs.

3.3.2 Online Monitoring

The verification flow of the *online* monitoring of an AMS design is shown in Figure 3.2. The AMS design is modeled using SRE based on the circuit description and simulated using the C-SRE simulator. Design properties are formally expressed in PSL. The PSL expression is then converted to the SRE notation. Finally, the input

stimulus and output traces are delivered to the monitor. The monitor evaluates the inputs and outputs of the simulator and checks whether the behavior satisfies the design specification. The monitoring is performed in an *online* fashion which means if the property is satisfied, the monitor reports the satisfaction; otherwise, the monitor terminates the simulation at the cycle when the violation occurs.

The input stimulus includes the input signal and environment such as control signals. The output trace of the simulator can be either from any observation point or from any component within the system. The AMS specifications we focus on are written in temporal logic. The evaluation of the relation between input and output with the design specification is carried out within the monitor.

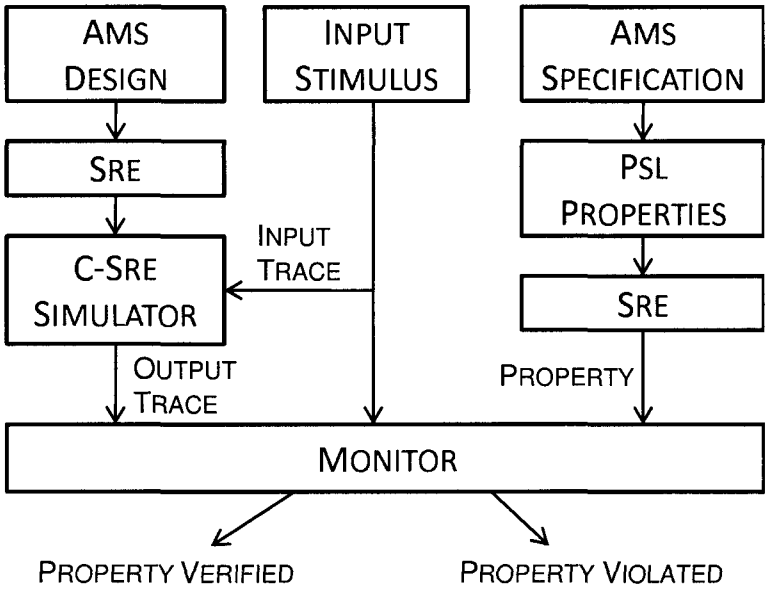


Figure 3.2: Online Runtime Verification

The monitor is used to check whether the current simulation behavior satisfies a given property of correctness. With the help of Basic Property (Definition 3), the properties of AMS design can be expressed properly in PSL. The properties are then translated into recurrence sequence notations. In our methodology, the input and output traces are available to the monitor at each simulation time instant.

Incorporated with the property checker (monitor) described above, at each time instant of the simulation, the violation of the property is also checked. The process is carried out as long as no violation is detected within simulation trace. Moreover, by taking advantage of the C-SRE simulator, which records all the transient data of all circuit blocks at runtime, the monitor is able to observe the property of individual block. This is useful when we want to verify the functionality of single component in a complex design. This allows us to check the interface between two components and the complex effects of one component brought by others. These two advantages enable us to verify large system with many blocks in a realistic environment.

There are two categories for PSL properties: *safety property* and *liveness property*. A safety property claims that something bad will never happen. For example, the property "after the frequency select signal *Freq_sel* is activated, the PLL will lock at the desired frequency within the lock time 1.5ms" is a safety property. This property verifies that the PLL will lock within 1.5ms after the activation of *Freq_sel*. If the PLL cannot lock within this period or it locks at a different frequency, the property fails. This safety property guarantees that the PLL system functions correctly. A liveness property claims that something good will eventually happen. For example, the property "after the frequency select signal *Freq_sel* is activated, the PLL will lock sometime in the future" is a liveness property. It is expected that the PLL will lock eventually after the activation of *Freq_sel*.

The temporal operator in PSL can be either strong or weak. A strong temporal operator is denoted by concatenating an exclamation point (!) to the ordinary operator. Examples of strong temporal operator are `eventually!` and `next!`. The operator without an exclamation point such as `next` is weak. Both strong and weak operators have the terminating condition . The terminating condition is a Boolean expression and the occurrence of it causes the property to complete. For example, the terminating condition of the property `always(a -> next[3] b)` is that signal `b` holds. There is no requirement about the terminating condition for

the weak operator, while the strong operator requires that the terminating condition eventually happen. The PSL property using a weak temporal operator will hold as long as nothing else has gone wrong. For example, in the case that the simulation path ends before three cycles after signal `a` holds, the property `a -> next[3] b` will holds because if we keep run the simulation for a few more cycles, the terminating condition might happen. On the other hand, in the case that the simulation path is not long enough, the property using a strong temporal operator will not hold, even if nothing else has gone wrong. Two properties `always(a -> next[3] b)` and `always(a -> next![3] b)` have different results when they are applied to the same simulation path shown in Figure 3.3. The terminating condition happens once at cycle 4 and the simulation path ends at 10. The property using weak operator holds because there is no requirement for the termination condition to happen again at cycle 11. However, the property using strong operator fails to hold. The reason is that the property requires that the terminating condition occur based on the simulation path shown in Figure 3.3 and does not consider what will happen after the cycle 10. The difference between weak and strong operators is important when the simulation path is "too short". For *offline* monitoring, the strong operator is difficult to achieve because we have to manually change the simulation path to make sure that it is long enough for the occurrence of the terminating condition. On the other hand, *online* monitoring is able to fully support the strong operator because the simulation keeps running until the violation is detected or the terminating condition of the property occurs.

Table 3.1 compares *online* and *offline* monitoring methods in terms of their support to safety and liveness properties and strong and weak semantics. In summary, the *online* monitoring is not only a good complement to *offline* as shown in Table 3.1, but also it can save the computational resource in terms of simulation time compared to *offline*. We will see more examples of *online* monitoring in the following chapter and the comparison with *offline* method in terms of simulation

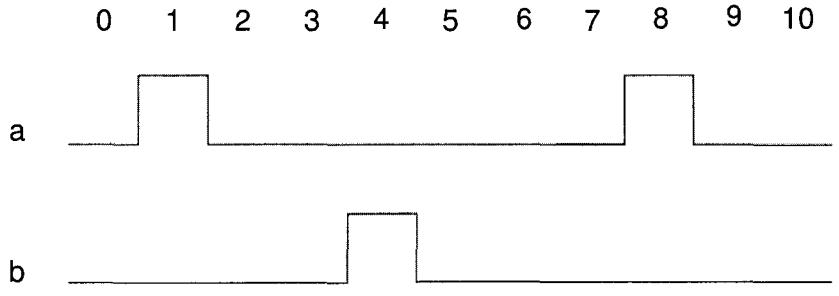


Figure 3.3: Online Runtime Verification

time and memory usage.

Table 3.1: Basic Comparison of Online Method and Offline Method

	Online monitoring	Offline monitoring
Safety property	Yes	Yes
Liveness property	Yes	Yes
Strong semantics	Yes	No
Weak semantics	Yes	Yes

3.4 Statistical Runtime Verification

In this section, we will present the statistical runtime verification methodology using Monte Carlo monitoring . The monitor is constructed to perform statistical hypothesis testing using Monte Carlo simulation. We first introduce statistical hypothesis test and the theory behind it. Then we describe how Monte Carlo method is applied to the statistical hypothesis test and to the evaluation of the hypothesis test performance. Finally we present the runtime verification methodology for the verification of statistical properties of AMS designs.

3.4.1 Hypothesis Testing

Statistical Hypothesis Testing

Statistical hypothesis testing is a technique which provides a decision making procedure about logic statements based on statistical information. The conclusion is drawn with a confidence level and an error estimate. Hypothesis testing is generally formulated in two parts. They are *null hypothesis*, denoted by H_0, which is what we want to test and *alternative hypothesis*, denoted by H_1, which is what we want to test against the null hypothesis. If we reject H_0 based on our statistical investigation, then the decision to accept H_1 is made. For example, we want to determine whether there is a difference in quality between two products A and B. The null hypothesis might be that there is no difference between the A and B. Then the alternative hypothesis might be that there is a difference (i.e., that one is better in quality than the other). The steps in statistical hypothesis testing are listed below

1. Determine the null and the alternative hypotheses.
2. Take a random sample from the population of interest.
3. Calculate a statistic from the sample that provides information about the null hypothesis.
4. If the value of the statistic is consistent with H_0, then accept H_0.
5. If the value of the statistic is inconsistent with H_0, then reject H_0 and accept H_1.

Error Bounds

There are two kinds of error bounds that apply when we are making a decision in statistical hypothesis testing. They are known as Type I error and Type II error [31]. A Type I error, or false positive, occurs when we reject H_0 which is actually true. A

Type II error, or false negative, arises when we accept H_0 which is actually false. α and β denote the probability of Type I error and Type II error respectively. Formally,

$$\alpha = Pr\{\texttt{reject } H_0 | H_0 \texttt{ is true }\}$$

$$\beta = Pr\{\texttt{accept } H_0 | H_0 \texttt{ is false}\}$$

Typically, α is the maximum probability of Type I error tolerated. In hypothesis testing, we are looking for significant evidence that the null hypothesis H_0 is false, namely that H_1 is valid. In order to avoid changing decision status unless there exists sufficient evidence guiding us, the probability of incorrectly rejecting H_0, namely Type I error α, is expected to be controlled.

Confidence Level

As defined, the Type I error is the probability of rejecting null hypothesis H_0 while it is true. In other words, it means the likeliness that we accept alternative hypothesis H_1 when H_0 is true. The confidence is drawn according to the compliment of the Type I error α. α is also called *significance level.* Formally, the confidence level δ is give by:

$$\delta = 1 - \alpha \tag{3.1}$$

For instance, $\alpha = 0.05$ refers to the confidence level of 95% and $\alpha = 0.01$ refers to the confidence level of 99%. Before performing the hypothesis testing, the Type I error (i.e., the confidence level) should be established. The reason is that in a hypothesis test we are looking for the significant evidence to reject null hypothesis H_0 (i.e., H_0 is false) and the probability that H_0 is true needs to be controlled.

Tail Test

In order to determine whether or not the observed statistic is consistent with H_0, we should know the distribution of the statistic under the condition that H_0 is true.

A *rejection region* is needed to perform the statistical hypothesis test. A rejection region, over which we would reject H_0, is the area covered by the PDF. The *critical value* is used to divide the domain of the test statistic into a rejection region and a non-rejection region. The rejection region depends on the distribution of the statistic under H_0, H_1 and the Type I error we are willing to tolerate. Generally, the rejection region is located at the tails of the distribution of the test statistic when H_0 is true. The test can take place either in the lower tail or the upper tail which depends on the alternative hypothesis H_1.

(a) ***Upper tail test***: If a large value of the test statistic would provide evidence for rejecting H_0, then the rejection region is in the upper tail of the distribution of the test statistic.

(b) ***Lower tail test***: If a small value of the test statistic would provide evidence for rejecting H_0, then the rejection region is in the lower tail of the distribution of the test statistic.

Hypothesis Testing Algorithm

There are several approaches for hypothesis testing. In this thesis, we address only the critical value approach. It is also important to note that in order to perform the hypothesis test, the distribution of the test statistic under the null hypothesis H_0 is assumed to be known. Before we describe the critical value approach, we first introduce an important concept, namely quantile function.

Quantile Function

Quantile function plays an important role in statistics [31]. The quantile q_p of a random variable X is defined as the smallest number q such that the cumulative distribution function (CDF) $P(X)$ is greater than or equal to some p, where $0 < p < 1$. This can be calculated for a continuous random variable with probability

density function $f(x)$ by solving

$$p = \int_{-\infty}^{q_p} f(x)dx \tag{3.2}$$

for q_p. The quantile function is the inverse of the cumulative distribution function (CDF) and is given by

$$q_p = quantile(p) = F^{-1}(p) \tag{3.3}$$

The p-th quantile of a random variable X is the value q_p such that

$$F(q_p) = P(X < q_p) = p \tag{3.4}$$

In general, the quantile function is the inverse of cumulative distribution function (CDF). In hypothesis test, we will see that quantile function is used to determine the decision about rejection of a hypothesis.

Critical Value Approach

In hypothesis testing, if the observed statistic is within some region, we reject the null hypothesis. The interval where the null hypothesis is rejected is called *critical region*, or *rejection region*. The *critical value* is used to divide the domain of the test statistic into rejection region and non-rejection region. The critical value approach is used to check whether the observed value falls into the rejection region. The procedure of this approach is outlined in Algorithm 1. The critical value approach requires α and T_{obs} to perform. α is the significant level, or Type I error, and T_{obs} is the observed value calculated by

$$T_{obs} = \frac{\bar{x} - \mu_0}{\bar{\sigma}} \tag{3.5}$$

where $\bar{x}$ is the sample mean of the random variable, μ_0 is the mean value under the null hypothesis and $\bar{\sigma} = \sigma_x/\sqrt{n}$ is the standard error of the sample. The algorithm is performed in two cases: upper tail test and lower tail test. In the case of upper tail

test (from line 1 to line 8), we first calculate the critical value using quantile function and the significance level $1 - \alpha$ (line 2). The hypothesis test is performed from line 3 to line 7. For upper tail test, we are looking for large significant evidence to reject the null hypothesis H_0. If the observed value T_{obs} is greater than the critical value, then we reject H_0; otherwise, the decision of accepting H_0 is made. The procedure of lower tail test (from line 9 to line 16) is similar except that the critical value, i.e,. the rejection region, is different. The critical value is calculated based on the significance level α. In this case, if T_{obs} is smaller than the critical value, then we reject H_0; otherwise, we accept H_0.

Algorithm 1 Hypothesis Testing- Critical Value Approach

Require: α, T_{obs}

1: **while** *Upper Tail Test* **do**
2: $critical\_value = quantile(1 - \alpha)$
3: **if** $T_{obs} > critical\_value$ **then**
4: *Reject* H_0
5: **else if** $T_{obs} < critical\_value$ **then**
6: *Accept* H_0
7: **end if**
8: **end while**
9: **while** *Lower Tail Test* **do**
10: $critical\_value = quantile(\alpha)$
11: **if** $T_{obs} < critical\_value$ **then**
12: *Reject* H_0
13: **else if** $T_{obs} > critical\_value$ **then**
14: *Accept* H_0
15: **end if**
16: **end while**

The probability of wrongly rejecting H_0, or Type I error, is supposed to be controlled before we perform a hypothesis test. The critical value depends on the significance level α, namely the Type I error. The typical values of α are $0.01, 0.05$, and 0.10. The critical value is found as a quantile (under the null hypothesis H_0) calculated using Equation 3.3. In the case of the lower tail test the significant value

is α, while $1-\alpha$ is chosen when we perform the upper tail test. For upper tail test, the large significant evidence is investigated. In other words, if the observed value T_{obs} is greater than the critical value we reject the null hypothesis H_0. Otherwise, we retain H_0. For lower tail test, a small value is needed as the evidence to reject H_0. Hence, if T_{obs} is less than the critical value, calculated using α in this case, we reject H_0. Otherwise we retain H_0.

For example, we are looking for the significant level and the rejection region of a normal distribution based random variables with zero mean ($\mu = 0$) and unit standard deviation ($\sigma = 1$). The Type I error we are willing to tolerate is 0.05. The rejection region (shaded area) of a lower tail test is shown in Figure 3.4. The critical value can be calculated as -1.645 using Equation 3.3. If the observed value (normalized value) is less than -1.645 (i.e., if it falls in the rejection region), then H_0 is rejected; while if the observed value is greater than the critical level (i.e., it appears in the non-rejection region), we fail to reject H_0. As a result, the decision to accept H_1 is reached.

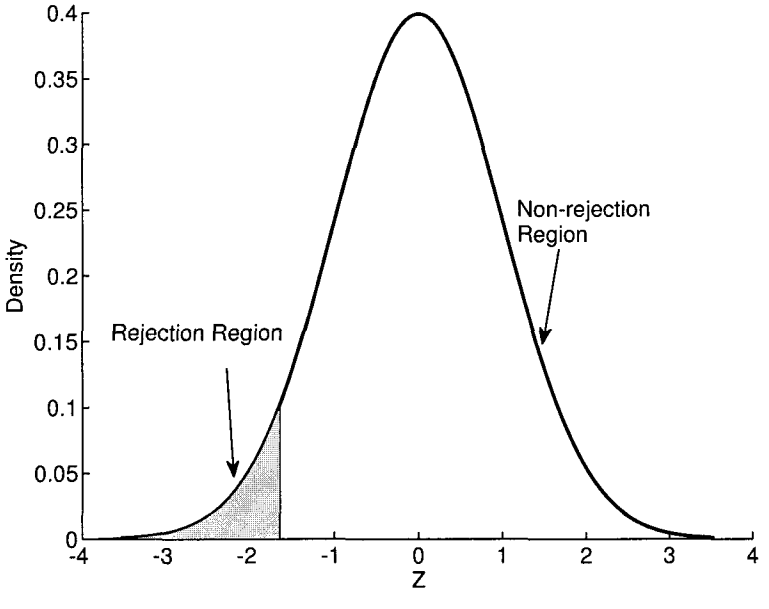


Figure 3.4: Rejection Region for a Lower Tail Test

Error Margin

For random variables, it is unlikely that the observed value of the sample is exactly equal the true value of the population parameter such as the mean or the variance. Hence, it is more useful to have an interval of numbers that contains the true value. The probability of the true value appearing in the interval is the confidence level δ we introduced previously. Let θ represent a population parameter that we wish to estimate. The observed value of the statistic is denoted as $\hat{\theta}$. An interval of θ can be expressed as

$$\hat{\theta}_{low} < \theta < \hat{\theta}_{up} \tag{3.6}$$

where $\hat{\theta}_{low}$ and $\hat{\theta}_{up}$ are the lower and upper bound of the interval respectively. Given the confidence level δ, we have

$$P(\hat{\theta}_{low} < \theta < \hat{\theta}_{up}) = \delta = 1 - \alpha \tag{3.7}$$

which indicates that the probability that θ present itself in the interval produced by a random sample is δ. Recall Equation 3.4, we can use quantile function to calculate the critical value for a specific probability. We have

$$P(X < q_{(1-\alpha/2)}) = 1 - \alpha/2 \tag{3.8}$$

$$P(X < q_{(\alpha/2)}) = \alpha/2 \tag{3.9}$$

Subtracting Equation 3.9 from 3.8, we achieve

$$P(q_{(\alpha/2)} < X < q_{(1-\alpha/2)}) = 1 - \alpha \tag{3.10}$$

where X is the normalized random variable. Because the quantile is calculated in normalized form, we normalize θ using the sample mean $\bar{X}$, the population mean μ

and standard error $\sigma/\sqrt{N}$, then we have

$$P(q_{(\alpha/2)} < \frac{\bar{X} - \mu}{\sigma/\sqrt{N}} < q_{(1-\alpha/2)}) = 1 - \alpha \tag{3.11}$$

Rearranging Eequation 3.11, we obtain

$$P(\bar{X} - \frac{q_{(1-\alpha/2)}\sigma}{\sqrt{N}} < \mu < \bar{X} - \frac{q_{(\alpha/2)}\sigma}{\sqrt{N}}) = 1 - \alpha \tag{3.12}$$

Eventually, we obtain the confidence interval

$$(\bar{X} - \frac{q_{(1-\alpha/2)}\sigma}{\sqrt{N}}, \bar{X} - \frac{q_{(\alpha/2)}\sigma}{\sqrt{N}}) \tag{3.13}$$

It is not difficult to see that $q_{(1-\alpha/2)} = -q_{(\alpha/2)}$. We define the error margin ϵ as

$$\epsilon = q_{(1-\alpha/2)}\frac{\sigma}{\sqrt{N}} \tag{3.14}$$

which depends on the confidence level δ and standard deviation σ of the population.

The error margin is provided together with the confidence level when the hypothesis test is done. The confidence level indicates the Type I error, which we establish in advance, of the statistical hypothesis testing. The error margin provides the confidence interval that contains the population parameter we want to estimate. The larger the confidence interval which incloses the population parameter is, the higher confidence level we can achieve.

3.4.2 Monte Carlo Methods for Hypothesis Test

In order to perform Monte Carlo monitoring, the distribution of the population is supposed to be known in advance or is assumed. Then a model which reflects the characteristics of the original population is made. The Monte Carlo simulation is used to generate random sample for estimation of the distribution of the original

population. Hypothesis testing is then used to check whether observed value of test statistic falls into the reject region specified by the estimated critical value.

The detailed procedure is illustrated in Algorithm 2 where T_{obs} is observed value calculated using Equation 3.5, n is the sample size, σ is the population standard deviation, $\bar{\sigma} = \sigma/\sqrt{n}$ is the standard error of the sample, and μ denotes the population mean. The loop between line 1 and line 5 is the Monte Carlo simulation repeated for M trials. In each trial, we randomly sample from the distribution of population under the null hypothesis with the same sample size n (line 2 and line 3) and then calculate and record the observed value of this pseudo sample T_{mc} (line 4) which is given by

$$T_{mc} = \frac{\bar{s} - \mu}{\bar{\sigma}} \tag{3.15}$$

where $\bar{s}$ is the mean value of the pseudo random sample. It is important to note that all the calculations till now are under the hypothesis that H_0 is true. The hypothesis testing is performed afterwards (from line 6 to line 21). The test is very similar to the original one (Algorithm 1) except that the calculation of the critical value. In Monte Carlo hypothesis testing, the critical value is estimated based on the model generated by Monte Carlo simulation. Whereas the critical value in Algorithm 1 is based on the standard normal distribution. The hypothesis test is carried out using the estimated critical value and the observed value T_{obs} the same way that Algorithm 1 does. Because for each hypothesis test the Monte Carlo simulation generates different random model, it is expected that the estimated critical value varies for each test.

3.4.3 Hypothesis Test Performance Assessment

When we use Monte Carlo simulation, we make assumption about the distribution of the sample. However, when the assumption is not correct, we need to assess the results. Monte Carlo method can also be used to evaluate the performance

Algorithm 2 Monte Carlo Hypothesis Testing

Require: α, T_{obs}, n, σ, $\bar{\sigma}$, μ

1: **for** $i = 1$ to M do **do**
2: $r = random\_number\_generator(n)$
3: $s = \sigma \cdot r + \mu$
4: $T_{mc}(i) = (mean(s) - \mu)/\bar{\sigma}$
5: **end for**
6: **while** *Upper Tail Test* **do**
7: $critical\_value = quantile(T_{mc}, 1 - \alpha)$
8: **if** $T_{obs} > critical\_value$ **then**
9: *Reject* H_0
10: **else if** $T_{obs} < critical\_value$ **then**
11: *Accept* H_0
12: **end if**
13: **end while**
14: **while** *Lower Tail Test* **do**
15: $critical\_value = quantile(T_{mc}, \alpha)$
16: **if** $T_{obs} < critical\_value$ **then**
17: *Reject* H_0
18: **else if** $T_{obs} > critical\_value$ **then**
19: *Accept* H_0
20: **end if**
21: **end while**

for hypothesis test. We choose Type I error as the reference for the performance evaluation. The reason is that it is established before the hypothesis test procedure. While the Type II error is according to the alternative hypothesis and can not be a reference for the performance.

Algorithm 3 indicates the procedure of the assessment in terms of the Type I error. Like critical value approach, the performance assessment is performed in two cases as well. We first set up a counter I (line 1). In the case of upper tail test, the critical value associated to the standard Type I error is then calculated using quantile function (line 2). The Monte Carlo simulation is carried out (from line 4 to line 10). In each Monte Carlo trial, we generate pseudo random sample s of size n under the null hypothesis (line 5 and line 6) as what we do in the Algorithm 2. The observed value of the pseudo sample T_{mc} is calculated using Equation 3.15 as

Algorithm 3 Performance Assessment of Monte Carlo Hypothesis Test

Require: n, σ, $\bar{\sigma}$, μ, α

$I = 0$
while *Upper Tail Test* **do**
  $critical\_value = quantile(1 - \alpha)$
  for $i = 1$ to M do **do**
    $r = random\_number\_generator(n)$
    $s = \sigma \cdot r + \mu$
    $T_{mc}(i) = (mean(s) - \mu)/\bar{\sigma}$
    if $T_{mc} \geq critical\_value$ **then**
      $I = I + 1$
    end if
  end for
end while
while *Lower Tail Test* **do**
  $critical\_value = quantile(\alpha)$
  for $i = 1$ to M do **do**
    $r = random\_number\_generator(n)$
    $s = \sigma \cdot r + \mu$
    $T_{mc}(i) = (mean(s) - \mu)/\bar{\sigma}$
    if $T_{mc} \leq critical\_value$ **then**
      $I = I + 1$
    end if
  end for
end while
$\hat{\alpha} = I/M$

well. Then we need to determine whether a Type I error has been committed in hypothesis test. In other words, the null hypothesis, which should not be rejected when H_0 is true, has been rejected. If the pseudo observed value T_{mc} falls into the rejection region, the counter I increases its value by one; otherwise, I remains its value. The procedure for lower tail test is similar except of the condition for specifying the commitment of Type I error. Repeat this procedure for M trials. Then the probability of making an actual Type I error when using Monte Carlo simulation is given by

$$\hat{\alpha} = \frac{1}{M} \sum_{i=1}^{M} I_i \tag{3.16}$$

The performance indicator of hypothesis testing, denoted as η, is defined as the deference between a specific significant level α and Type I error concluded using algorithm 3 $\hat{\alpha}$ and it should be a positive quantity. Then we have

$$\eta = |\hat{\alpha} - \alpha| \tag{3.17}$$

The performance indicator η does not affect the hypothesis results. It provides, however, the level of performance of the Monte Carlo simulation to the hypothesis testing. Smaller η indicates that the estimated distribution is very close to the real one.

3.4.4 Hypothesis Test Summary

So far, we have introduced the Monte Carlo hypothesis test algorithm. Before coming to statistical runtime verification methodology, we summarize the important concepts in Table 3.2. The first column lists the names of the concept. The descriptions and the mathematical equations are illustrated in the second and the third column, respectively.

Table 3.2: Summary of Hypothesis Test Concepts

Concept	**Description**	**Formula**
Quantile function (*quantile*)	Inverse CDF function	q_p=$quantile(p)$=$F^{-1}(p)$
Significance level (α)	Type I error	p=$\{\text{reject}H_0 \mid H_0\text{is true}\}$
Critical value (cv)	Divides rejection region and non-rejection region	cv=$quantile(\alpha)$ cv=$quantile(1-\alpha)$
Confidence level (δ)	Significance level α	δ=$1-\alpha$
Error margin (ϵ)	Confidence interval for observed value	ϵ=$q_{(1-\alpha/2)}\frac{\sigma}{\sqrt{N}}$
Performance indicator (η)	Performance assessment	η=$\lvert\hat{\alpha}-\alpha\rvert$

3.4.5 Statistical Runtime Verification

As shown in Figure 3.5, the statistical property, such as mean or variance, we want to verify is expressed as a null hypothesis H_0. The alternative hypothesis H_1 becomes the counterexample naturally. The Monte Carlo monitoring is then carried out based on the confidence level δ we specify. The decision is made based on the significance level α with respect to the confidence level δ. In Monte Carlo Monitor, the property is verified using hypothesis test incorporated with Monte Carlo simulation. The statistical property is verified if the decision of accepting the null hypothesis H_0 is made. The rejection of null hypothesis H_0 leads to the violation of the property. All the decisions are produced under the specific confidence level δ along with the error margin ϵ. The error margin specifies a confidence interval where the estimated statistic falls with the probability of δ.

The Monte Carlo simulation is then employed to evaluate the performance of the hypothesis test in terms of the Type I error. The difference between the significance level α and the actual Type I error $\hat{\alpha}$ committed during the procedure provides a performance indicator for the hypothesis testing.

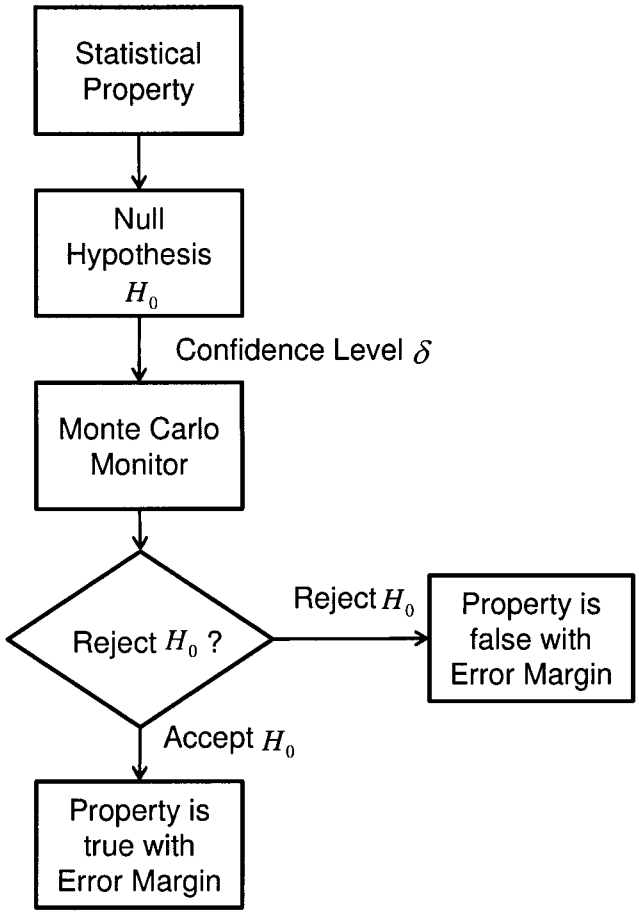


Figure 3.5: Monte Carlo Based Statistical Runtime Verification

3.5 Summary

The proposed runtime verification methodology contains the modeling stage, the functional runtime verification stage, and the statistical runtime verification stage. We employ SRE expressions to model the AMS system. All the SRE descriptions are simulated using the C-SRE simulator. The functional runtime verification is carried out in an *online* fashion. It allows the simulator to terminate execution at the moment a violation or satisfaction is determined. We also proposed a statistical runtime verification for the AMS system with stochastic process. The combination of Monte Carlo simulation and statistical hypothesis testing enriches us to analyze the random process without any knowledge about the distribution function of the statistic and to evaluate the performance of the verification process. The confidence level and error margin are provided along with the verification results. The two runtime verification methods are implemented in two different monitors. The functional *online* monitor is constructed by translating the PSL expression of the properties into SRE notations. The Monte Carlo monitor is implemented using statistical functions in the Matlab environment [32].

In the following chapter, we apply our methodology on a typical AMS system, a PLL based frequency synthesizer, as case study. Each stage of the proposed methodology is described in details. Several interesting functional and statistical properties are analyzed and verified. The comparison of the *online* monitoring and the *offline* monitoring is illustrated. In addition, we will discuss the performance of the Monte Carlo monitoring technique.

Chapter 4

Case Study: PLL Based Frequency Synthesizer

In this chapter, we will apply the runtime verification methodology proposed in this thesis on an important AMS design, the PLL based frequency synthesizer. We first present the SRE modeling of the system. We then describe several interesting functional properties in PSL and illustrate the functional verification results. For statistical verification, we present an introduction to SRE modeling of jitter property followed by the experimental results.

4.1 SRE Modeling

The PLL based frequency synthesizer shown in Figure 4.1 is an important AMS design for communication systems. It is used to generate a certain range of frequency. In this case study, we use a simple frequency synthesizer, which is able to generate a signal with the frequency as twice as that of reference signal, to demonstrate the proposed methodology. In the frequency synthesizer, the reference signal is a sinusoid signal with the reference frequency ω_0. The VCO output is a *Cosine wave* signal with frequency $N+1$ times of ω_0. N is determined by the frequency control

signal *Freq_sel*. If *Freq_sel* is set to 0, the frequency of the reference input and VCO output will be the same.

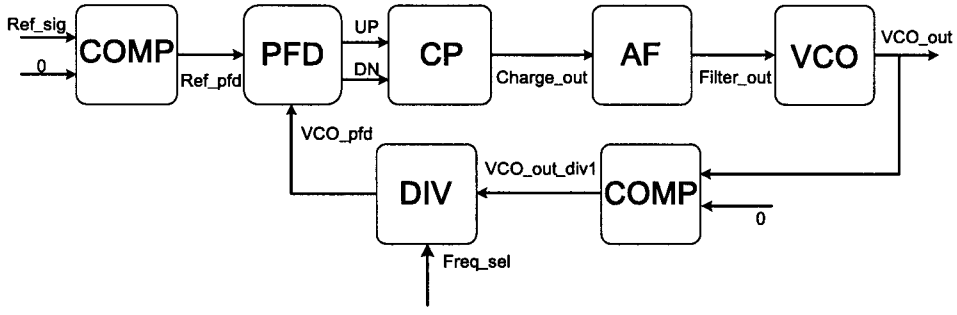


Figure 4.1: PLL Frequency Synthesizer Architecture

In the following, we present the SRE modeling of the PLL. We first illustrate the high level description of each block in detail. Then we will show how to model those behaviors using SRE.

Reference Signal Comparator

The comparator extracts the positive value of the input signal and generates binary sequence with the same frequency and phase as the input. The comparator is a mixed signal component as the input is analog signal and the output is digital. The SRE model of a comparator is given by

$$\texttt{Ref\_pfd(n+1)} = \texttt{IF}\{\texttt{Ref\_sig(n)} \geq 0,\ 1,\ 0\}$$

$$\texttt{Ref\_sig(n)} = \sin(\omega_0 \texttt{n} T\texttt{0})$$

where *Ref_sig* is one of the inputs to the comparator.

Phase and Frequency Detector

The implementation of a phase and frequency detector (PFD) is shown in Figure 4.2. It is composed of two edge-triggered, resetable D flipflops with their D inputs connected to V_{DD} (i.e. logical one). It is a pure digital block in the PLL system. Two

input signals Ref_pfd and VCO_pfd, which are the reference signal and feedback VCO output signal respectively, act as the clocks for the flipflops. Each input triggers at its rising edge and propagates the supply voltage from the data port D to the output port Q. The outputs of interest, UP and DN, reflect the difference in both frequency and phase between the two input signals. When UP and DN are simultaneously high, the AND gate resets both flipflops. The rising edge trigger behavior can be express in SRE as:

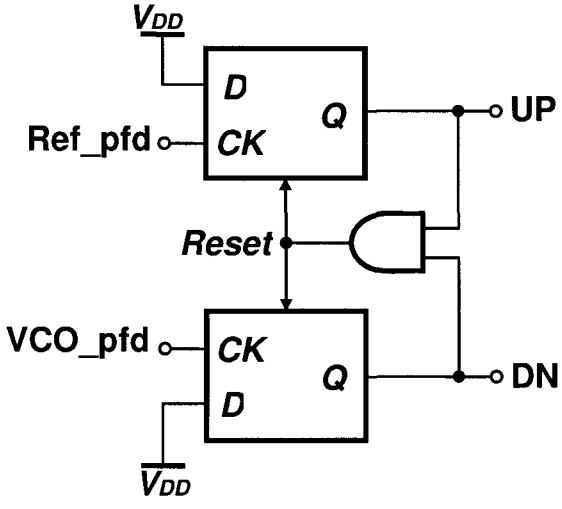


Figure 4.2: Phase and Frequency Detector

$$\texttt{UP(n+1) =IF\{Ref\_pfd(n) = 1} \wedge \texttt{ Ref\_pfd(n-1) = 0, 1, UP(n)\}}$$
$$\texttt{DN(n+1) =IF\{VCO\_pfd(n) = 1} \wedge \texttt{ VCO\_pfd(n-1) = 0, 1, DN(n)\}}$$

Because reset determines the initial condition of the flipflops, we nest reset SRE expression outside the rising edge SRE. We have:

$$\begin{aligned}\texttt{UP(n+1) = IF\{[UP(n) = 1} &\wedge \texttt{ DN(n) = 1], 0, IF\{Ref\_pfd(n) = 1} \wedge \\ &\texttt{Ref\_pfd(n-1) = 0, 1, UP(n)\}\}} \\ \texttt{DN(n+1) = IF\{[UP(n) = 1} &\wedge \texttt{ DN(n) = 1], 0, IF\{VCO\_pfd(n) = 1} \wedge \\ &\texttt{VCO\_pfd(n-1) = 0, 1, DN(n)\}\}}\end{aligned}$$

Charge Pump

The charge pump is usually interposed between the PFD and analog filter to provide voltage or current for the capacitance in the successive filter. It is a typical mixed signal component as the input of is digital signal and the output is a continuous-time signal. In our case, we adopt voltage as source supply. The resulting source supply is proportional to the difference of the output signal UP and DN from PFD. The SRE model of CP is given by

$$\texttt{Charge\_out(n)} = V_c \times \texttt{[UP(n-1) - DN(n-1)]}$$

The SRE of the charge pump is the difference equation of the functional behavior.

Analog Filter

The analog filter is an important analog block in PLL as well as other AMS designs.

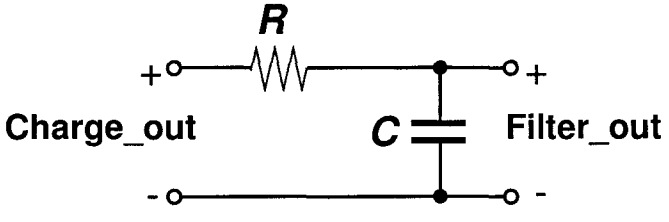


Figure 4.3: First Order Lowpass Filter

It is used to operate continuous-time signals. The analog filter is usually constructed by combining resistances and capacitances. In our frequency synthesizer, a simple first order lowpass filter is employed. The implementation of the analog filter is shown in Figure 4.3. When modeling an analog filter, we are given the transfer function in frequency domain as:

$$H(s) = \frac{1}{1 + \frac{s}{\omega_c}} \tag{4.1}$$

where $\omega_c = \frac{1}{RC}$. ω_c is the cutoff frequency of the lowpass filter. After applying Impulse-Invariant z transform [6], we obtain z domain transfer function:

$$H(z) = \frac{\omega_c T}{1 - z^{-1} exp(-\omega_c T)} \tag{4.2}$$

where T is the sampling time. Taking the inverse z transform of Equation 4.2, we achieve the time domain difference equation of the lowpass filter as

$$Filter\_out(n) = \frac{T}{RC} \times Charge\_out(n) + Filter\_out(n-1) \times e^{-\frac{T}{RC}} \tag{4.3}$$

The corresponding SRE is expressed as:

```
Filter_out(n) = IF{true,
                (T/RC) × Charge_out(n) + Filter_out(n-1) × e^(-T/RC), 0}
```

The SRE is the difference equation of the lowpass filter.

Voltage Controlled Oscillator

The voltage controlled oscillator (VCO) is a key component in the PLL system. In practice, a VCO can be implemented using a ring oscillator [41] or an LC oscillator [20]. It is a pure analog circuit. However, in this thesis we do not model the VCO at circuit level. We instead focus on a highly abstracted mathematical model as higher abstraction approach. An ideal VCO, as shown in Figure 4.4 is a circuit whose output frequency is a linear function of its input voltage (the output of the lowpass filter) and the relation between the two is given by

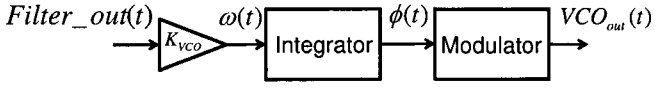


Figure 4.4: Voltage Controlled Oscillator

$$\omega_{out}(t) = \omega_0 + K_{VCO} \cdot Filter\_out(t) \tag{4.4}$$

where ω_0 represents the initial radian frequency when $Filter\_out(t) = 0$ and K_{VCO} is called VCO gain (rad/s/V). The VCO generates a sinusoidal wave signal $V(t) = Asin(\phi(t))$. The argument of the sinusoid $\phi(t)$ is called the "total phase" of the signal. The radian frequency can be defined as the derivative of the phase according to time

$$\omega(t) = \frac{d\phi(t)}{dt} \tag{4.5}$$

Equation 4.5 suggests that if the radian frequency of a signal $\omega(t)$ is known as a function of time, then the phase can be expressed as

$$\phi(t) = \int \omega(t)dt + \phi_0 \tag{4.6}$$

where ϕ_0 is the initial phase. Recall Equation 4.4, we have the mathematical presentation of VCO

$$VCO_{out}(t) = A\cos(\int \omega_{out}(t)dt + \phi_0) \tag{4.7}$$

$$= A\cos(\omega_0 t + K_{VCO}\int_0^t Filter\_out(\tau)d\tau + \phi_0) \tag{4.8}$$

The physical meaning of an integral is that the integral of a function over a finite region is equal to the area the function covers over the same region. An approximation to the integral can be constructed using the sum of individual rectangular areas divided by small intervals within the same region

$$\int_b^a f(x)dx \approx \sum_{i=1}^{n} f(t_i)\Delta_i \tag{4.9}$$

where $a = x_0 \leq x_1 \leq x_2 \cdots \leq x_{n-1} \leq x_n = b$ and the interval $\Delta_i = x_i - x_{i-1}$. When the intervals are relatively small and equal to each other, the error introduced can be ignored. Suppose T to be the smallest time step in C-SRE simulator, the SRE

expression of phase term in Equation 4.8 is given by

$$\texttt{phase(n)} = T \times \texttt{Filter\_out(n)} + \texttt{phase(n-1)}$$

where $Filter\_out(n)$ is the output of lowpass filter and control voltage signal for VCO. The SRE of the VCO block together with the succeeding comparator is given by

$$\texttt{VCO\_out\_1(n)=IF}\{\cos[\omega_0 nT + K_{VCO} \times \texttt{phase(n)} + \phi_0(n)] \geq 0,\ 1,\ 0\}$$

Divider

The divider block (DIV) works as a frequency divider. In order to achieve half frequency, the rising edge of the output occurs every two periods of the input signal. When the frequency select signal $Freq\_sel$ is activated, the output of the divider holds the frequency as half as the input signal. The SRE modeling of the entire functionality of the divider is listed as

$$\begin{aligned}
\texttt{VCO_out_2(n+1)} &= \texttt{IF}\{(\texttt{VCO_out_1(n)} = 1) \wedge \\
&\qquad (\texttt{VCO_out_1(n-1)} = 0),\ \neg\texttt{VCO_out_2(n)},\ \texttt{VCO_out_2(n)}\} \\
\texttt{VCO_pfd(n+1)} &= \texttt{IF}\{\texttt{freq_sel(n)} = 1,\ \texttt{VCO_out_2(n)},\ \texttt{VCO_out_1(n)}\}
\end{aligned}$$

4.2 Online Monitoring of Functional Properties

After SRE modeling, we simulate the PLL using the C-SRE simulator. The simulation results are shown in Figure 4.5. The PLL system with different damping factors ξ [6] (0.1, 0.5 and 0.707, respectively) and same nature frequency are simulated. After the activation of $Freq\_sel$ at 0.5ms, the system begins to track the new frequency and is locked at the same frequency in each case with different lock times. Instead of examining the actual VCO output frequency, we evaluate the lock status by checking whether the output of the lowpass filter stays at the proper DC level. Figure 4.5 indicates two things: (1) the SRE model and the C-SRE simulator

can offer accurate and reliable modeling and simulation for the AMS system; (2) the output of the lowpass filter is quite intuitive for high level abstraction and used as a reference for the functional verification and jitter noise analysis presented in following sections. The verification results of three functional properties are presented in the following as well as a comparison between *online* and *offline* monitoring techniques in terms of simulation time and memory usage.

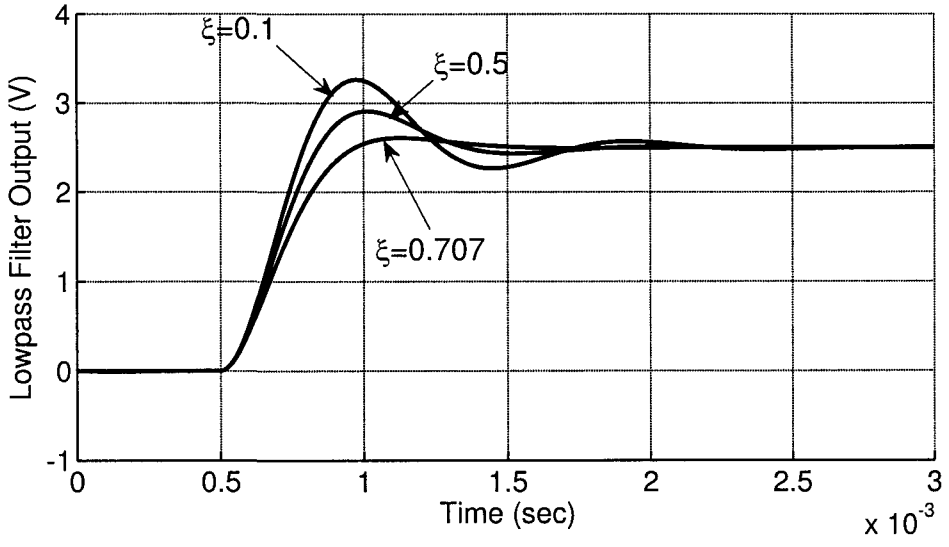


Figure 4.5: Lowpass Filter Output Voltage with Different ξ

Property 1 Lock-time is one of the most important properties of the PLL. It determines how fast the frequency synthesizer stabilizes from one frequency to another. This is the key factor when designing PLL circuits. According to the parameters listed in Table 4.1, the lock time of our system is 0.001 sec. The lock time property can be described as: after the *Freq_sel* signal changes from 0 to 1, the output of the lowpass filter will reach the new DC value within the lock time. This is a safety property. The PSL style definition of the property is shown below

```
Property_1 :  always {Freq_sel==0;Freq_sel==1} |->
                        next![Lock_time](Filter_out==New_DC_Level)
```

Table 4.1: Frequency Synthesizer Parameters

Parameters	Value	Description
T (s)	10^{-8}	Sampling time
RC (s)	0.0001	Filter RC parameter
α	exp(-T/RC)	Charge time parameter
Vc (V)	5	Voltage supply
ω_0(rad· Hz)	$2\pi \times 10^6$	Input signal frequency
ω_{VCO}(rad· Hz)	$2\pi \times 10^6$	VCO operating frequency
K_{VCO} (rad·Hz·V^{-1})	$2\omega_0/Vc$	VCO gain
New_DC_Level(V)	2.5	Filter output

The SERE concatenation operator (;) indicates that the two Boolean expressions it connects hold consecutively [2]. `(Filter_out==New_DC_Level)` is the basic property. The `next!` operator is strong semantics. The SRE expressions of the property are shown below:

`IF{Freq_sel(n-1)==0` $\wedge$ `Freq_sel(n)==1, Subproperty, false }`

`Subproperty=IF{Filter_out(n+Lock_time)==New_DC_Level, true, false}`

We use the nested form for SREs. Instead of showing the SRE in one line, we extract the inner `If-formula` as sub-property. The strong operator *next!* requires that the terminating condition (`Subproperty`) happen. The property monitor implemented in C language is shown below

```
while (freq_sel[i]==1 && freq_sel[i-1]==0){
    for(int n=i; n!=N_max; n++){
        if (filter_out[n]== New_DC_Level && T_sample*(n-i)<=Lock_time){
            property_lock_time = 1; // Satisfied
        } else{
            property_lock_time = 0; // Violated
        }}
    i++;}
```

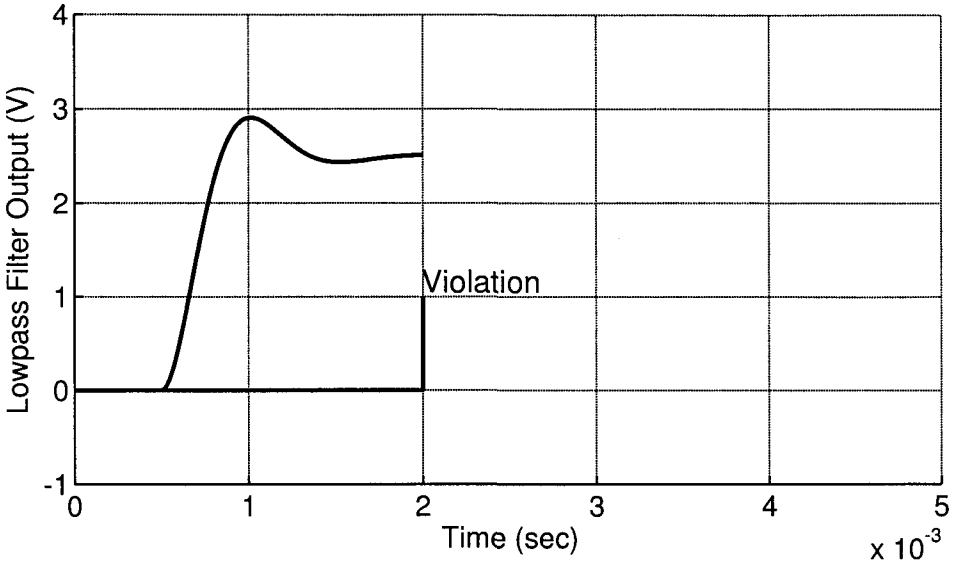


Figure 4.6: Locktime Property

Figure 4.6 indicates that as soon as the time reaches 1.5ms after *Freq_sel* signal changes from 0 to 1, the monitor reports a violation for the property. The simulation then is suspended at 1.5ms. There is no need to look at the simulation trace after 1.5ms in this case. Due to the prompt violation alert, the simulation time is expected to be saved. In addition, two more interesting properties are described below.

Property 2 If the *Freq_sel* is activated the VCO output signal should change to a new frequency eventually. The VCO output stability can be decided by the filter output. If the filter output signal becomes stable (fixed to a *New_DC_Level*), then the VCO output will also be stable at the new frequency. This is a liveness property. The property is expressed in PSL as:

```
Property_2 :  always {Freq_sel==0;Freq_sel==1} |->
                        eventually! {Filter_out == New_DC_Level}
```

The `eventually!` is an LTL style operator in PSL. It specifies that a property holds at the current cycle or at some future cycle. The verification result is shown

in Figure 4.7. This time the monitor is sensitive to satisfaction rather than violation. The reason for that is because this is a liveness property which implies that something good eventually happens. The detection of satisfaction is more feasible than violation in this case. The simulation is terminated when the property is verified at 2.4ms.

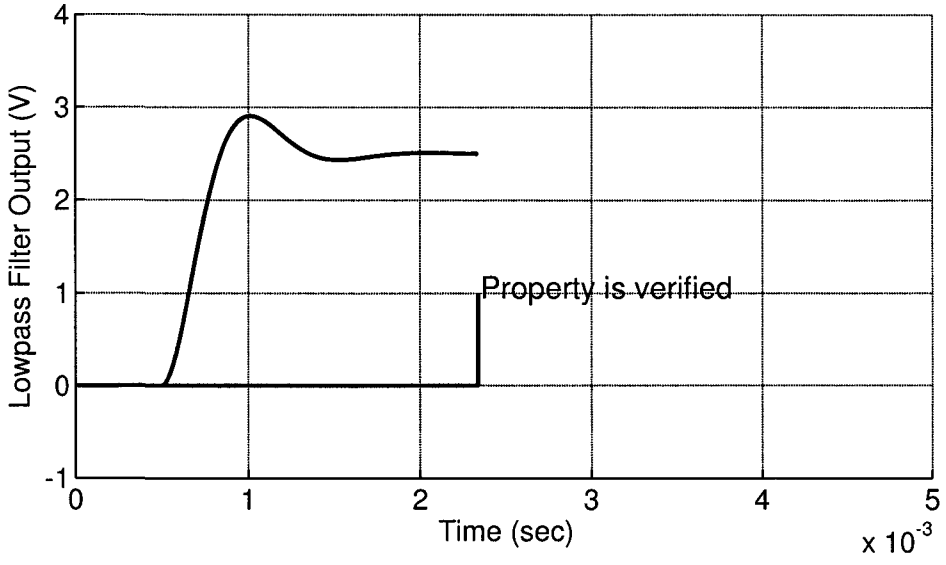


Figure 4.7: Verification Results of Property 2

Property 3 After reset, the *Freq_sel* will be '0', and *Filter_out* will also be '0'. If the *Freq_sel* changes to '1', the *Filter_out* will increase until *New_DC_Level*. Hence, the *Freq_sel* will be '0' until *Filter_out* is larger than '0'. This is a safety property. The property is expressed in PSL as:

```
Property_3 : always (Filter_out == 0) until! (Freq_sel ≠ 0)
```

The `until!` is like the LTL operator `until` in strong form which requires that the termination condition eventually happens. In context, the property 3 requires that signal *Filter_out* is expected to change. This property was successfully verified.

We compared our methodology to the work presented in [3] and the experimental results are listed in Table 4.2. Thereafter, we notice that our proposed *online* monitoring technique performs better than [3] in terms of simulation time. The reason is that *online* monitoring stops the simulation process as soon as the property is verified or violated. The memory usage of our methodology is slightly higher than that of [3] due to the computational efficiency in Matlab. Both methodologies were run on the same ULTRA SPARC-IIIi server (177 MHz CPU, 1GB memory), where all properties described above are satisfied.

Table 4.2: Simulation Results

Property	Online Monitoring		Offline Monitoring		Verification Status
	Simulation Time(sec)	Memory Use(MB)	Simulation Time(sec)	Memory Use(MB)	
Property 1	13.42	36.2	38.87	32.4	Violated
Property 2	13.47	38.2	37.61	32.4	Verified
Property 3	13.35	38.1	43.65	32.4	Verified

In this section, we presented the experimental results of the PLL functional runtime verification. In next section, we will see how the proposed statistical runtime verification applied to random noise in the PLL design.

4.3 Offline Monitoring of Statistical Properties

In this section, we present the statistical runtime verification applied to the jitter noise in the PLL design. We first introduce jitter noise and its metrics. Then we describe how to include jitter noise using SRE. Finally, we present the verification results.

4.3.1 Jitter Noise in Frequency Synthesizer

Jitter is simply the deviation in time between a noisy signal and an ideal one. It affects the quality of the system especially for high frequencies. For example, in communication systems, a large jitter in a clock signal may cause wrong synchronization which results in unexpected information transmission or communication failure. As mentioned in the previous section, PLL based frequency synthesizers generate the signal with the frequency according to that of reference signal. They are used in communication systems as clock generator or clock recovery circuits to provide clean clock signal. Jitter noise may come from outside or inside the PLL system. As shown in Figure 4.8, a major outside source of jitter is the reference clock input. The active components within the PLL are also a major source of jitter noise. The jitter noise in different blocks exhibits differently. In this thesis, we focus only on the jitter associated within the VCO.

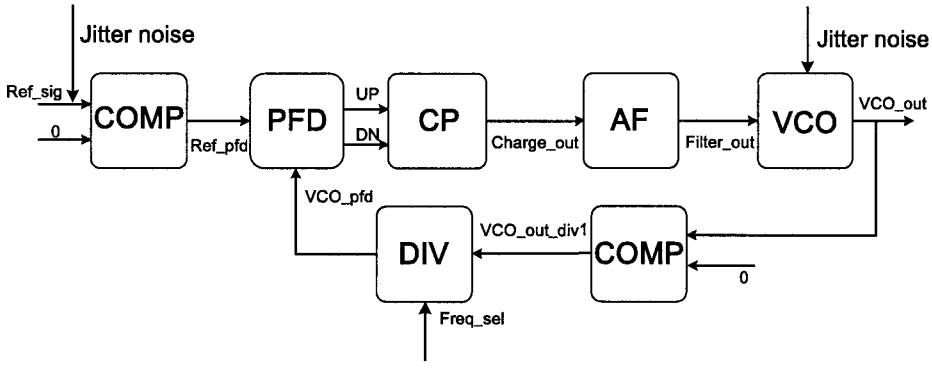


Figure 4.8: PLL Frequency Synthesizer with Jitter Sources

Jitter can be divided into two categories as deterministic jitter (DJ) and random jitter (RJ). The amplitude of DJ, in terms of time, is bounded and that of RJ is unbounded [29]. In AMS systems, the interaction between analog components and digital ones would introduce both types of jitter. In this thesis, we focus on random jitter which features stochastic process.

4.3.2 Jitter Metrics

As mentioned, jitter is defined as the deviation of the edge transition between jittery signals and ideal signals. However, for clock signals, the variation in period from one cycle to another is considered important for analysis. There are three types of jitter metrics. They are phase jitter, period jitter, and cycle-to-cycle jitter.

Phase Jitter

The phase jitter is defined as an edge transition timing difference from the corresponding ideal clock timing. Mathematically, the phase jitter Δt_n is formulated by

$$\Delta t_n = t_n - T_n \tag{4.10}$$

where t_n and T_n are timing values of the edge transition of nth cycle of the jittery clock and the ideal one, shown in Figure 4.9, respectively.

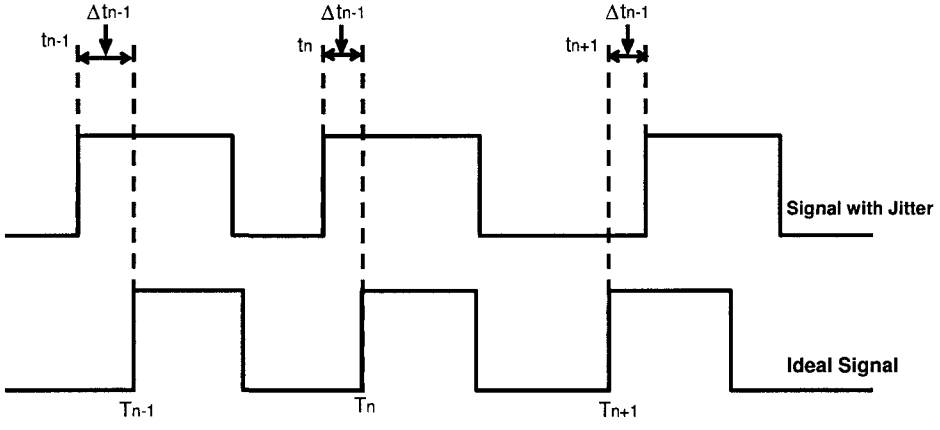


Figure 4.9: Jitter Metrics

Period Jitter

Period jitter is the difference of the actual period from the ideal period. The nth

period is given by $(t_n - t_{n-1})$. Hence, period jitter is defined as

$$\Delta t_{pn} = (t_n - t_{n-1}) - T_0 \tag{4.11}$$

where T_0 is the ideal clock period. For the ideal clock, each cycle has the same period, we have

$$T_{n+1} - T_n = T_n - T_{n-1} = T_0 \tag{4.12}$$

Substituting equation 4.12 for equation 4.11 , we arrive at

$$\Delta t_{pn} = (t_n - t_{n-1}) - (T_n - T_{n-1}) = (t_n - T_n) - (t_{n-1} - T_{n-1}) = \Delta t_n - \Delta t_{n-1} \tag{4.13}$$

It is worth noting that the period jitter is the difference between the phase jitter of current cycle and that of the previous one. The relationship of period jitter and phase jitter helps us to derive one if we have knowledge of the other.

Cycle-to-Cycle Jitter

Cycle-to-cycle jitter is defined as the period deviation of the two consecutive periods. According to the definition, we have

$$\Delta t_{cn} = (t_n - t_{n-1}) - (t_{n-1} - t_{n-2}) = t_n + t_{n-2} - 2t_{n-1} \tag{4.14}$$

Remember the definition of period jitter in Equation 4.11. The difference between two consecutive period jitter is

$$\Delta t_{pn} - \Delta t_{pn-1} = (t_n - t_{n-1} - T_0) - (t_{n-1} - t_{n-2} - T_0) = t_n + t_{n-2} - 2t_{n-1} \tag{4.15}$$

Investigating Equation 4.14 and Equation 4.15, it is not difficult to identify the relationship between cycle-to-cycle jitter and period jitter

$$\Delta t_{cn} = \Delta t_{pn} - \Delta t_{pn-1} \tag{4.16}$$

Further investigation can be carried out by substituting Equation 4.13 to Equation 4.16, we obtain

$$\Delta t_{cn} = \Delta t_{pn} - \Delta t_{pn-1} = (\Delta t_n - \Delta t_{n-1}) - (\Delta t_{n-1} - \Delta t_{n-2}) \tag{4.17}$$

Equation 4.17 presents the interrelationship between phase jitter, period jitter and cycle-to-cycle jitter that cycle-to-cycle jitter is the first difference function of period jitter and the second difference function of the phase jitter [29]. In this thesis, we employ period jitter metric in the experiments.

4.3.3 Jitter in VCO

VCO oscillates with the frequency proportional to the input voltage signal coming from the lowpass filter.

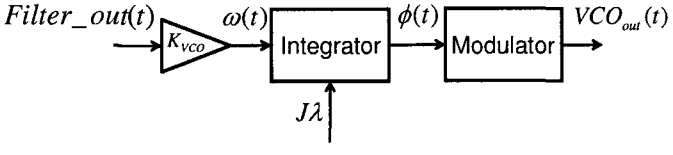


Figure 4.10: VCO Model with Jitter Noise

As mentioned, jitter noise can be found within PLL system especially in VCO. The jitter in VCO is mainly caused by thermal noise of the circuit. Hence, it exhibits Gaussian random process. The model of a VCO with jitter noise is illustrated in Figure 4.10. In fact, VCO generates the sine wave by dealing with the frequency. Hence, the jitter, which is defined as variation in the period, has to be modeled as

a variation in the frequency of the VCO. Assume that, the frequency of a periodic signal without jitter is given by

$$f = \frac{1}{T} \tag{4.18}$$

where T is the period of the ideal signal. The jittery frequency can be represented as

$$f_{jitter} = \frac{1}{T + \Delta T} = \frac{1}{\frac{1}{f} + \Delta T} = \frac{f}{1 + \Delta T \cdot f} \tag{4.19}$$

$$\Delta T = J\lambda \tag{4.20}$$

where J is the jitter deviation and λ is a zero mean unit-variance Gaussian random process. Let $\phi(t)$ be the phase of the integral term in Equation 4.8. We have

$$\phi(t) = K_{VCO} \int_0^t Filter\_out(\tau) d\tau \tag{4.21}$$

Suppose $\omega(t) = 2\pi f(t) = K_{VCO} \cdot Filter\_out(t)$, we obtain

$$f(t) = \frac{K_{VCO} \cdot Filter\_out(t)}{2\pi} \tag{4.22}$$

which relates the input control voltage $Filter\_out(t)$ and frequency $f(t)$ by multiplication of VCO gain factor. By substituting Equation 4.22 into Equation 4.19, we achieve

$$f_{jitter}(t) = \frac{f(t)}{1 + \Delta T \cdot f(t)} = \frac{f(t)}{1 + \Delta T \cdot \frac{K_{VCO} \cdot Filter\_out(t)}{2\pi}} \tag{4.23}$$

We finally derive the formula of VCO with jitter noise by integrating the Equation 4.23 into the original VCO Equation 4.8.

$$VCO_{out}(t) = Acos(\omega_0 t + K_{VCO} \int_0^t \frac{Filter\_out(\tau)}{1 + \frac{J\lambda \cdot K_{VCO} \cdot Filter\_out(\tau)}{2\pi}} d\tau + \phi_0) \tag{4.24}$$

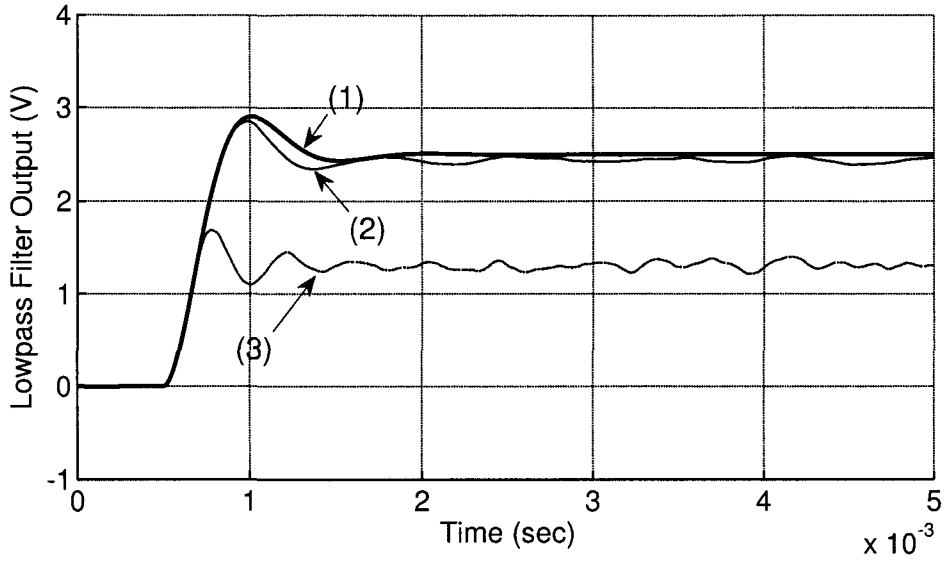


Figure 4.11: Filter Output with Jitter Noise in VCO

Jitter noise is nothing but the phase noise in frequency domain. The random jitter presented in the Equation 4.24 delivers the change in phase after the integral. The deviation of the VCO output may cause the phase error information different from the ideal. Hence, the system characteristic such as lock time may change. Figure 4.11 shows the effect caused by jitter noise. The thick line labeled (1) represents the output of the lowpass filter without jitter noise. The thin line (2), whose $J = 1.1 \times 10^{-7}$s shows that the lowpass filter output almost settles to a new DC level but not stable at the level. The dotted line (3), whose $J = 3 \times 10^{-7}$s, exhibits that the lowpass filter output does not settle to the desired DC value and PLL is unable to lock within the lock time specification of the PLL. It is intuitive that larger jitter noise is expected to cause the PLL lock failure.

4.3.4 Statistical Runtime Verification

Intuitively, we expect the violation to occur for the jitter noise shown in Figure 4.11. However, functional verification can only indicate whether or not the system behaves correctly. In this section, we use the Monte Carlo Monitor introduced in Section 3.4.2

to analyze the jitter noise. In PLL, the VCO exhibits accumulating jitter. The accumulating jitter depends not only on the direct input but also the on output of previous transitions. In order to provide an accurate analysis, the period jitter metric is employed. Compared with phase jitter, period jitter refers not only to the ideal signal but also to itself. In the VCO, the output signal varies during the oscillation process. Hence the cycle-to-cycle jitter, which refers only to the signal itself, is not a proper metric when measuring jitter for VCO. The period jitter is measure using Equation 4.11 and calculated as standard deviation of the sample periods.

We conduct a hypothesis test using Monte Carlo method in an offline fashion. The reason we chose offline observation is that the more information we collect the more reliable decision we will make. The property is expressed as: the period jitter of the given system is less than a specific value. The specific value comes from the system specification of phase noise. For example, if the phase noise is $\mathcal{L} = -25$dBc at the offset frequency 10Hz, the corresponding period jitter to this phase noise is calculated as 5.62ns [26]. As a result, the null hypothesis H_0 and the alternative hypothesis H_1 of this property can be expressed as

$$H_0 : J_{period} \leq 5.62ns; \tag{4.25}$$

$$H_1 : J_{period} > 5.62ns. \tag{4.26}$$

where J_{period} is the period jitter of the VCO output. We estimate J_{period} by observing the information along the simulation path (0.005s) as a sample. The observed period jitter is denoted by J_{obs}. Since a large value would provide the evidence for the rejection of the null hypothesis H_0. An upper tail test scenario is considered in this case. The experimental results for several jitter deviation factors J (given in Equation 4.20) with the confidence level $\delta = 0.95$ ($\alpha = 0.05$) are shown in Table 4.3. The simulation was carried out under the significance level $\alpha = 0.05$. J varies from

10^{-8}s to 10^{-7}s. The acceptance of the null hypothesis H_0 indicates that the property is satisfied and the rejection of H_0 indicates that the property is violated and the period jitter in VCO is larger than the specification. Due to the upper tail test, the evidence of rejecting H_0 is that the observed value T_{obs} is greater than the critical value based on the significance level. When $J = 5 \times 10^{-7}$s, the Monte Carlo monitor announces the rejection of H_0 based on the fact that T_{obs} is greater than the critical value (i.e., it falls into the rejection region). The experiment was performed with the Monte Carlo trials $M = 1000$. The last column lists the error margins (ϵ) for the confidence interval of 95% when J varies. Each error margin forms a confidence interval with the observed value J_{obs} for J_{period}. For example, when $J = 5 \times 10^{-7}$s, the probability that the true value of the period jitter of the whole simulation path presents within the interval (90.4403, 92.5857)ns is 95%. The interval falls into the rejection region which indicates that we have the confidence level of 95% to reject H_0 in this case. The error margin ϵ is calculated using Equation 3.17 introduced in Section 3.4.3.

Table 4.3: Statistical Runtime Verification with Different J

J(s)	*Critical Value*	T_{obs}	J_{obs} (ns)	H_0	α	ϵ (ns)
1×10^{-8}	1.5069	-3.8455	3.4609	Accept	0.05	1.0710
5×10^{-8}	1.5896	-1.7013	4.6667	Accept	0.05	1.0589
1×10^{-7}	1.6597	0.5900	5.9552	Accept	0.05	1.0711
5×10^{-7}	1.5442	152.73	91.513	Reject	0.05	1.0727

It is noted from Table 4.3 that when J increases from 1×10^{-7}s to 5×10^{-7}s, the Monte Carlo monitor experiences a procedure that the decision changes from acceptance to rejection. Table 4.4 shows the verification results influenced by the variation of J and α. J increases by a small step from 1×10^{-7}s to 1.4×10^{-7}s. The decision tends to change from acceptance to rejection. However, for certain selection of J, if we change the significance level α, the decision can be different. For example, in the case of $J = 1.2 \times 10^{-7}$s, we accept H_0 when α is 0.05; while

we have to reject H_0 when α is 0.1. Figures 4.12 (a) and (b) show the observed value T_{obs} (small triangle) and the rejection region (shaded area) in the case that $J = 1.2 \times 10^{-7}$s and α is 0.05 and 0.1, respectively. In Figure 4.12 (a), the observed value is located outside the rejection region. In Figure 4.12 (b), the rejection region is enlarged and includes the observed value T_{obs}. It can be explained using the critical value approach: the fact that reducing the confidence level δ, or increasing the significance level α, makes the critical value smaller. As a result, the rejection region is enlarged accordingly. If the observed value happens to fall within the enlarged rejection region, the null hypothesis is rejected. Similar situation occurs when $J = 1.3 \times 10^{-7}$s except that when the confidence level increases from 95% to 99% (α from 0.05 to 0.01), the Monte Carlo monitor accepts H_0 instead of rejecting it. This situation is shown in Figures 4.12 (c) and (d). In addition, we notice that the error margin ϵ increases as the confidence level decreases and vice versa. The reason is that in order to achieve higher confidence level, the interval is supposed to be larger to allow the estimated value to be included there. In other words, the probability that the estimated value falls into the narrower interval is smaller than that for the wider one.

Table 4.4: Statistical Runtime Verification with Different J and α

J(s)	*Critical Value*	T_{obs}	J_{obs} (ns)	H_0	α	ϵ (ns)
1×10^{-7}	1.6597	0.5900	5.9552	Accept	0.05	1.0711
1.1×10^{-7}	1.6383	0.9244	6.1432	Accept	0.05	1.0730
1.2×10^{-7}	1.6571	1.5768	6.5267	Accept	0.05	1.0953
1.2×10^{-7}	1.1920	1.5768	6.5267	Reject	0.1	9.0523×10^{-1}
1.3×10^{-7}	1.5874	1.9480	6.8476	Reject	0.05	1.1482
1.3×10^{-7}	2.4287	1.9480	6.8476	Accept	0.01	1.3280
1.4×10^{-7}	1.6803	2.8203	7.2094	Reject	0.05	1.1057
1.4×10^{-7}	1.3506	2.8203	7.2094	Reject	0.01	1.4374

The hypothesis test results can be different for different confidence levels when the observed value is approaching the critical value. The accuracy would be affected

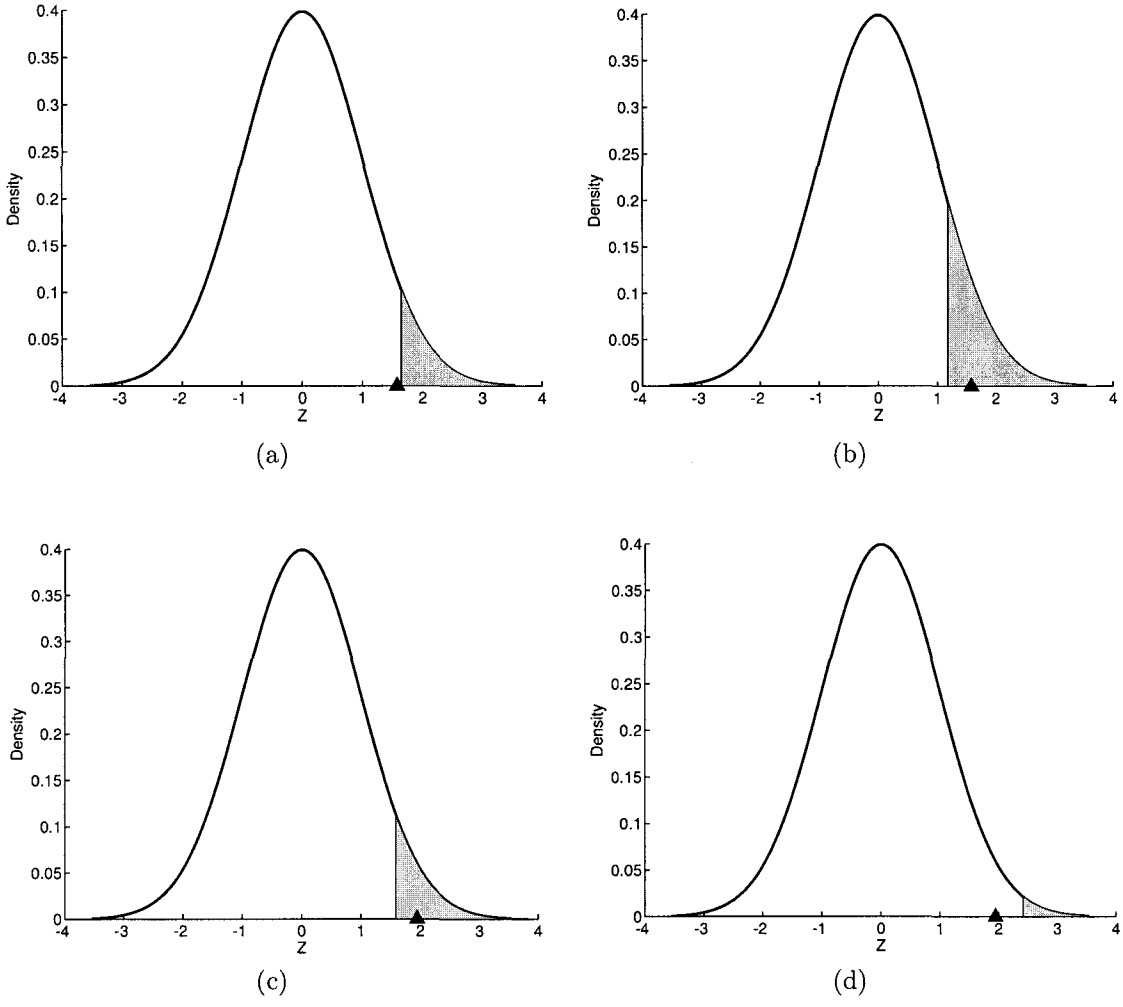


Figure 4.12: Effects of Confidence Level Selection

if the confidence level is too high or too low. On the other hand, the confidence level influences the error margin. Higher confidence level would increase the error margin and degrade the reliability; lower confidence level on the other hand would increase the rejection region and cause low accuracy. The reason is that the interval needs to be enlarged in order to include the estimated value for higher probability, or higher confidence level. 95% of confidence level, which compromises the two situations, is the most commonly used and suitable for most engineering and science researches [40].

In order to evaluate the performance of the Monte Carlo monitor we chose different numbers of trials M which apply both to Monte Carlo hypothesis testing and performance evaluation. At the same time, we kept the factor $J = 1.4 \times 10^{-7}$s and the hypothesis H_0 and H_1 is the same as given in Equation 4.25 and 4.26. Table 4.5 lists the estimated significant value $\hat{\alpha}$, the performance indicator η and the simulation time for different selection of M. When M increases, the performance indicator η decreases accordingly. This implies that the more trials we apply on a Monte Carlo simulation, the more accurate estimation it will produce. However, the tradeoff for large number of trials is the simulation time they consume. It is shown that the simulation time is almost doubled when M increases from 100 to 5000. A tradeoff can always be found between desired accuracy and simulation time.

Table 4.5: Performance of Monte Carlo Monitoring with Different Trials M

M	α	$\hat{\alpha}$	η	Simulation Time (s)
100	0.05	0.03	0.02	1.7316
500	0.05	0.042	0.008	2.0436
1000	0.05	0.049	0.001	2.3088
5000	0.05	0.0502	0.0002	4.4148

4.4 Discussion

In this chapter, a PLL based frequency synthesizer is modeled and several interesting properties are verified. The SRE demonstrates its power to model pure analog, pure digital, and even mixed signal circuits. In addition, the PSL properties are written in SRE to perform *online* monitoring. The SRE model is simulated in C-SRE simulator. The simulator allows both *online* and *offline* monitoring. According to the comparison of the two, the *online* monitoring technique surpasses the *offline* method in terms of simulation time. Based on the original intention of runtime verification, the savings in computational resources is expected. Moreover, the high level abstraction for AMS designs using SRE costs less efforts in terms of modeling and simulation time [1]. On the other hand, in statistical runtime verification, we applied the Monte Carlo hypothesis testing to the jitter noise property in the VCO. The verification is performed in an *offline* fashion. The reason is that the jitter noise in VCO depends not only on the input control voltage but also on the previous output value. We need a large sample of the jitter noise to perform the Monte Carlo monitoring. The computational expense, such as the simulation time, is not the issue for statistical runtime verification. The confidence level and error margin are of significant interest to us. In the end, the performance of the hypothesis test is evaluated using Monte Carlo simulation. There is a dilemma in the proposed statistical runtime verification. Once we intend to increase the confidence level of the test, the accuracy and reliability has to be compromised. It is not unusually the case when statistical methods are applied. In fact, we estimate the statistical property of the entire population by observing a sample sequence of it. The confidence level of 100% is impossible to reach. However, there are some techniques to provide more reliable decision with certain confidence level such as to make sure the confidence interval within the rejection region or non-rejection region.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, we presented a methodology for functional and statistical runtime verification of AMS designs. We introduced the concepts of SRE and an SRE-based simulator, namely C-SRE. The ability to choose the right level of abstraction makes it possible for the verification engineer to describe important components of the design accurately, which is a significant concern in analog design and verification. SRE demonstrates its power in modeling pure analog, pure digital, or even the mixed signal components at high level of abstraction. The high level abstraction using SRE not only saves the simulation time, but also saves the time to model the AMS design.

For functional runtime verification, the properties are expressed in PSL formally. The monitor is constructed by translating the PSL into SRE notations. By doing this, the consistency between the model of the AMS design and the property is established. We performed the *online* monitoring in the C-SRE simulator. The functional runtime verification offers a dynamic monitoring method to the behavior of the AMS system. Compared with *offline* monitoring, the proposed methodology saves the computational resources in terms of simulation time. Another benefit of

the proposed methodology is that the verification point can either be located at analog signals or digital signals. This enables the monitor to verify the functional properties both for the whole system and the individual block. In the case study, several functional properties of a PLL based frequency synthesizer were verified using the proposed methodology.

We also used hypothesis testing and Monte Carlo simulation for statistical runtime verification of the AMS design. The hypothesis test makes the decision between the null hypothesis and its exclusive alternative hypothesis. There are two types of errors for hypothesis test: Type I error α and Type II error β. Type I error is also called significance level. The whole procedure operates based on confidence level δ which is related to α by $\delta = 1 - \alpha$. By evaluating the observed value of the test sample and the critical value associated with the significance level, the decision of whether or not rejecting null hypothesis is made. Each decision comes with a confidence level along with the margin of error. The error margin indicates the probability that the estimated value is inclosed to an interval is the confidence level. Monte Carlo simulation is used to generate the estimate random model in the case that we do not have the knowledge about the distribution of the population. This makes the hypothesis robust to most stochastic processes. We present the random jitter analysis using the proposed statistical runtime verification method. The effects of the confidence level selection are illustrated and discussed. Higher confidence level increases the reliability and enlarges error margin for the interval. The conclusion is that the situation is inevitable and the choice of the confidence level has to be made according to the system specification. In addition, the performance of the Monte Carlo monitor is evaluated and discussed.

The main advantages of our methodology are: (1) the SRE expression is likely to be understood both by analog and digital engineers; (2) SRE fully supports safety and liveness property as well as strong and weak semantics ; (3) since both the AMS design and PSL properties are described using SREs, continuous time behavior can

be simulated and monitored together in discrete time; (4) online monitoring saves cost in terms of memory usage and simulation time; and (5) statistical runtime verification is able to analyze the statistical properties of the system with a stochastic process.

5.2 Future Work

The C-SRE simulator in this thesis performs both the simulation and *online* functional runtime verification. However, enhancement can be made to the monitor part, such as the automation of the SRE model generation in C language. Another limitation of the monitor is that only one property can be verified at a time. Future work on the monitor would be the automation of the monitor construction and implementation of the support for multiple property verification.

We believe our first attempt to statistical runtime verification to AMS design was successful. However, the approach would be extended to *online* fashion without losing any accuracy and reliability in terms of confidence level and error margin. The benefits of *online* statistical monitoring would be: (1) interactively increase the simulation trace according to the current observed statistical information in order to guarantee the accuracy of the results ; (2) interactively change the input in order to improve the coverage especially for the analysis of noise.

Finally, the statistical runtime verification needs to be integrated with the verification language such as PSL. Because the statistical properties are also important for the verification of AMS system especially for dynamic characteristics.

Bibliography

[1] N. Abbasi, R. Narayanan, G. Al-Sammane, M. Zaki, and S. Tahar. Enabling AMS Simulation using Recurrence Notations, Technical Report, Department of ECE, Concordia University, Montreal, Canada, May 2008. http://hvg.ece.concordia.ca/Publications/TECH_REP/CSRE_TR08/.

[2] Accellera Property Specification Language Reference Manual (version 1.1). http://www.eda.org/vfv/docs/PSL-v1.1.pdf.

[3] G. Al Sammane, M.H. Zaki, Z.J. Dong, and S. Tahar. Towards Assertion Based Verification of Analog and Mixed Signal Designs Using PSL. In *Proc. Forum on Specification and Design Languages*, pages 293–298, 2007.

[4] R. Alur, T.A. Henzinger, and M.Y. Vardi. Parametric real-time reasoning. In *Proc. of the 25th Annual Symposium on Theory of Computing*, pages 592–601. ACM Press, 1993.

[5] A. Aziz, K. Sanwal, V. Singhal, and R.K. Brayton. Verifying Continuous Time Markov Chains. In *Computer Aided Verification*, volume 1102 of LNCS, pages 269–276. Springer, 1996.

[6] R.E. Best. *Phase-Locked Loops: Design, Simulation, and Applications.* McGraw-Hill, 2003.

[7] E. Clarke, A. Donzé, and A. Legay. Statistical Model Checking of Mixed-Analog Circuits with an Application to a Third Order Δ-Σ Modulator. In *Hardware*

and Software: Verification and Testing, volume 5394 of *LNCS*, pages 149–163. Springer, 2009.

[8] E. Clarke, D. Kroening, J. Ouaknine, and O. Strichman. Computational Challenges in Bounded Model Checking. *International Journal on Software Tools for Technology Transfer*, 7(2):174–183, 2005.

[9] Collet International Research. Survey, 2002.

[10] Intel Corporation. Moores Law Timeline. http://download.intel.com/press-room/kits/ events/moores_law_40th/MLTimeline.pdf, 2008.

[11] H. de Bellescise. La rception Synchrone. *Onde Electrique*, 11:230–240, 1932.

[12] C. Eisner. PSL for Runtime Verification: Theory and Practice. In *Runtime Verification*, volume 4839 of *LNCS*, pages 1–8. Springer, 2007.

[13] C. Eisner and D. Fisman. *A Practical Introduction To PSL.* Springer, 2006.

[14] E.A. Emerson. Temporal and Modal Logic. In *Handbook of Theoretical Computer Science*, pages 995–1072. Elsevier, 1990.

[15] E.A. Emerson and J.Y. Halpern. "Sometimes" and "Not never" Revisited: On Branching Versus Linear Time Temporal Logic. *Journal of the ACM*, 33(1):151–178, 1986.

[16] G. Frehse. PHAVer: Algorithmic Verification of Hybrid Systems past HyTech. *International Journal on Software Tools for Technology Transfer*, 10(3):263–279, 2008.

[17] G. Frehse, B.H. Krogh, R.A. Rutenbar, and O. Maler. Time Domain Verification of Oscillator Circuit Properties. *Electronic Notes in Theoretical Computer Science*, 153(3):9–22, 2006.

[18] M.J.C. Gordon and T.F. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic.* Cambridge University Press, 1993.

[19] O. Hasan. *Formal Probabilistic Analysis using Theorem Proving.* Ph.D. Thesis, Concordia University, 2008.

[20] E. Hegazi, J. Rael, and A. Abidi. *The Designer's Guide to High-Purity Oscillators.* Springer, 2004.

[21] A. Jesser, S. Laemmermann, A. Pacholik, R. Weiss, J. Ruf, W. Fengler, L. Hedrich, T. Kropf, and W. Rosenstiel. Advanced Assertion Based Design for Mixed-Signal Verification. *Transactions on Fundamentals of Electronics, Communications and Computer Sciences.* E91(12):3548–3555, 2008.

[22] K. Jones, V. Konrad, and D. Nickovic. Analog Property Checkers: A DDR2 Case Study. In *Proc. Formal Verification of Analog Circuits (FAC)*, 2008.

[23] M. Kim. Information Extraction for Run-time Formal analysis. PhD thesis, University of Pennsylvania, 2001.

[24] M. Kim, S. Kannan, I. Lee, O. Sokolsky, and M. Viswanathan. Java-mac: A run-time assurance approach for java programs. *Formal Methods in System Design*, 24(2):129–155, 2004.

[25] T. Kropf. *Introduction to Formal Hardware Verification.* Springer, 2000.

[26] K. Kundert. Predicting the Phase Noise and Jitter of PLL-Based Frequency Synthesizers. http://www.designers-guide.org/.

[27] P. L'Ecuyer. Uniform Random Number Generation. *Annals of Operations Research*, 53:77C120, 1994.

[28] E.L. Lehmann. *Testing Statistical Hypotheses.* Springer, 2008.

[29] M.P. Li. *Jitter, Noise, and Signal Integrity at High-Speed.* Prentic Hall, 2007.

[30] O. Maler and D. Nickovic. Monitoring Temporal Properties of Continuous Signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, volume 3253 of *LNCS*, pages 152–166. Springer, 2004.

[31] W. L. Martinez and A. R. Martinez. *Computational Statistics Handbook with MATLAB* . Chapman & Hall/CRC, 2001.

[32] The MathWorks. Matlab. http://www.mathworks.com/.

[33] N. Metropolis and S. Ulam. The Monte Carlo Method. *Journal of the American Statistical Association*, 39:335–341, 1949.

[34] A. Meyer. *Principles of Functional Verification.* Newnes, 2003.

[35] MLDesign Tech. http://www.mldesigner.com.

[36] K. Morin-Allory, L. Fesquet, B. Roustan, and D. Borrione. Asynchronous online-monitoring of logical and temporal assertions. In *Embedded Systems Specification and Design Languages*, volume 10 of *LNEE*, pages 243–253. Springer, 2008.

[37] D. Nickovic and O. Maler. AMT: A Property-Based Monitoring Tool for Analog Systems. In *Formal Modeling and Analysis of Timed Systems*, volume 4763 of *LNCS*, pages 304–319. Springer, 2007.

[38] A.V. Oppenheim, R.W. Schafer, and J.R. Buck. *Discrete-time Signal Processing.* Prentice Hall, 1999.

[39] PVS. http://pvs.csl.sri.com, 2008.

[40] T. Pyzdek and P. Keller. *Quality Engineering Handbook.* CRC, 2003.

[41] B. Razavi. *Design of Analog CMOS Integrated Circuits.* McGraw-Hill, 2001.

[42] U. Sammapun, I. Lee, and O. Sokolsky. RT-MaC: Runtime Monitoring and Checking of Quantitative and Probabilistic Properties. In *Proc. Real-Time Computing Systems and Applications*, pages 147–153, 2005.

[43] B. Stackhouse, S. Bhimji, C. Bostak, D. Bradley, B. Cherkauer, J. Desai, E. Francom, M. Gowan, P. Gronowski, D. Krueger, C. Morganti, and S. Troyer. A 65 nm 2-Billion Transistor Quad-Core Itanium Processor. *IEEE Journal of Solid-State Circuits*, 44:18–31, 2009.

[44] SystemC-AMS Standard Draft 1, 2008. http://www.systemc-ams.org/.

[45] Verilog-AMS Language Reference Manual, 2004. http://www.accellera.org/.

[46] VHDL-AMS Language Reference Manual, 2004. http://www.eda.org/vhdl-ams/.

[47] H.L.S. Younes and R.G. Simmons. Probabilistic Verification of Discrete Event Systems Using Acceptance Sampling. In *Computer Aided Verification*, volume 2404 of *LNCS*, pages 23–39. Springer, 2002.

[48] H.L.S. Younes. Error control for probabilistic model checking. In *Verification, Model Checking, and Abstract Interpretation*, volume 3855 of *LNCS*, pages 142–156. Springer, 2006.

[49] J. Yuan, C. Pixley, and A. Aziz. *Constraint-Based Verification.* Springer, 2006.

[50] M. Zaki, S. Tahar, and G. Bois. Formal Verification of Analog and Mixed Signal Designs: A Survey. *Microelectronics Journal*, 39(12):1395–1404, 2008.