VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV MIKROELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF MICROELECTRONICS

MODERNÍ METODY VERIFIKACE SMÍŠENÝCH INTEGROVANÝCH OBVODŮ

MODERN METHODS OF MIXED-SIGNAL INTEGRATED CIRCUIT VERIFICATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. JAROSLAV HRADIL
AUTHOR

VEDOUCÍ PRÁCE doc. Ing. LUKÁŠ FUJCIK, Ph.D.
SUPERVISOR

BRNO 2014

VYSOKÉ UČENÍ
TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

Ústav mikroelektroniky

Diplomová práce

magisterský navazující studijní obor
Mikroelektronika

Student:	Bc. Jaroslav Hradil	***ID:***	125452
Ročník:	2	***Akademický rok:***	2013/2014

NÁZEV TÉMATU:

Moderní metody verifikace smíšených integrovaných obvodů

POKYNY PRO VYPRACOVÁNÍ:

Přehledně zpracujte problematiku moderních metod verifikace integrovaných obvodů pracujících ve smíšeném módu. Zaměřte se na „assertion based methodology“, která je běžná v oblasti verifikace číslicových obvodů a postupně proniká do oblasti smíšených a analogových IO. Porovnejte možnosti popisných jazyků podporujících „assertion based methodology“ z hlediska jejich možností použití pro smíšené případně analogové obvody. V diplomové práci vytvořte ve vybraném jazyce kód pro verifikaci řídicího obvodu spínaných napájecích zdrojů nebo jeho bloků.

DOPORUČENÁ LITERATURA:

[1] Chen J., Henrie M., Mar M. F., Nizic M., „Mixed-Signal Methodology Guide“, Cadence Design Systems, Inc., August 2012, ISBN:978-1-300-03520-6

Termín zadání: 10.2.2014 ***Termín odevzdání:*** 29.5.2014

Vedoucí práce: doc. Ing. Lukáš Fujcik, Ph.D.
Konzultanti diplomové práce:

prof. Ing. Vladislav Musil, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Autor diplomové práce nesmí při vytváření semestrální práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

BRNO UNIVERSITY
OF TECHNOLOGY

Faculty of Electrical Engineering
and Communication

Department of Microelectronics

Master thesis

Master's study programme
Microelectronics

Student:	Bc. Jaroslav Hradil	***ID:***	125452
Year of study:	2	***Academic year:***	2013/2014

THESIS TITLE:

Modern verification methods of mixed-signal integrated circuits

INSTRUCTION:

Describe modern verification methods of mixed-signal integrated circuits. Focus on the „assertion based methodology“, which is common in verification of digital circuits and gradually expands into the field of analog and mixed-signal integrated circuits. Compare descriptive languages supported in „assertion based methodology“ in terms of their potential use for analog and mixed-signal circuits. In the chosen descriptive language, create assertions for verification of switching power supply control circuit, or its blocks.

RECOMMENDED LITERATURE:

[1] Chen J., Henrie M., Mar M. F., Nizic M., „Mixed-Signal Methodology Guide“, Cadence Design Systems, Inc., August 2012, ISBN:978-1-300-03520-6

Date of assignment: 10.2.2014 ***Date of submission:*** 29.5.2014

Thesis supervisor: doc. Ing. Lukáš Fujcik, Ph.D.
Thesis consultant:

prof. Ing. Vladislav Musil, CSc.
Specialisation council chairman

WARNING:

The author of this master thesis must not infringe the rights of third parties, especially by interfering with copyrights of others in an unauthorized manner. The author is fully aware of the legal consequences of an infringement of provisions as per Section 11 and following of Act No 121/2000 Coll. On copyright and rights related to copyright and on amendments to some other laws (the Copyright Act) in the wording of subsequent directives including the possible criminal consequences as resulting from provisions of Part 2, Chapter VI, Article 4 of Criminal Code 40/2009 Coll.

ABSTRAKT

Tato diplomová práce se zabývá verifikací integrovaných obvodů pracujících ve smíšeném módu. Teoretická část práce obsahuje přehled moderních verifikačních metod a zaměřuje se zejména na „assertion based methodology“ . V praktické části práce jsou pak rozebrány popisné jazyky používané u této metody, a následně je vytvořen kód pro verifikaci bloku řídícího obvodu spínaných zdrojů.

KLÍČOVÁ SLOVA

verifikace, behaviorální modelování, wreal, assertion, mixed-signal

ABSTRACT

This master thesis deals with verification methods of mixed-signal integrated circuits. Theoretical part contains summary of modern verification methods with emphasis on „assertion based methodology“ . The practical part analyses descriptive languages used in this method and a code for verification of a power supply control circuit block is created.

KEYWORDS

verification, behavioral modeling, wreal, assertion, mixed-signal

BIBLIOGRAFIKCÁ CITACE

HRADIL, J. Moderní metody verifikace smíšených integrovaných obvodů. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2014. 56 s. Vedoucí diplomové práce doc. Ing. Lukáš Fujcik, Ph.D..

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „ Moderní metody verifikace smíšených integrovaných obvodů “ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne ....

podpis autora

research centre
sensor, information and communication systems

Faculty of Electrical Engineering
and Communication

Brno University of Technology
Technicka 12, CZ-61600 Brno, Czechia

http://www.six.feec.vutbr.cz

Experimentální část této diplomové práce byla realizována na výzkumné infrastruktuře vybudované v rámci projektu CZ.1.05/2.1.00/03.0072
Centrum senzorických, informačních a komunikačních systémů (SIX)
operačního programu Výzkum a vývoj pro inovace.

MINISTERSTVO ŠKOLSTVÍ,
MLÁDEŽE A TĚLOVÝCHOVY

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

2007–13
OP Výzkum a vývoj
pro inovace

OBSAH

INTRODUCTION

Mixed-signal design is a combination of analog and digital circuitry. Mixed-signal applications are among the fastest growing market segments in the electronic and semiconductor industry. Mixed signal content in most of today's integrated circuits has increased from 10-20 % to 50 % or more due to increased needs for mobility, higher performance and integration of interfaces. Similarly, what used to be pure analog blocks now include significant amounts of digital logic either to increase functionality or to assist the analog portions of the design achieve target performance.

This escalating complexity poses severe challenges for mixed-signal verification and uncertainties in verification coverage. According to industry estimates [2] , more than 60 % of SoC design re-spins at 45 nanometers and below are due to mixed-signal errors. A re-spin costs extra money and delays a product rollout for weeks or months. Many re-spins are due to commonplace, avoidable errors such as inverted or disconnected signals. To avoid these errors, mixed-signal SoC teams need to implement modern verification methodologies.

The aim of this master thesis is to provide a summary of modern verification methods used in mixed-signal designs. Emphasis is put on „assertion based methodology" , which is common in verification of digital circuits and gradually expands into the field of analog and mixed-signal integrated circuits. The practical part analyses descriptive languages used in this method and a code for verification of a power supply control circuit block is created.

1 MIXED-SIGNAL VERIFICATION

Verification is a procedure used for checking that designed circuit meets requirements and specifications and that it fulfills its intended purpose.

The basic verification process of electronic devices involves creating a verification plan, development of test benches, simulation, post processing of results including measurements and comparison with the specification [1].

1.1 Gap between digital and analog verification process

Verification of mixed-signal designs with plainly separated analog and digital parts was possible in the past. Today's complex ICs have analog and digital functionality tightly integrated throughout the whole design at different levels of hierarchy, and cannot be verified separately [5].

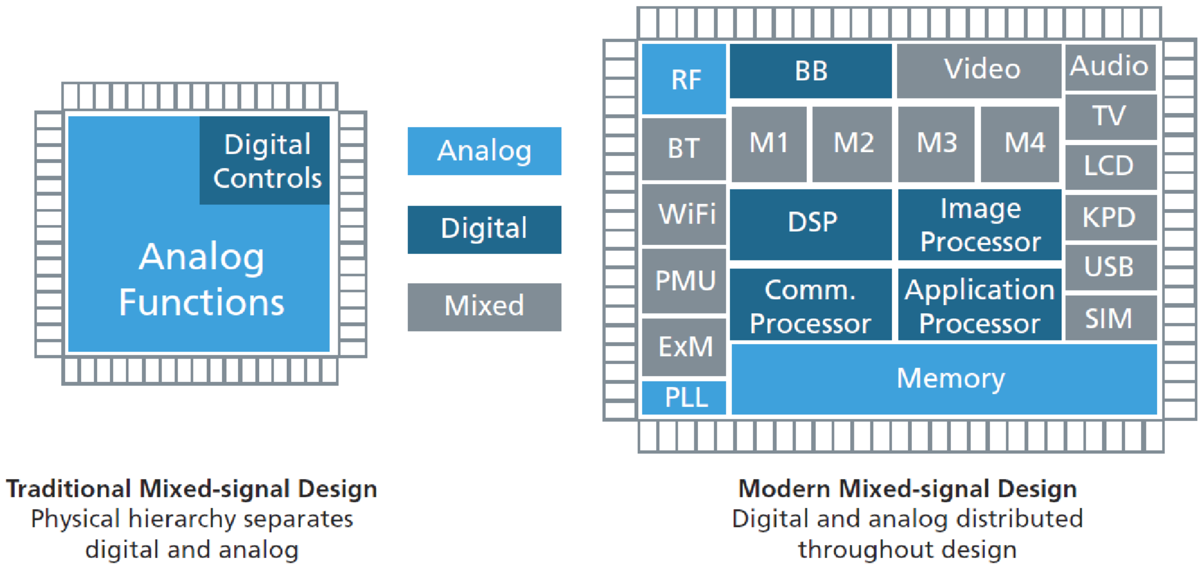


Figure 1: Complexity of mixed-signal design [2].

In the classic analog world, verification is performed using SPICE simulators at a detailed transistor level and is usually done in a bottom-up fashion. This means implementation of individual blocks from their specifications using transistor-level representation and their ensuing isolated verification to match specific verification goal. Thus verified block is then integrated with similarly verified blocks and the integration process proceeds from block to progressively higher levels of integration. This approach works quite well when design size is small, as design size and complexity grows and design characteristics start changing from pure analog to mixed-signal, a bottom-up methodology shows severe limitations. The most obvious being the increasing cost of resources needed to perform simulation at the detailed transistor level and the lack of methodology to integrate block-level verification tasks with system-level specifications [5].

On the other hand, the digital verification approach is essentially top-down and is driven by a chip-level verification plan that causes the verification process to start at an early stage of design. Such verification plan guides the simulation planning as well as the levels of models required at each stage of integration. The state space of the design is effectively explored by directed random metrics and tests, such as functional coverage and provides feedback regarding how much of the verification plan has been exercised by the existing regression suite [5].

1.1.1 Analog and digital simulation

Analog and digital simulations used as a basis of verification are fundamentally different. Analog signals can change in almost infinitely small increments of terms of time and amplitude. During transient analysis, analog simulators are tasked with solving a set o matrices at every time step. Each element in the design can have an instantaneous influence on any other element in the matrix.

The typical analog simulator breaks the time axis into small time steps and then calculates the equation solution that describes what should happen over each step. Then the simulator decides how big time step it can safely take and it must iterate and converge toward a solution that solves the Kirchhoff's laws at the new time point [1], [3].

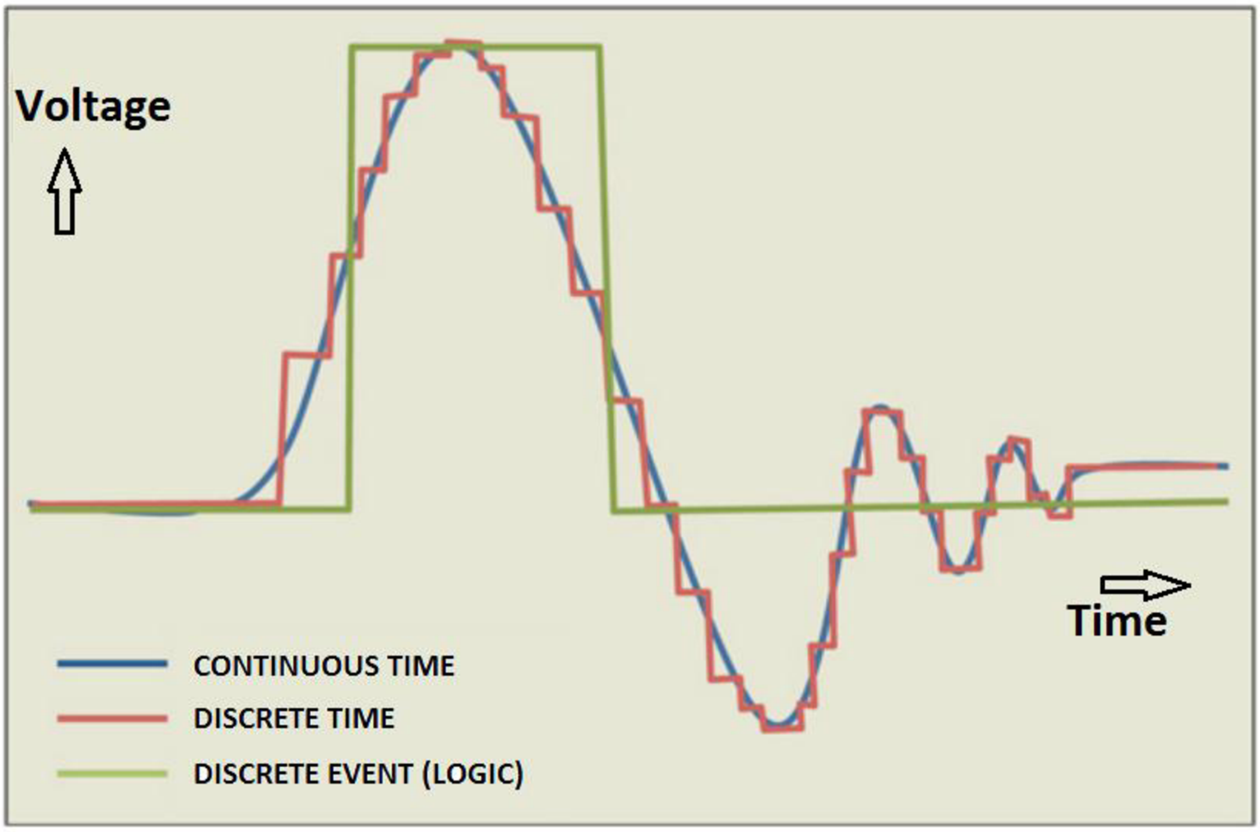


Figure 2: Comparison of various sampling methodologies[3].

The behavior of digital circuits is described by Boolean relations. Digital simulators solve logical expressions sequentially by triggering events and do not require an iterative nonlinear equation solver. They are therefore much faster than their analog counterparts [1].

1.2 Challenges of mixed-signal verification

As complexity of ICs increases, the verification task is growing rapidly. The main challenge in verifying today's mixed-signal designs is that traditional direct test verification methods are becoming much harder to apply [1].

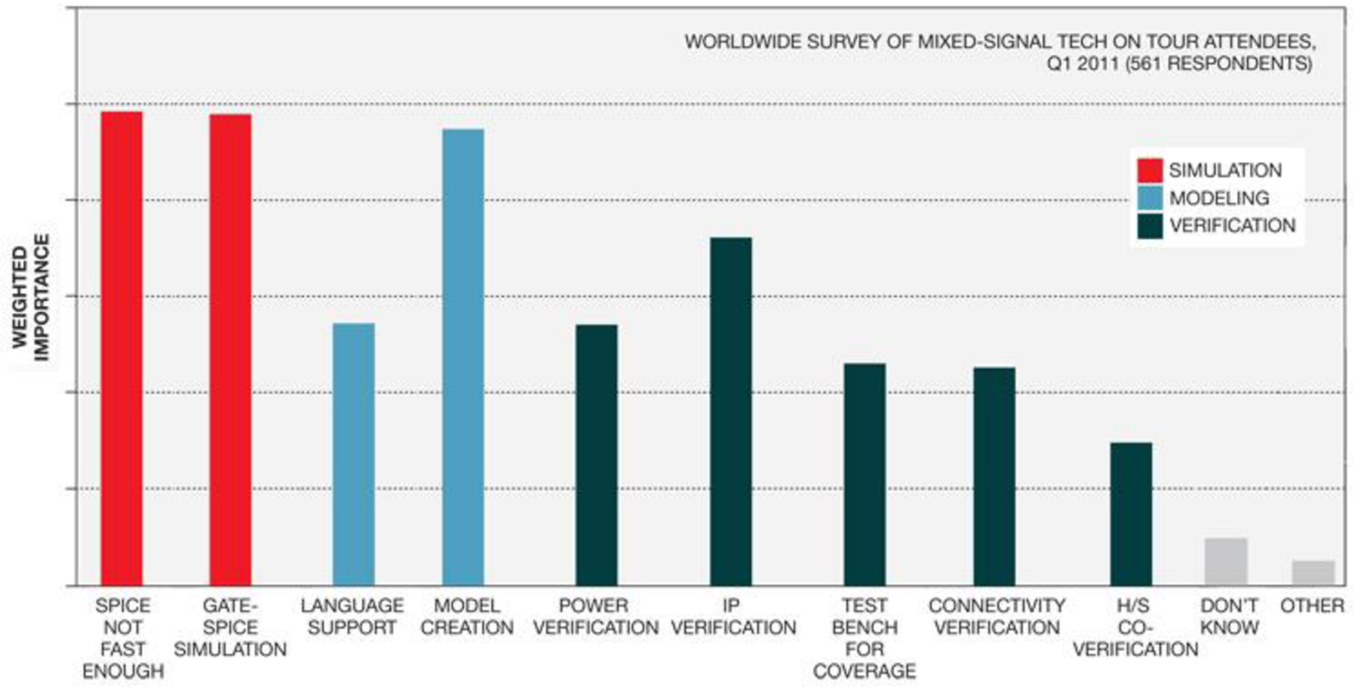


Figure 3: Results of a 2011 survey showing biggest mixed-signal verification challenges [3].

As shown in Figure 3, analog simulation, as a component of the mixed-signal verification, is a major bottleneck. Advancements in SPICE simulation, such as Fast-SPICE, provide additional speed and capacity at the cost of some accuracy however a single simulation run could take days even with the fastest simulator.

To tackle the poor performance of SPICE, many mixed-signal teams are turning to analog behavioral modeling. This approach can increase simulation speed, but the creation of good models can be challenging.

A 2011 Design Automation Conference (DAC) panel [6] discussed the need for analog design and verification to become more like digital, more structured, and more top-down. Debug methodologies such as Assertion-based verification (ABV), Metric-driven verification (MDV) and Universal Verification Methodology (UVM) need to be introduced for analog and mixed-signal designs [5].

2 MODERN MIXED-SIGNAL VERIFICATION METHODS

In order to properly verify today's complex mixed-signal ICs, several verification techniques have been recently introduced into mixed-signal world.

2.1 Behavioral modeling

Behavioral modeling is a key component in a mixed-signal verification methodology. Describing analog and mixed-signal blocks in a higher level of abstraction makes mixed-signal simulation more effective. Since creation of models is not a simple task, there are several challenges:

- The intended purpose and scope of the model must be well understood and suitable model architecture/template chosen. In a top-down methodology, models are developed before circuits are available and for functional verification at the system level a simpler model might be sufficient. In bottom-up approach, the model might need to match an already implemented block for performance verification, and thus a more accurate model is used.
- The model must be validated to make sure that it is sufficiently equivalent to circuit or specification with required accuracy.
- The model must be kept in sync with changes in the circuit or specification.
- The model needs to be written in way that does not cause convergence issues during simulation.
- Modeling is hard to automate and typically requires specialized engineering talent. Model creation requires an understanding of analog and mixed-signal simulation algorithms, knowledge of analog and mixed-signal circuits, design techniques, coding and debugging.

Analog and mixed-signal modeling has a wide range of possible features to model. Depending on complexity, development of a model can take from minutes to months, and the simulation can run a rate slower than the transistor-level design to a million times faster. There is no single correct modeling approach, but there are areas where poor decision in modeling dimensions can result in models, that are not suitable for intended tasks [4].

2.1.1 Types of modeling

It is common, to use variety of several modeling formats during verification of a large IC. Typical formats are:

- **Device based design (Spectre, SPICE)** - schematics built using process-specific devices is the standard transistor-level design technique. A macromodeling approach that uses generic elements and dependent source to define simple block operation can also be used.
- **Analog modeling (Verilog-A)** - defines analog description of relations between current/voltage.
- **Mixed-signal modeling (Verilog-AMS, VHDL-AMS)** - allows description of both analog and digital behavior for corresponding portions of the block.
- **Discrete real number modeling (Verilog-AMS, VHDL, SystemVerilog)** - models analog block operation as discrete real data. Typically ignores impedance effects.
- **Logic modeling (Verilog, VHDL, SystemVerilog)** - model defines discrete logic data flow, ignores analog operations [3].

Figure 4 shows the tradeoff between simulation accuracy and performance among SPICE, Fast-SPICE, analog behavioral models (Verilog-A, Verilog-AMS, VHDL-AMS), real number models and pure digital simulation. These numbers can vary significantly for different applications. SPICE level simulations are used as a golden reference simulation, analog behavioral modeling provides wide range of accuracy and performance. Digital models may be sufficient for verification tasks like connectivity checks and real number models provide high simulation performance with restricted accuracy [2], [7].

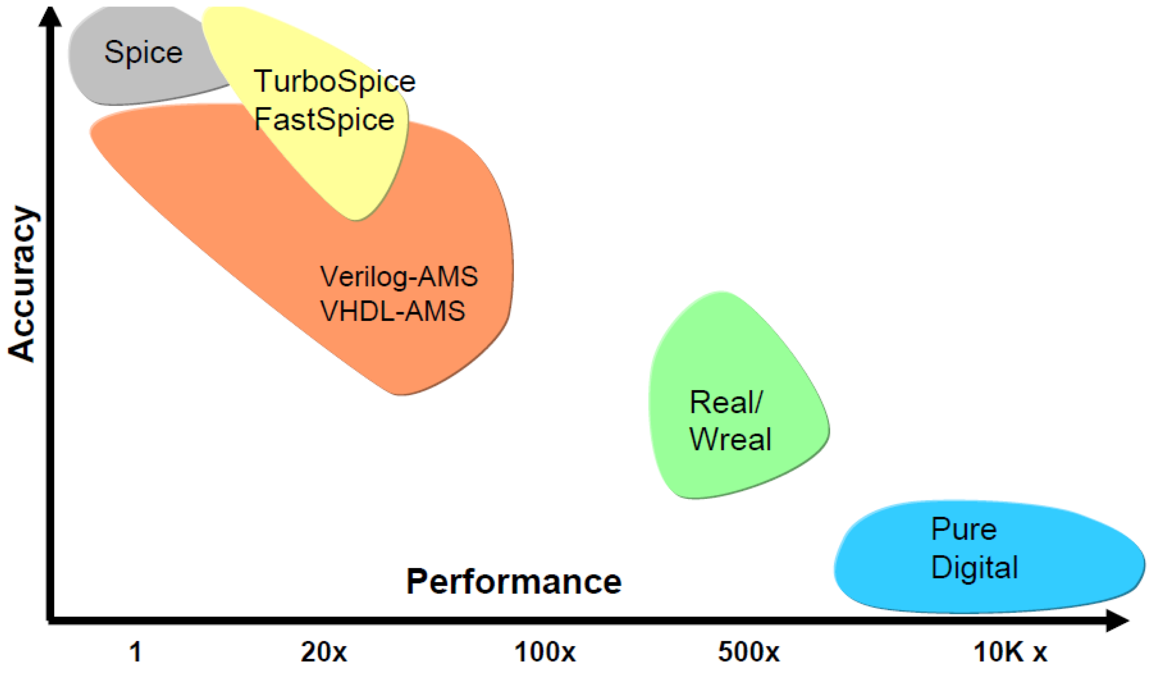


Figure 4: Model accuracy vs. performance gain for mixed-signal simulation [7].

Another important factor is the required effort to set up a simulation and create the model. Figure 5 illustrates the general trends. Although SPICE simulations run slowly, they are easy to set up. Analog behavioral model creation effort can range from hours to weeks, Real valued models are inherently restricted to the signal flow approach and analog convergence is not an issue. Consequently, the modeling effort is significantly lower compared to analog behavioral models and the same applies for pure digital models [2], [7].

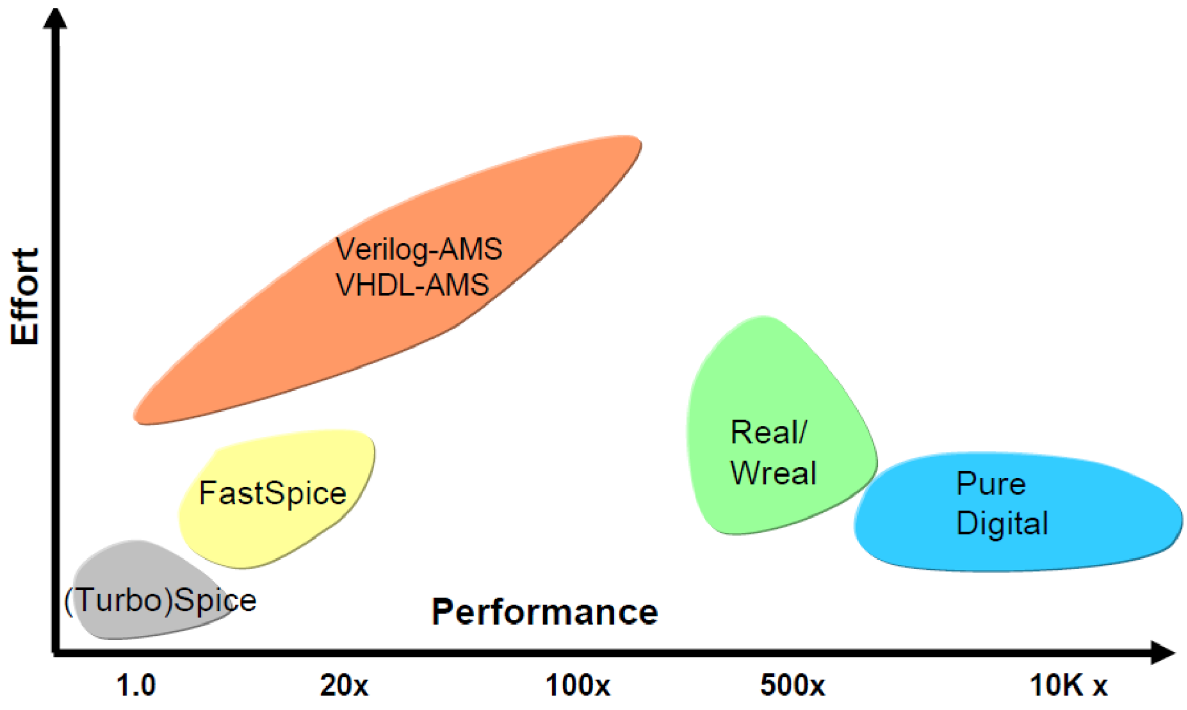


Figure 5: Required effort vs. performance gain for mixed-signal simulation [7].

To choose what types of models should be developed, it is important to understand the purpose, capabilities and limitations of each style of modeling.

2.1.2 Discrete digital modeling

Pure digital solvers can be used to model the digital input/output characteristics of a system. Available languages include Verilog, SystemVerilog and VHDL. This approach does not handle analog signals, but is extremely efficient at handling logic and timing relationships using a discrete event simulation kernel. It is commonly used for pure digital modeling and for black-boxed analog subsystems, where only the digital operations are modeled. The discrete event simulation approach can be extended to model analog signals as discrete values (Real number modeling) [4].

2.1.3 Continuous analog modeling

Pure analog languages (SPICE-like) can be used to model the electrical nature of a system. Verilog-A is the standard language for analog behavioral modeling. The language creates a description of the interrelationships between voltages and currents in the system. Impedance characteristics along with integral and derivative dependencies can be written directly. Verilog-A model is converted into a set of simultaneous equations (nonlinear, ordinary differential equations) suitable for a simulator. The built-in models for transistors and other components are also defined internally to create sets of equations in a similar format. During transient analysis, matrix-based numerical analysis techniques are used to solve the complete set of voltage and current equations at each analog time step.

Well-defined analog models can result in a speedup, in the range of 10x to 50x, compared to transistor-level models. Simulation speed depends on the size and complexity of the equations to be solved and additionally on the time step used in the transient simulation. The performance increase from usage of behavioral models is based on the reduction in the number of equations and nodes in the system and the ability to take larger time steps due to fewer lower-level nonlinearities in the system.

If logic signals are present in the analog model, the must be converted to electrical signals that swing between defined voltage levels with specified rise and fall times. Analog simulations of very active logic networks often simulate relatively slowly due to the small time steps required during each logic transition. A simulation performed using separate analog and digital simulators using Inter-Process Communication (IPC) between the simulators can suffer from the same speed problem, because all logic signals must be converted to analog waveforms before usage on the analog side of the co-simulation environment, resulting in similar small time step issues [4].

2.1.4 Mixed-signal modeling

Mixed-signal simulation combines the analog continuous time and discrete digital solvers within a single simulator. For model description, modeling languages Verilog-AMS and VHDL-AMS can be used. Mixed-signal languages allow the most natural modeling of mixed-signal systems, since the analog parts can be modeled with the standard electrical modeling approach, while the digital portions can be modeled using discrete modeling techniques. Data and events are transparently passed between the two simulation algorithms. This is also the preferred language for writing mixed-signal testbenches. Verilog-AMS code can be used to write procedures that read both analog and digital quantities, making it an optimal environment for mixed-signal verification testbench development. Real number modeling techniques can also be used in these languages.

By using well-defined AMS models, simulation speedup depends on the amount of transistor-level circuitry being replaced. Employing AMS techniques removes the digital circuitry from analog simulation engine, and the replacement of the remaining analog circuits with AMS operations can speed up the analog portion by a factor of 10x to 50x [4].

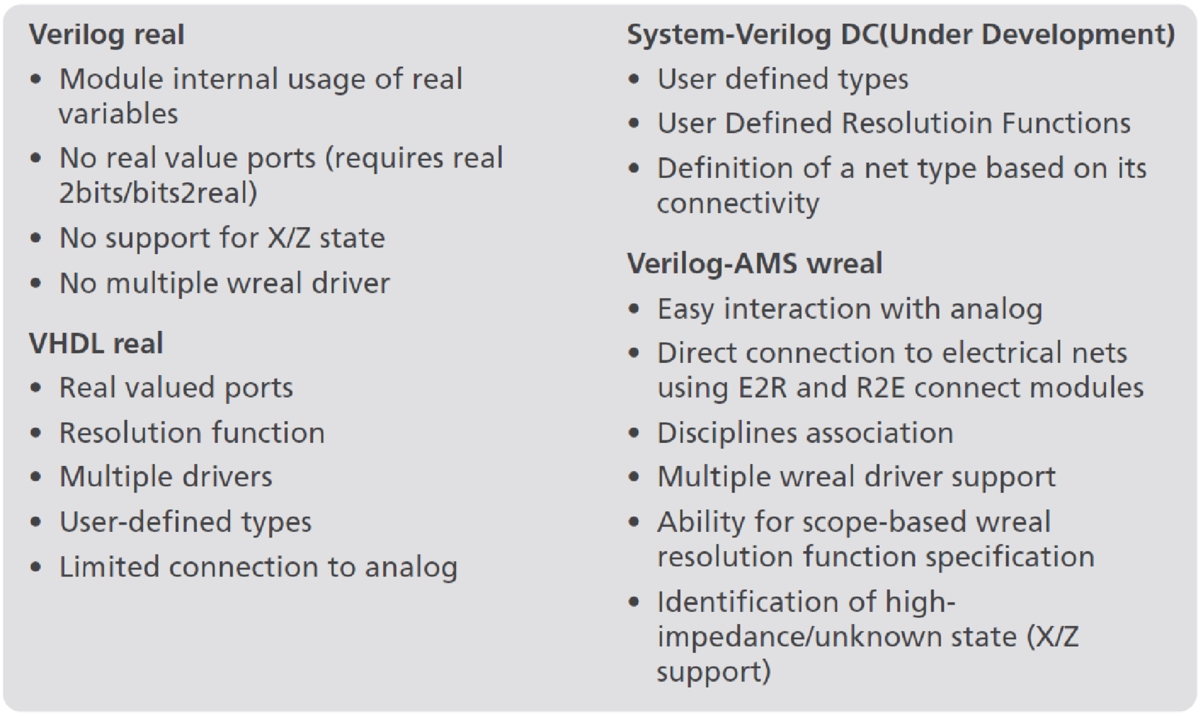
2.1.5 Real number modeling

Real number modeling (RNM) is a special technique which models electrical signals by representing them as a time-varying sequence of real values. In a typical analog simulator, the models define a set of equations which the simulator augments via addition of topology constraints using Kirchhoff's laws, and then it solves the overall constrained system of simultaneous equations at each time step to compute the voltages and currents from those equations. In a discrete real environment, there are no voltage/current equations, no Kirchhoff's laws, and no simultaneous equation solution step. The output is directly computed from the input, ignoring currents and feedback mechanisms that could have caused interdependencies between drive and load in electrical environment [4].

The concept of RNM is straightforward. If the input/output relationship is a direct transfer characteristic, a mathematical expression can be written that describes how to update the output whenever input changes. Checking for proper biasing is also simple. The power supply, bias current and voltage inputs would be passed into the model as real numbers and the simulator would check if they are within reasonable tolerance. The outputs would only be driven if the proper bias and control are applied [4].

It is already a common practice to verify subsystems at the transistor level, and then use behavioral models in higher-level simulations, so it is a natural extension to create that behavioral model using RNM rather than AMS modeling techniques [4].

Many languages support RNM including Verilog, SystemVerilog, VHDL, and Verilog-AMS. The first three support a real data type, while Verilog-AMS supports real-wire or wreal. Verilog-AMS is more advanced in the area of connect modules, while VHDL is slightly more flexible in terms of resolution function [2], [7].



Verilog real
- Module internal usage of real variables
- No real value ports (requires real 2bits/bits2real)
- No support for X/Z state
- No multiple wreal driver

VHDL real
- Real valued ports
- Resolution function
- Multiple drivers
- User-defined types
- Limited connection to analog

System-Verilog DC(Under Development)
- User defined types
- User Defined Resolutioin Functions
- Definition of a net type based on its connectivity

Verilog-AMS wreal
- Easy interaction with analog
- Direct connection to electrical nets using E2R and R2E connect modules
- Disciplines association
- Multiple wreal driver support
- Ability for scope-based wreal resolution function specification
- Identification of high-impedance/unknown state (X/Z support)

Figure 6: Language support for real number modeling [2].

2.1.6 Combined approaches

When working in an AMS environment, it is common to develop models that use a combination of techniques. For example, an RF receiver could be modeled using RNM techniques for the RF signal path, electrical for the baseband signals, biasing and power supplies, and discrete logic for control signals. Such approach has a benefit of reasonable simulation times due to the high-speed signal processing and digital control performed in the discrete environment, along with easy interface to transistor-level subsystems from the analog baseband and bias connections [4].

2.2 Assertion-based verification

By definition, an assertion is a check against the specification of a design that captures the intended behavior. Assertion-based verification is a powerful verification approach, by which can designers verify their designs by writing the assertions into blocks to test whether the blocks work correctly in common scenarios. They act as monitors during simulation, detecting errors close to their source and reporting both errors and coverage information [5], [8], [9].

Assertions are written during development of the design and the verification environment. Both designers and verification engineers can be involved in identifying requirements and capturing them as assertions. Through the use of assertions, verification can start earlier while detection and removal of bugs is faster. Also in contrast to traditional way of eye-balling waveforms and tracing them back to failure, graphical assertion browser leads to quicker identification of bugs as shown in Figure 7 [5], [9].

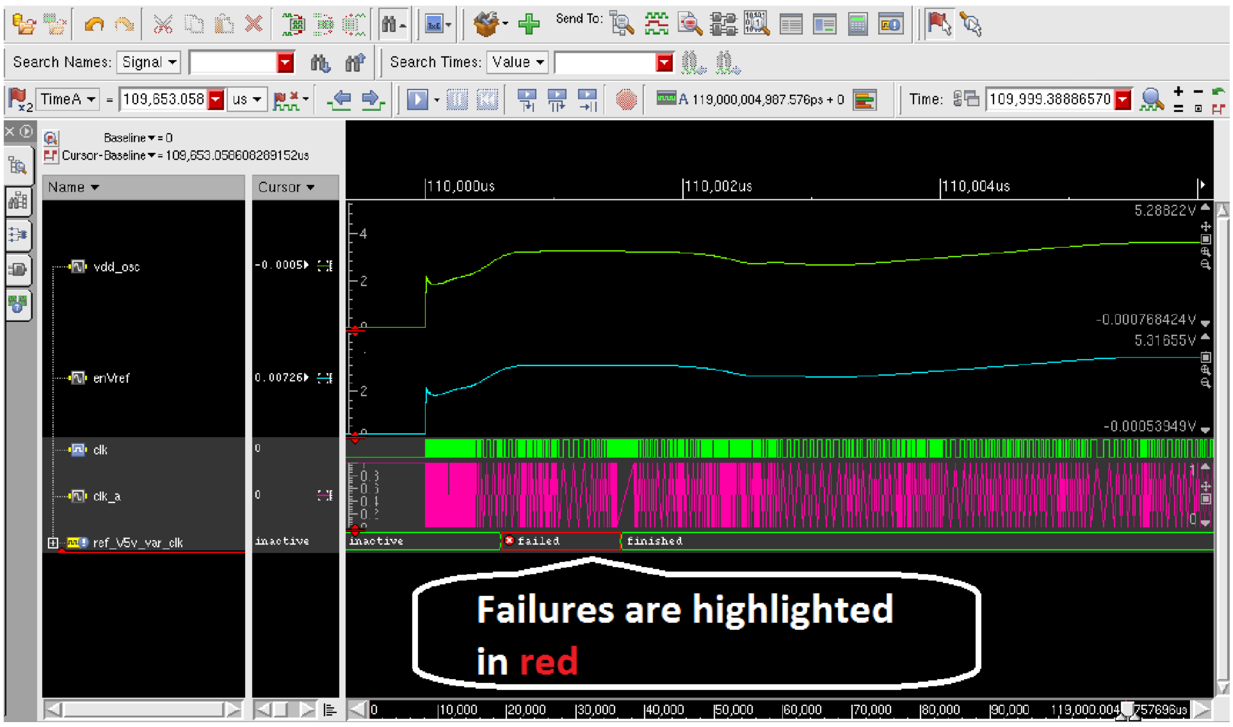


Figure 7: SimVision assertion browser shows failed assertion [12].

In the digital world, assertion-based verification is a well established methodology that has evolved to meet the needs of logic designers. It is based on standard assertion languages such as PSL and SVA. Assertions can for instance provide a formal framework for: [5]

- Checking a set of behavior of a signal that must occur independently on time.
- Checking a behavior of a signal that must occur within a certain time frame.
- Checking boundary conditions that must trigger a set of behavior.
- Specifying values or sequences that would describe an error condition.
- Watching signal value of a certain signal critical to the functionality of the design.

2.2.1 Assertions in mixed-signal space

Certain form of assertions already exists in the analog domain. The SPICE Device Operating Condition Checkers can be used to set a custom characterization check that specifies the safe operating conditions of the circuit. These checks are useful to verify the device-level characteristics but cannot be used for verification of more complex circuit conditions [10].

Major difficulties in verification of analog or mixed-signal systems are:

- Absence of a consistent language and methodology to express the verification intent in the form of assertions across the spectrum of continuous, discrete event-driven and mixed-signal systems.
- Information expressed by one group of design or verification engineers, in the analog/mixed signal domain, does not flow easily to another group, or from one level of design abstraction to another.
- Absence of a standard verification plan which would include analog or mixed-signal blocks. It is not possible to combine items tested in isolation, with the same items in context of the complete system. This challenge includes verifying aspects like current leakage, power sequences or noise figures from respective blocks in a context of full system.

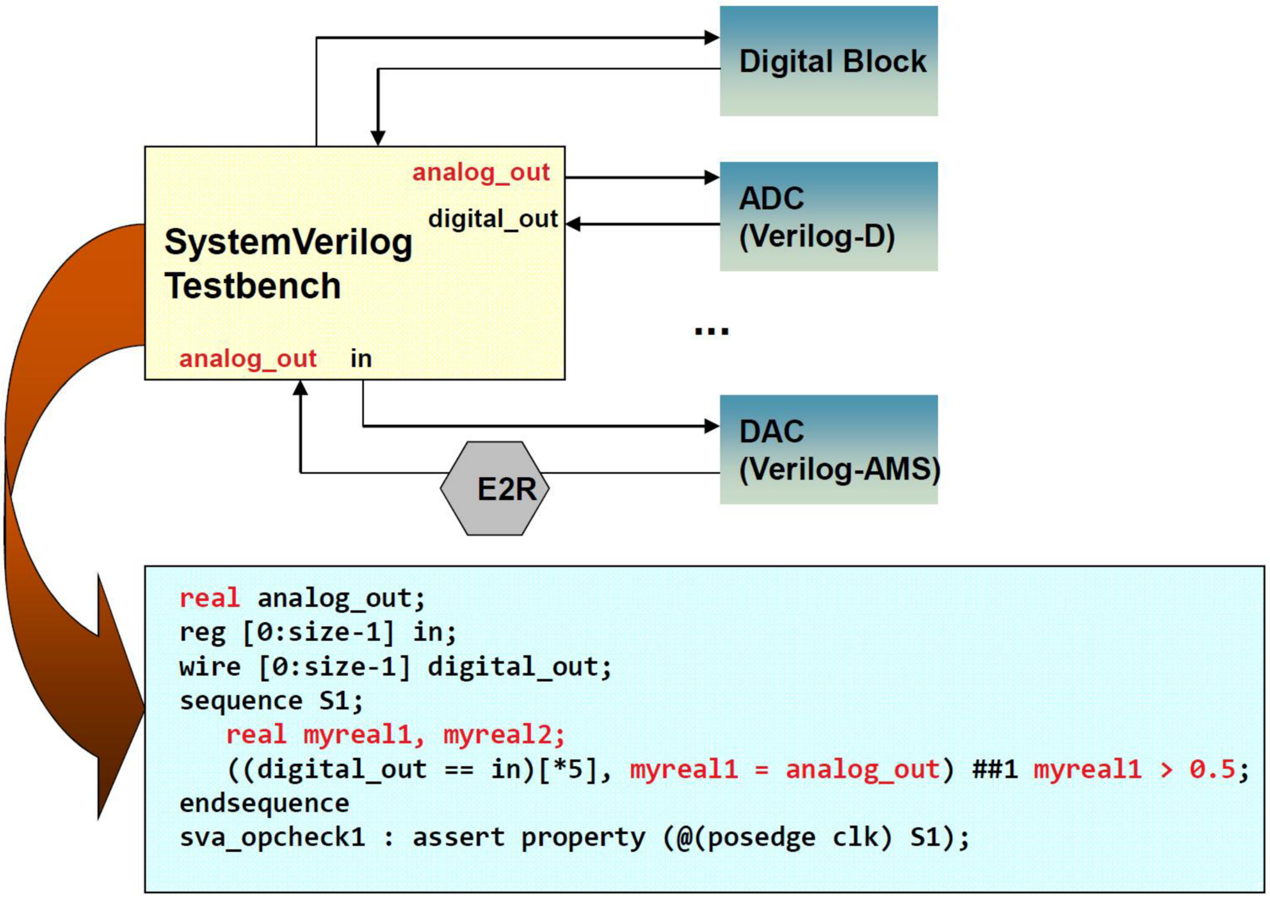
In a view of the challenges mentioned above, it is natural to attempt to apply the well established concepts of assertions from digital space into analog and mixed-signal domains. Two standard groups are working towards standardizing analog/mixed-signal assertions [5]:

- The analog System Verilog Assertions committee is focused on analog/mixed-signal extensions to the SVA subset of the SystemVerilog language.
- The SystemVerilog-AMS (SV-AMS) committee is working on alignment of Verilog-AMS with the SystemVerilog into new SystemVerilog-AMS standard [13].

2.2.2 SystemVerilog Assertion SVA

SystemVerilog Assertions (SVA) is a legal subset if the SystemVerilog P1800-2012 standard. SystemVerilog deals with discrete logic data flow and does not allow the presence of continuous domain object (2.1.1). However, recent extensions to SystemVerilog allow the usage of real data types (2.1.5), which may be used to connect real valued ports to the electrical domain by inserting „electrical to real“ connect modules as shown inFigure 8 [5], [10].

Usability of SystemVerilog Assertion is restricted to digital and real net types. An analog quantity of interest (node voltage, current) has to be converted into real data type in order to perform a simulation. A major advantage of this is the ability to use the same testbench with different model types. The same assertion that can be used to check a Verilog model of an analog block can be used later on in the design when SPICE netlist is available [15].



```
real analog_out;
reg [0:size-1] in;
wire [0:size-1] digital_out;
sequence S1;
   real myreal1, myreal2;
   ((digital_out == in)[*5], myreal1 = analog_out) ##1 myreal1 > 0.5;
endsequence
sva_opcheck1 : assert property (@(posedge clk) S1);
```

Figure 8: SystemVerilog Real variable connecting to Verilog-AMS electrical domain [12].

2.2.3 Verilog-AMS PSL

Writing assertions in Verilog-AMS is possible through Property Specification Language (PSL). Unlike SVA, Verilog-AMS PSL Boolean expressions can contain analog expressions. They can appear in clocking expressions and as arguments in property and sequence instances, when there is a single top-level clock defined [5], [10], [12].

```
electrical int_node, int_node2;
reg clk;
...
...
// psl mixed_signal_check:
// assert always (clk -> next(V(int_node2) < 0.6))
@(cross(V(int_node) - 1.25));
```

Figure 9: Assertion containing analog expression in Verilog-AMS PSL [12].

Verilog-AMS PSL also supports real number models (2.1.5) which are in Verilog-AMS represented by Wreal data type. Expressions involving wreal objects can appear within PSL assertions in Boolean expression, clocking expressions and as arguments in property and sequence instances [5], [12].

2.2.4 PSL vs SVA

PSL and SVA have similar capabilities, assertions written in either language are sufficient to describe a set of behavior in analog/mixed-signal blocks.

PSL:

```
assert always (
  {req && ack} |=>
  {!req within gnt[->1]}
)@(posedge clk);
```

SVA:

```
always @(posedge clk)
assert property (
  req && ack |=>
  (!req within gnt[->1])
);
```

Figure 10: A simple assertion in both PSL and SVA [16].

Figure 10 shows a simple assertion written in both languages. The assertions are evaluated according to the rising edge of clk. If req and ack are both true, then req must be false until the first time point at which gnt is true [16].

SVA is tightly tied into SystemVerilog and as a result, inherits its expression language including data types, expression syntax and semantics. SVA can be also written directly into SystemVerilog design. PSL is a separate language designed to work with many HDLs and their expression layers. Unlike SVA, it cannot be written directly into designs, but can be attached to HDL models using binding directives (2.2.5) [16].

The tight coupling of SVA with SystemVerilog means that assertions can be written to interact with other testbench components without crossing the boundary of a programming language interface. The failure or passing of an assertion can be defined to trigger an execution of a specific block of SystemVerilog code which may call an error handling task, update a testbench coverage database or influence the heuristic parameters of a reactive self-adaptive testbench [16].

PSL has a structure of multiple abstraction layers and a rich set of operators that can be used at different levels of abstraction. As a result, PSL provides the capability to write assertions that range from system-level down to RT level. To summarize - PSL is a multi-purpose, multi-level assertion language while SVA is tightly connected to SystemVerilog. Both PSL and SVA have similar capabilities and both can be used in mixed-signal verification.

2.2.5 Module bound verification units

Module bound verification unit (vunit) is an auxiliary file that is linked to the design file for simulation. Vunits are mainly used to store PSL assertion code. They can also be used for storing values in variables/registers. Vunit is a useful feature if the source text of the design block should not or cannot be modified.

Verification units can be used to add assertions to Verilog, Verilog-AMS, SystemVerilog and VHDL instances.

2.3 Metric-driven verification

Metric-driven verification (MDV) is a verification methodology originally used for verification of large digital designs, where huge amount of state spaces made simulation of all possible combinations impossible. MDV helped to achieve good functional verification by coverage-directed random stimulus generation. As functional complexity of today's designs increases, MDV is being adapted for use with analog and mixed-signal circuits [5], [6].

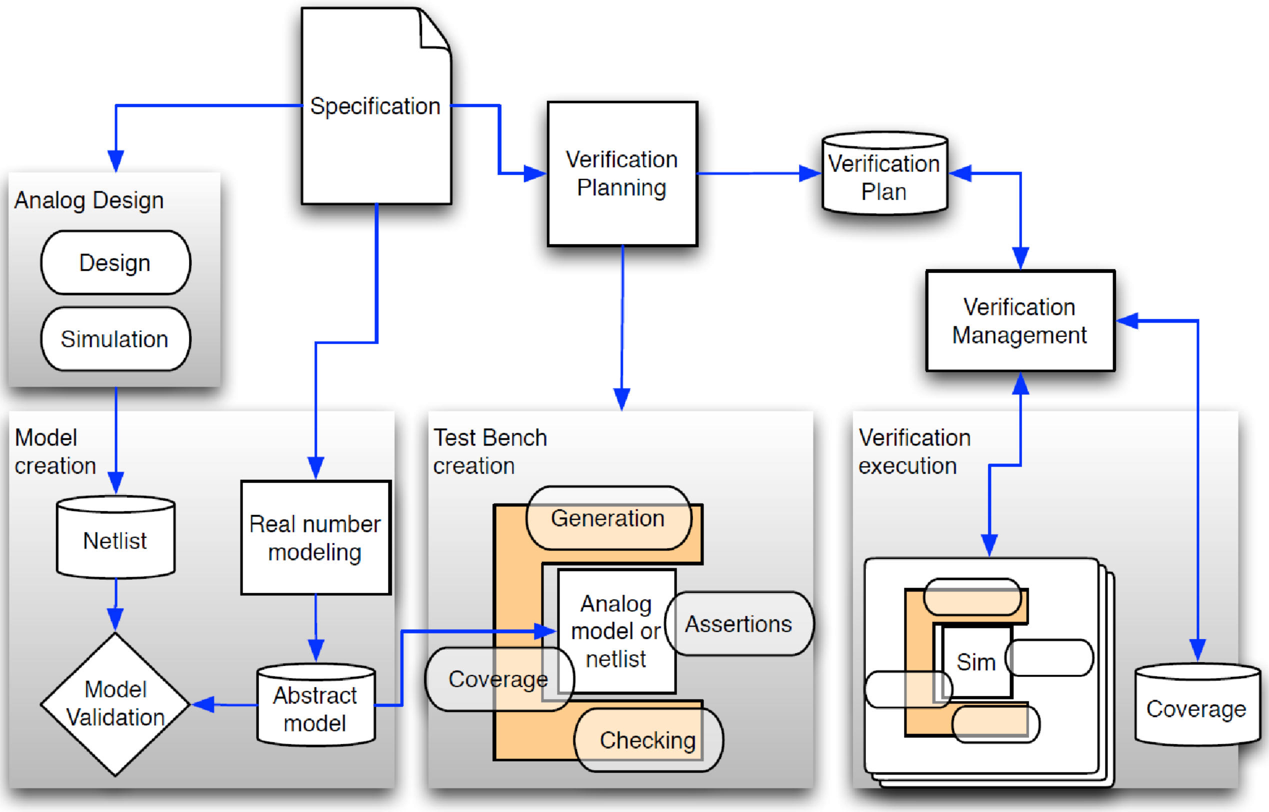


Figure 11: Verification flow for analog IP [16].

The concept of MDV methodology is shown in Figure 11. MDV is based on a verification plan. Verification plan outlines the testing scenarios, coverage metrics and specifies which features should meet the specification by measurement. These measurements are called functional coverage. A testbench with automatic stimulus generation is created to check the functionality and measure coverage of a design. For best verification performance, the analog circuit should be modeled as a real number model (2.1.5). The methodology can also be applied to Verilog-AMS models or SPICE netlists.

3 PRACTICAL PART

Aim of the practical part was to choose one of the mentioned descriptive languages, and create assertions for verification of power supply control circuit or its blocks. All assertions were simulated in Cadence SimVision 10.20 using the Virtuoso AMS Designer Simulator.

3.1 PSL and SVA assertions

To decide which descriptive language is more suitable, a simple assertion written both in PSL and SVA was tested.

Assertion below tests, that voltage on selected node stays within specified range, when controlling logic signal is asserted. First part of the Verilog-AMS code defines input ports. Port *in_v* is of type electrical and is meant for connection of monitored voltage. Port *in_l* is for connection of controlling logic signal. Parameters *v_max* and *v_min* specify desired voltage range. Next part defines 500 MHz clock whose rising edge is used for assertion evaluation. Final part is the PSL assertion itself.

```
//Verilog-AMS HDL for "cn19Proj", "assertion_1" "verilogams"

`include "constants.vams"
`include "disciplines.vams"
`timescale 1ns / 10ps

module assertion_1 ( in_v, in_l );

input in_v, in_l;
electrical in_v;
logic in_l;

parameter real v_max = 5.1;
parameter real v_min = 4.9;

///////////////////////////////////////////////
reg clk;
initial clk=0;
always #1 clk=~clk;
///////////////////////////////////////////////
//psl default clock = (posedge clk);
//psl voltage_check: assert always
//    ((in_l == 1) -> ((V(in_v) >= v_min) && (V(in_v) <= v_max)));
endmodule
```

Following code in SVA describes the same assertion. This time, port *in_v* is of type real, because System Verilog does not allow presence of continuous domain objects.

```
//systemVerilog HDL for "cn19Proj", "a1SV" "systemVerilog"
`timescale 1ns / 10ps

module a1SV( in_v, in_l );

  input in_v, in_l;
  real in_v;
  logic in_l;

parameter v_max = 5.1 ;
parameter v_min = 4.9 ;
///////////////////////////////////////////////
reg clk;
initial clk=0;
always #1 clk=~clk;
///////////////////////////////////////////////
v_check_SVA: assert property(
  @(posedge clk)
     ((in_l == 1) |-> ((in_v >= v_min) && (in_v <= v_max)))
   );

endmodule
```

Although both assertions are the same, simulation results differ. Figure 12 shows monitored voltage (*in_v*) and controlling logic signal (*enable*). PSL assertions within Verilog-AMS test block make evaluations with voltage values sampled every positive edge of the defined clock. SVA assertions make evaluations with converted real values. This converted wave (*in_Real*) slightly differs from the original (*in_v*). Because of these conversion inaccuracies, assertion states are detected several clock cycles late, or not detected at all. This is obvious from Figure 12 at time 3.5 µs, where PSL assertion detects failure but SVA does not.

Simulation results show three types of resulting assertion states. „Inactive" state means, that specified first condition (*enable* == 1) was not met and assertion was not evaluated. States „finished" and „failed" indicate, whether voltage stayed within specified range.

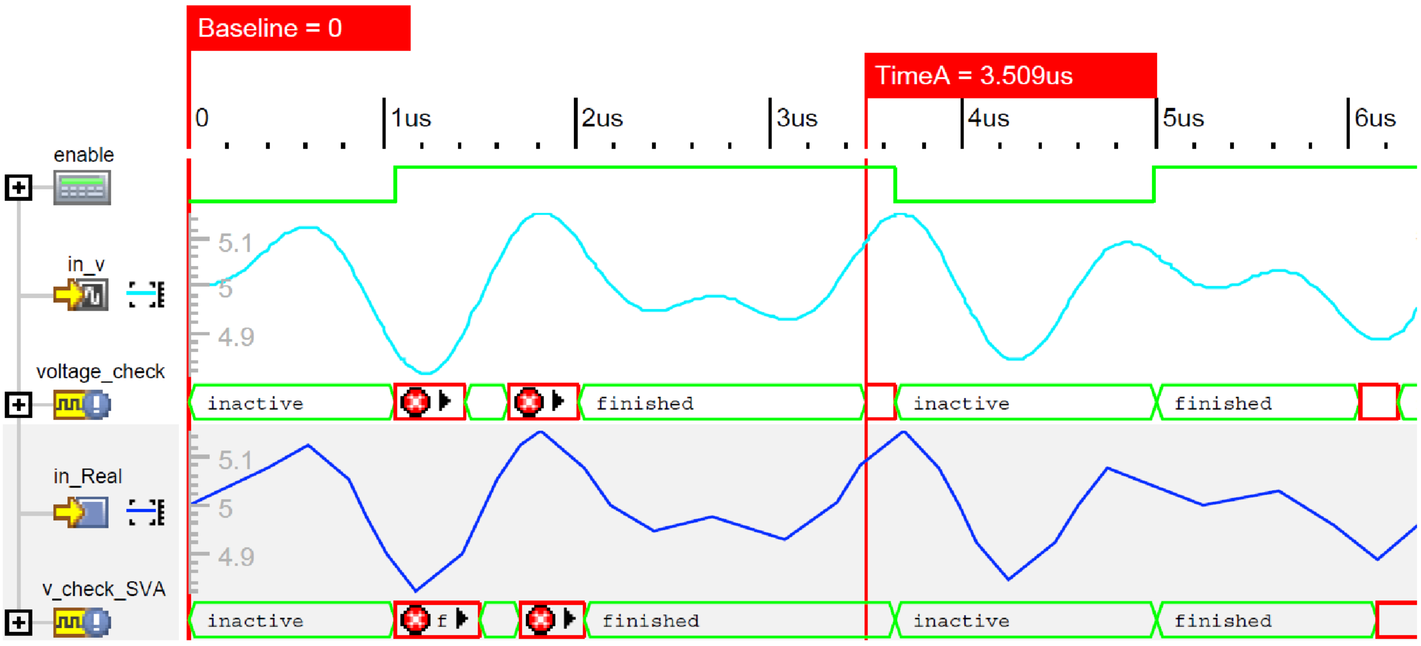


Figure 12*: Simulation results of PSL and SVA assertion in SimVision.*

The conversion inaccuracies could be probably suppressed by editing the electric to real conversion modules. Due to the fact that Cadence Verilog-AMS PSL supports analog expression arguments in property and sequence instances, Verilog-AMS PSL seems like more suitable choice for the purpose of mixed-signal verification. For this reason, all following assertions are created in this language.

3.2 Assertion clocking

This chapter mentions three basic approaches in assertion clocking.

3.2.1 Defined clock

The most basic form of assertion clocking is using a system clock. If there is no clock (analog circuits) , clock for assertion evaluation can be explicitly declared in a Verilog-AMS code, as in chapter 3.1. Values of voltages and currents are then sampled every positive or negative clock edge.

The obvious drawback of assertions clocked this way is, that it cannot detect glitches that occur for time intervals shorter, than period of defined sampling clock. For instance a 1 MHz sampling clock (1 µs period), would unlikely detect a 20 ns error. Increasing the clock frequency would lead to better accuracy, but it would also slow down the simulation.

3.2.2 Analog event function

Verilog-AMS supports function *cross* as clocking event.

cross (expression , direction, time_tolerance, expression_tolerance)

The *cross* function is used for generating a monitored analog event. Event is generated each time the expression crosses zero in the specified direction. The direction can only evaluate to +1, -1, or 0. If set to 0, or not specified, event will occur on both signal crossings. If set to +1, event occurs on rising transition, if -1, then the event occurs only on falling transitions of the signal. For example *cross(V(in) - 2.5, +1))* generates event every time the voltage at node *in* crosses 2.5 Volts in a positive direction [17].

This approach should have minimal effect on simulation time. The drawback is that every crossing has to be explicitly declared, which may be laborious in more complex assertions with several signals.

3.2.3 Variable clock

Variable clock can be created using *absdelta* event function in Verilog-AMS.

absdelta (expression , delta, time_tolerance, expression_tolerance)

The *absdelta* function generates event every time the expression changes more than delta ± expression tolerance, relative to the value at previous event time. Time tolerance specifies a time increment after the previous time point. No event is generated when the current time is within tolerance of previous event time. Specified time tolerance that is smaller than time precision of the simulation is ignored and the time precision is used instead [17].

Example of variable clock code is in Appendix B, this method of assertion clocking has great accuracy with reasonable impact on simulation time.

3.3 Examples of created assertions

This chapter contains examples of several types of PSL assertions that were used in verification of power supply control circuit block.

3.3.1 Voltage level monitor assertion

This assertion is similar to those mentioned in chapter 3.1. Difference is, that instead of using defined clock for evaluation, Verilog-AMS analog event function *cross* is used.

```
//Verilog-AMS HDL for "cn19Proj", "v_level_ena" "verilogams"
`include "constants.vams"
`include "disciplines.vams"

module v_level_ena ( in_v, in_l );

input in_v, in_l;
electrical in_v;
logic in_l;

parameter real tresh = 0.1;
parameter real v_max = 5.1;
parameter real v_min = 4.9;
//////////////////////////////////////////////////
//psl voltage_check: assert always
//    ((in_l == 1) -> (V(in_v) >= v_min) && (V(in_v) <= v_max))
//     @(cross(V(in_v) - tresh, 0) or cross(V(in_v) - v_min, 0) or
//        cross(V(in_v) - v_max, 0) or in_l);
endmodule
```

For monitoring voltage level with controlling logic signal, assertion needs to be evaluated when voltage crosses upper or lower tolerance value or when controlling logic signal changes. This is ensured by adding *cross* functions for these cases.

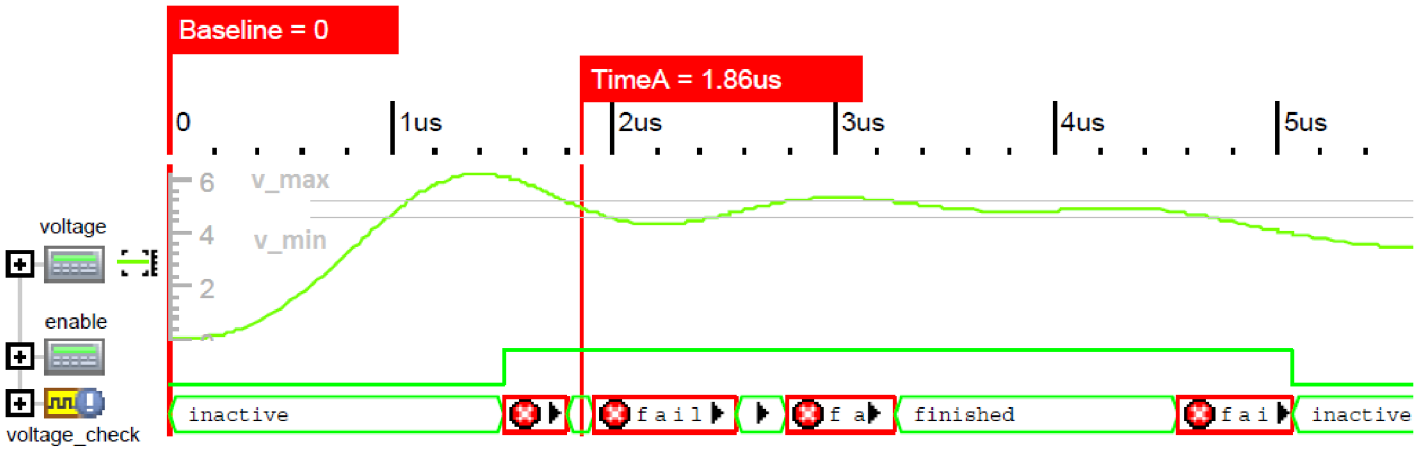


Figure 13: Voltage level monitor - simulation result in SimVision.

3.3.2 Voltage difference assertion

This assertion monitors voltage difference between two nodes. When this difference exceeds value set in parameter *delta_vmax*, assertion fails, as can be seen in Figure 14.

```
//Verilog-AMS HDL for "cn19Proj", "delta_v" "verilogams"
`include "constants.vams"
`include "disciplines.vams"

module delta_v (in_v1,in_v2 );

input in_v1, in_v2;
electrical in_v1, in_v2;

parameter real delta_vmax = 200m ;
///////////////////////////////////////////////
//psl delta_v_check: assert never
// (( v1 - v2 > delta_vmax ) || ( v2 - v1 > delta_vmax ))
// @(cross((V(in_v1) - V(in_v2)) - delta_vmax, 0) or
//   cross((V(in_v2) - V(in_v1)) - delta_vmax, 0));
endmodule
```

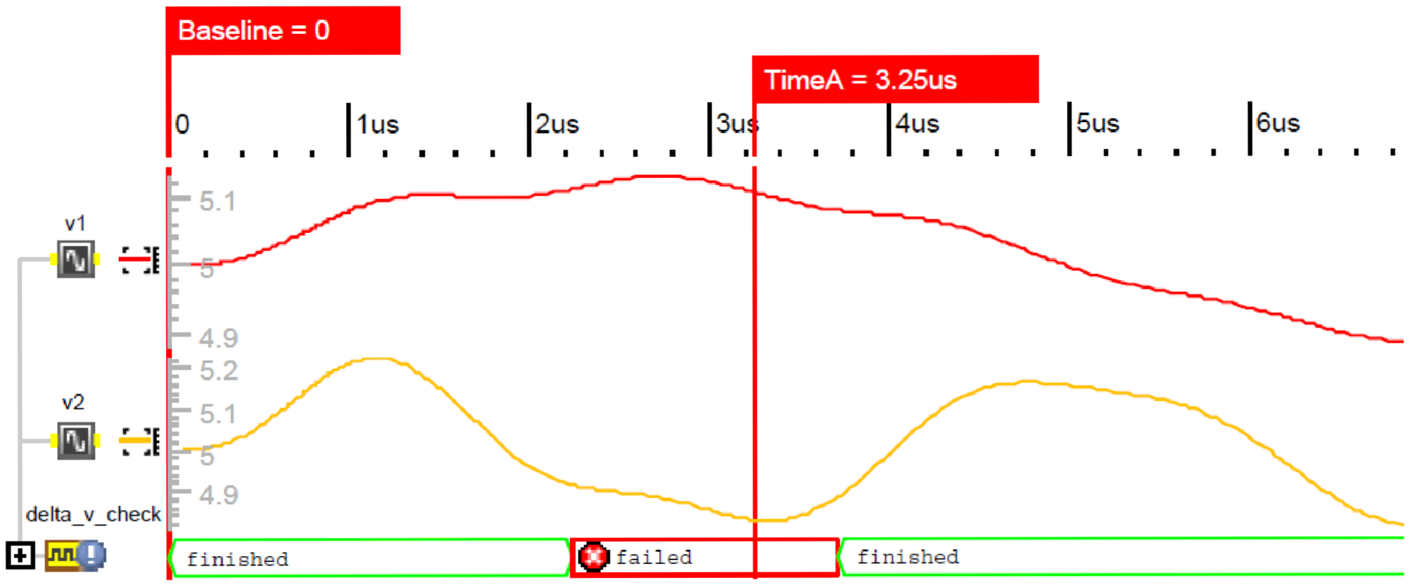


Figure 14: Voltage difference assertion - simulation result in SimVision.

3.3.3 Voltage start-up assertion

Assertion monitors voltage during start-up and checks, that it stabilizes in specified time without undershoots or overshoots.

There are several parameters in the assertion code - *t_to_stabil_us* specifies the maximum acceptable time in which the voltage must stabilize within desired value range, given by parameters *v_stable_max* and *v_stable_min*. Parameter *stablefor_us* defines the time period for which the voltage must stay within range, in order to be considered stable.

When voltage crosses zero in positive direction, value of absolute time is stored into real variable *starttime*. Every time voltage crosses one of the boundary stable values, time is stored in real variable *stoptime*. If voltage stays in range for 5 µs (*stablefor_us*), assertion gets evaluated and both times are subtracted. Assertion fails if the time difference is greater than 10 µs given by parameter *t_to_stabil_us*. If the voltage does not stabilize in time greater than 22.5 µs (*timeout)*, assertion fails.

```
//Verilog-AMS HDL for "cn19Proj", "startup" "verilogams"
`include "constants.vams"
`include "disciplines.vams"
`timescale 1us / 10ns

module startup (in);
input in;
electrical in;

parameter real t_to_stabil_us  = 10;
parameter real stablefor_us    = 5;
parameter real v_stable_max    = 5.05;
parameter real v_stable_min    = 4.95;
parameter real max_overshoot   = 5.5;
parameter real max_undershoot  = 4.5;
real starttime, stoptime, t_stable, timeout;
reg clk, evaluate;
integer count, count2;
/////////////////////////////////////////////////////////
initial
begin
   count = 0;
   count2 = 0;
   evaluate = 0;
   clk = 0;
   t_stable = t_to_stabil_us * 1e-6;
   timeout  = 1.5 * (t_to_stabil_us + stablefor_us);
end

always #1 clk=~clk;

/////////////////////////////////////////////////////////start
```

```
/////////////////////////////////////////////////////////start
always @(cross(V(in) , 1))
      begin
        count2 = 0;
        starttime = $abstime;
      end

always @(cross(V(in) - v_stable_max, 0) or cross(V(in) - v_stable_min, 0))
      begin
        stoptime = $abstime;
       count = 0;
      end
/////////////////////////////////////////////////////////stable_count
always @(clk)
      begin
        count = count + 1;
        if (count == stablefor_us)
        begin
           evaluate = ~evaluate;
        end
      end
/////////////////////////////////////////////////////////time-out_count
always @(clk)
      begin
        count2 = count2 + 1;
        if (count2 >= timeout)
        begin
           evaluate = ~evaluate;
        end
      end
/////////////////////////////////////////////////////////
//psl v_start_tstable_check: assert always
// ((V(in) > 0) ->((stoptime - starttime) <= t_stable)
//  || ((V(in) >= v_stable_max) && (V(in) <= v_stable_min)))
//   @(cross(V(in), 1) or evaluate);
//
//psl max_overshoot_check:   assert always
//      (V(in) <= max_overshoot) @(cross(V(in) - max_overshoot, 1));
//
//psl max_undershoot_check:  assert always
//      (V(in) >= max_undershoot) @(cross(V(in) - max_undershoot, -1));

endmodule
```

In Figure 15, monitored voltage stabilizes in less than 10 µs, and there is no overshoot or undershoot above allowed 4.5 V and 5.5 V. Voltage in Figure 16 stabilizes in time, but both overshoot and undershoot assertions fail.

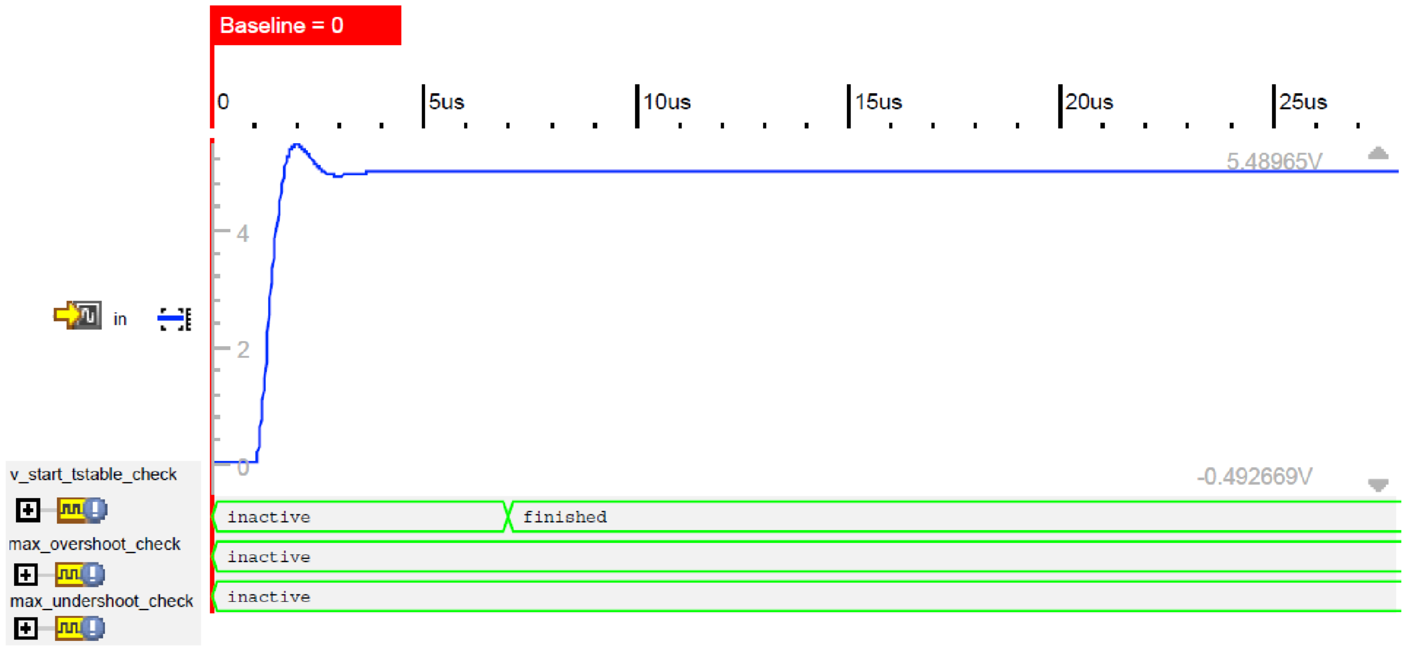


Figure 15: Voltage start-up assertion - simulation result in Simvision.

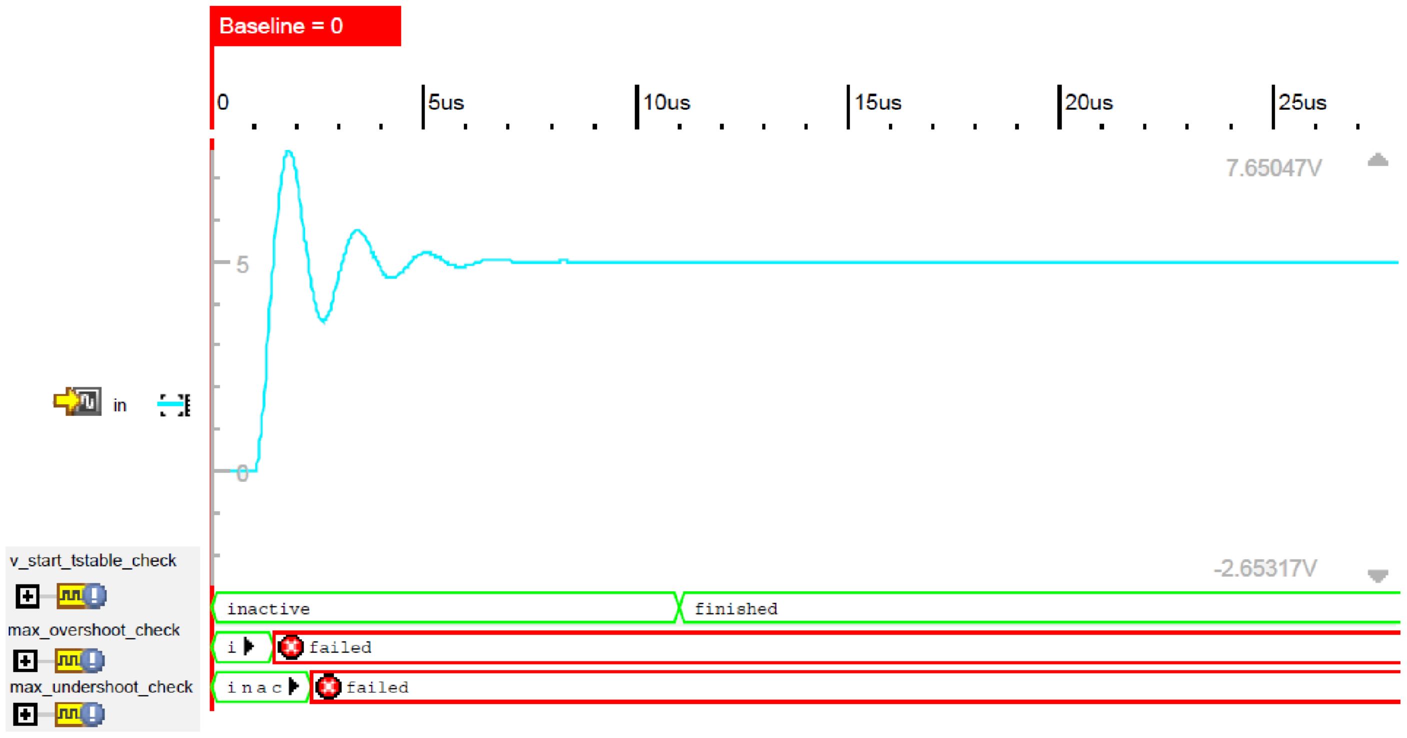


Figure 16: Voltage start-up assertion - second simulation result in Simvision.

Figure 17 shows the case where assertion failed after 22.5 µs due to *timeout*.

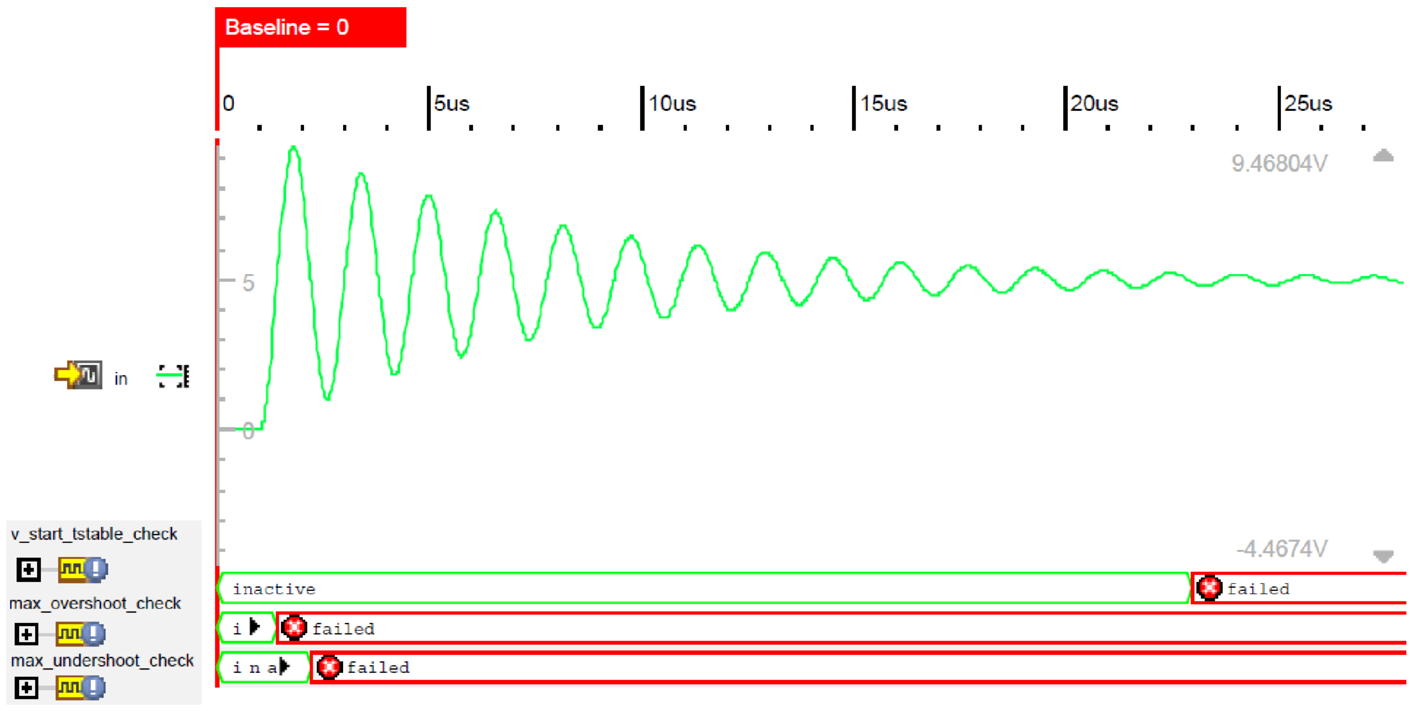


Figure 17: Voltage start-up assertion - third simulation result in Simvision.

3.3.4 Transition time assertion

Assertion checks, that transition between logic 0 to logic 1 and vice versa, occurs in specified time.

There is auxiliary Verilog-AMS code with 2 always blocks for each assertion. During transition from logic 0 to logic 1, values of absolute time are stored when signal crosses 10 and 90 percent of logic high value in positive direction. For 5 volts logic, time is stored when signal crosses 0,5 V and 4,5 V. At 4,5 V - assertion is evaluated and time difference is compared to maximum rise time (tr_rise_max). The same principle is used for transition from logic 1 to logic 0.

```
//Verilog-AMS HDL for "cn19Proj", "transition" "verilogams"
`include "constants.vams"
`include "disciplines.vams"

module transition (in_l);
input in_l;
electrical in_l;

parameter real tr_rise_max = 500n;
parameter real tr_fall_max = 500n;
parameter real        v_90 = 4.5;
parameter real        v_10 = 0.5;
real rise_01;
real rise_09;
real fall_09;
real fall_01;

///////////////////////////////////////////////t_rise_check
always @(cross(V(in_l) - v_10, 1))
        rise_01 = $abstime;

always @(cross(V(in_l) - v_90, 1))
        rise_09 = $abstime;
///////////////////////////////////////////////t_fall_check
always @(cross(V(in_l) - v_90, -1))
        fall_09 = $abstime;

always @(cross(V(in_l) - v_10, -1))
        fall_01 = $abstime;
//////////////////////////////////////////////////////////
//psl t_rise_check: assert always
//      ((rise_09 - rise_01) <= tr_rise_max)
//       @(cross(V(in_l) - v_90, 1));

//psl t_fall_check: assert always
//      ((fall_01 - fall_09) <= tr_fall_max)
//       @(cross(V(in_l) - v_10,-1));
endmodule
```

As can be seen in Figure 18, both assertions failed after the first transition, because parameters *tr_rise_max* and *tr_rise_min* were exceeded. The following three transitions from logic 0 to logic 1 happened in time, the last one caused assertion to fail.

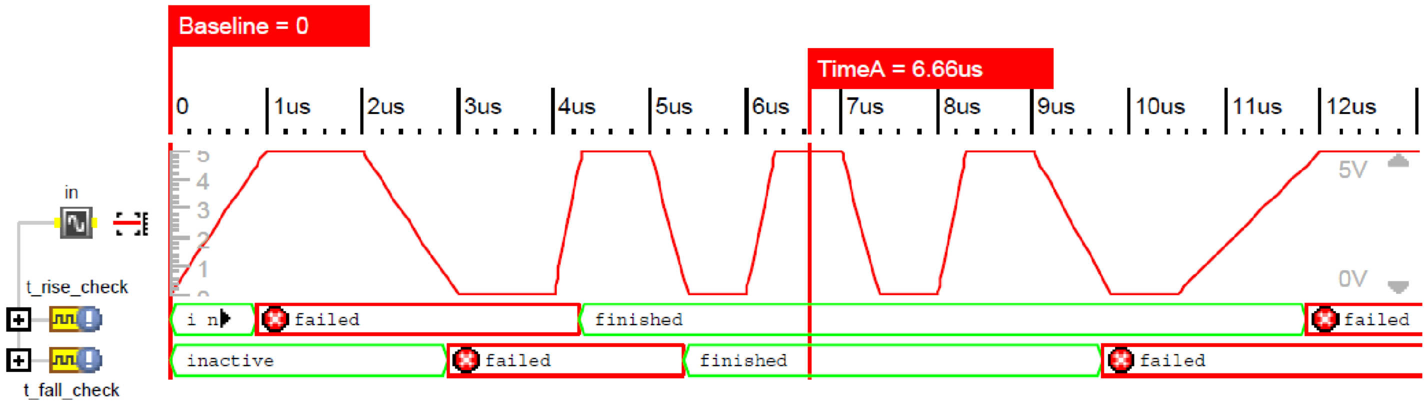


Figure 18: Transition time assertion simulation results in SimVision.

3.3.5 Setup and hold time assertion

Following assertions check, that the data signal is held steady for specified time period before positive edge of the clock (setup time). After the clock event, the data signal should stay steady for another time period (hold time).

Setup time assertion stores time values during signal transitions. Assertion is evaluated, when clock reaches logic 1. Hold time assertion code contains a counter that is used to generate an assertion evaluation event. This event is generated when data signal stays in its logic level for sufficient time after the clock transition. The hold time assertion is evaluated when data signal changes, or when counter evaluation event is generated.

```
//Verilog-AMS HDL for "cn19Proj", "setup_hold" "verilogams"

`include "constants.vams"
`include "disciplines.vams"
`timescale 1ps / 10fs

module setup_hold (in_data, in_clock);
input in_data, in_clock;
electrical in_data, in_clock;

parameter real setup_time_min = 50n;
parameter real  hold_time_min = 50n;
parameter real            v50 = 2.5;
parameter real            v90 = 4.5;

real setup_time1, setup_time2, ht;
integer count, start, finished;
reg clk, eva_ht;

initial
   begin
     ht     = hold_time_min * 1e12;
     clk    = 0;
     eva_ht = 0;
   end

/////////////////////////////////////////////////////////setup time
always @(cross(V(in_data) - v50, 0))
        setup_time1 = $abstime;

always @(cross(V(in_clock) - v50, 1))
         setup_time2 = $abstime;
/////////////////////////////////////////////////////////hold time
always @(cross(V(in_clock) - v50, 1))
      begin
        count  = 0;
        start  = 1;
      end
always @(cross(V(in_clock) - v50, -1))
        finished = 0;
/////////////////////////////////////////////////////////hold time
```

```
//////////////////////////////////////////////////////////hold time
always #1 clk=~clk;
always @(clk)
      begin
        count = count + 1;
        if (count == ht && start == 1)
        begin
           eva_ht   = ~eva_ht;
           start    = 0;
           finished = 1;
        end
      end

//////////////////////////////////////////////////////////
//psl setup_time_check: assert always
//       ((setup_time2 - setup_time1) >= setup_time_min )
//           @(cross(V(in_clock) - v90, 1));
//psl hold_time_check: assert always
//   ((((setup_time1 > setup_time2) && (start == 1)
//    && (finished == 0)) || ((setup_time2 > setup_time1)))
//     ->(finished == 1))
//       @(cross(V(in_data) - v50, 0) or eva_ht);

endmodule
```

During the first transition, both assertions finished successfully. Setup and hold times were greater than 50 ns declared in parameters *setup_time_min* and *hold_time_min*. After the second transition, both assertions failed, because setup and hold times were too short. During the third clock transition, both assertions again finished successfully. The fourth transition was a hold time failure, because data changed to logic 1 before hold time period elapsed.

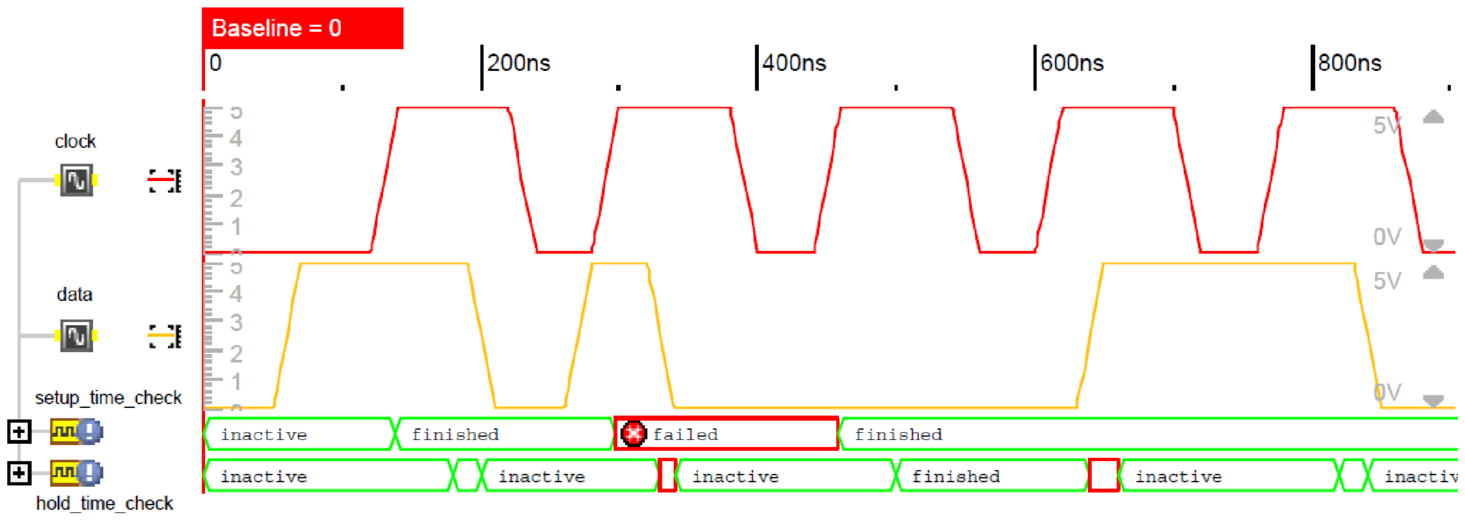


Figure 19: Setup and hold time assertions simulation results in SimVision.

3.4 Case study: Voltage supply block

Assertions mentioned in chapter 3.3 were used for verification of a supply block from power supply control circuit. The supply block creates internal voltage supplies for analog and digital parts of the circuit. There are also voltage and current references with controlling logic.

3.4.1 Monitored events

The main subject of the test was to monitor the four voltage supplies (vdda, vddd, vdd_int, and vdd_osc) and check, that when logic signal enVref is asserted, they never go above maximum specified values (5.3 V, 5.4 V, 5.2 V and 5.2 V) . Also, the voltage difference between them should never be greater than ± 100 mV or ± 200 mV. When signal enVref is deasserted, voltages on vdda and vddd should not exceed 30 mV. On vdda and vddd is also applied the voltage start-up assertion from chapter 3.3.3, which ensures that these voltages stabilize in time. Two voltage (5 V, 0.2 V) and current (1 µA, 5 µA) references are also tested and should not go out of specified range. Assertion code can be seen in Appendix A.

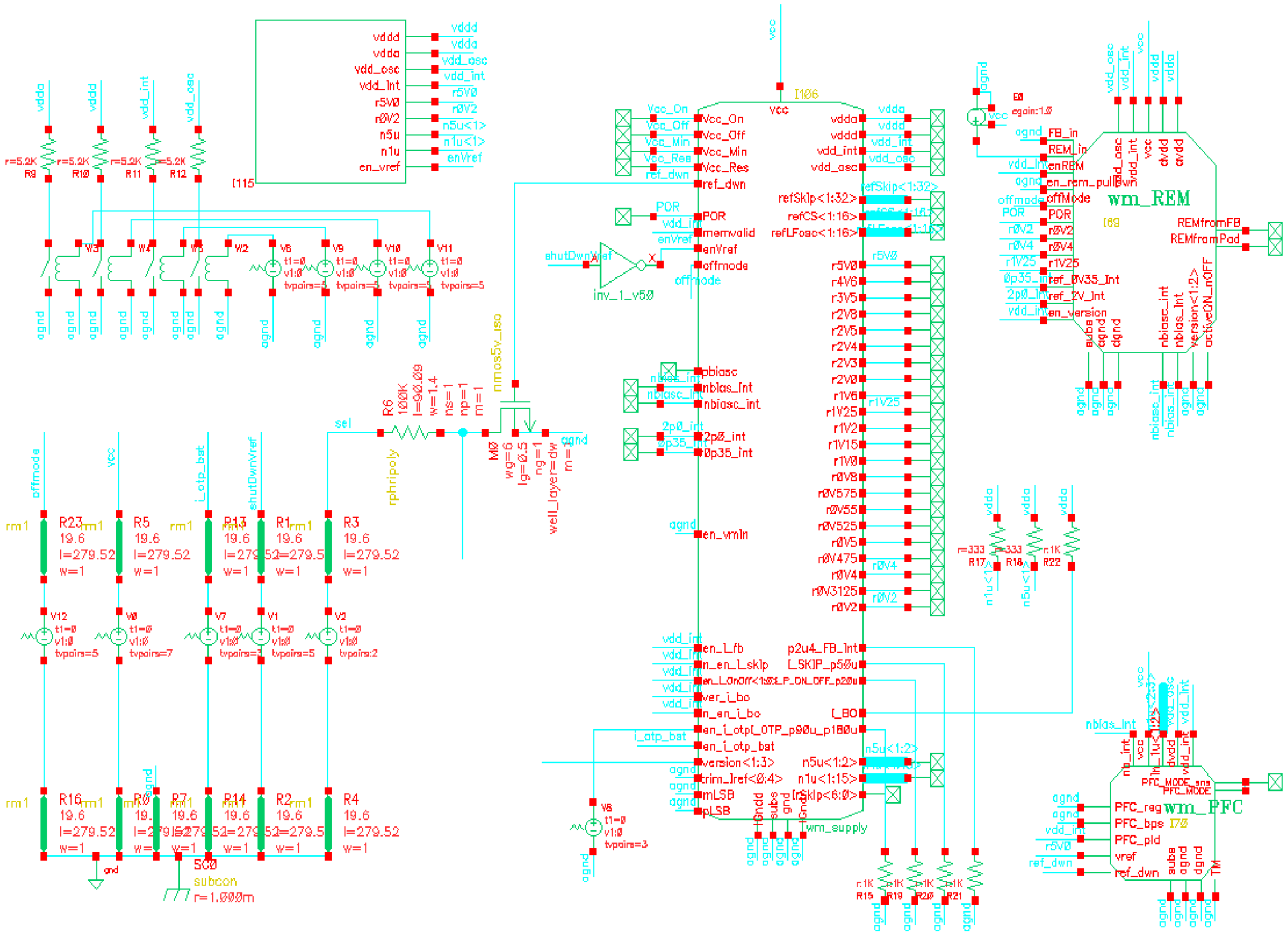


Figure 20: Testbench of voltage supply block

Simulation results

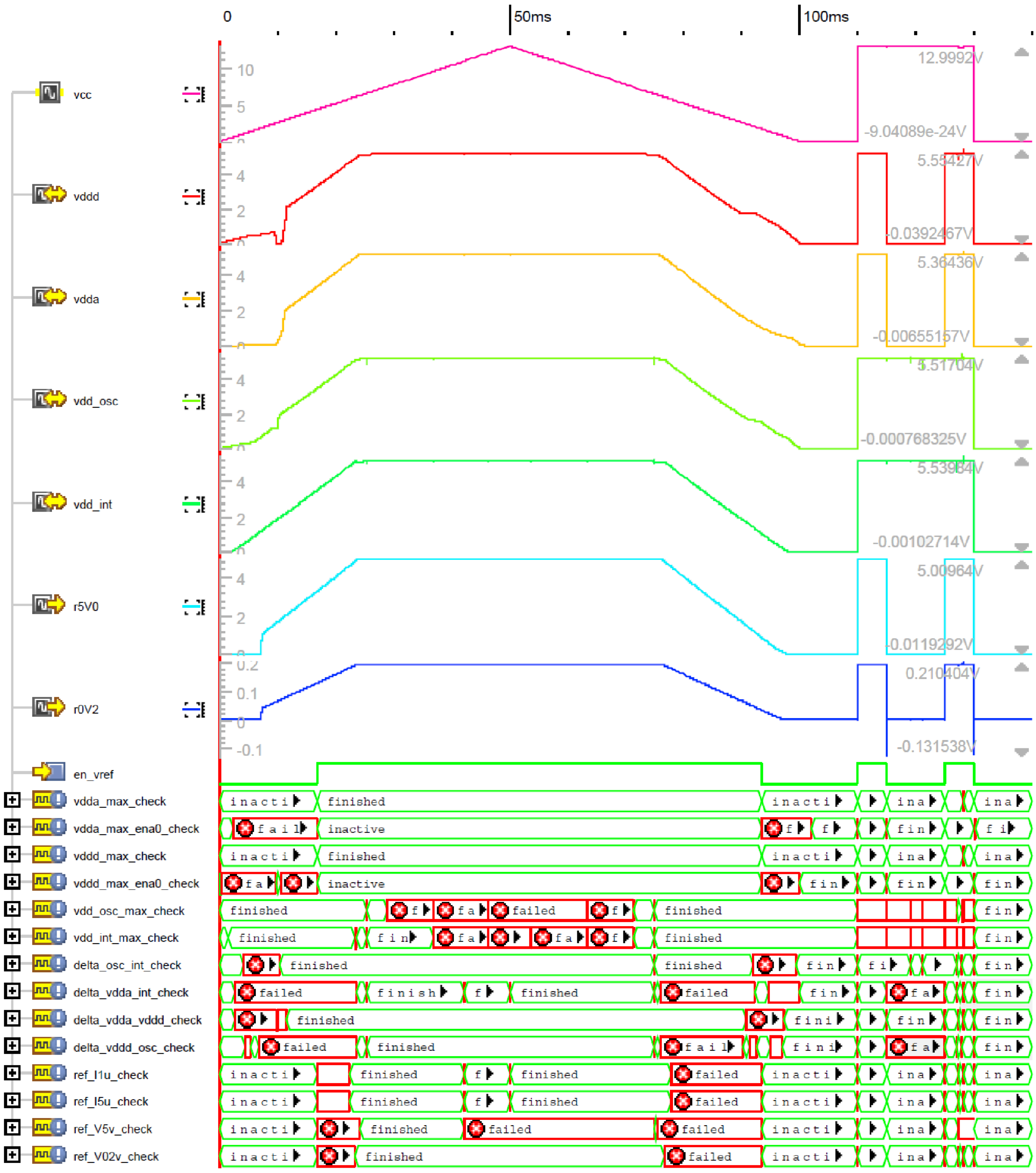


Figure 21: Simulation results of supply block assertion clocked by cross function.

The simulation results can be seen in Figure 21. The voltage supply block should work for vcc ranging between 7 V to 18 V. Supplies vdd_osc and vdd_int are independent from controlling signal en_vref. During the simulation, vcc varies from 0 V to 12 V and once it crosses the threshold of 7 V, the monitored voltages reach their nominal value. Loads are connected for short time periods (1 ms) from time 25 ms to 75 ms, which resulted in several assertion failures. The total failure count can be seen in Table 1.

Figure 22 shows detail of vdda and vddd start-up at time 125 ms, after signal en_vref is asserted. Both voltages stabilized in less than 30 µs without overshoots.

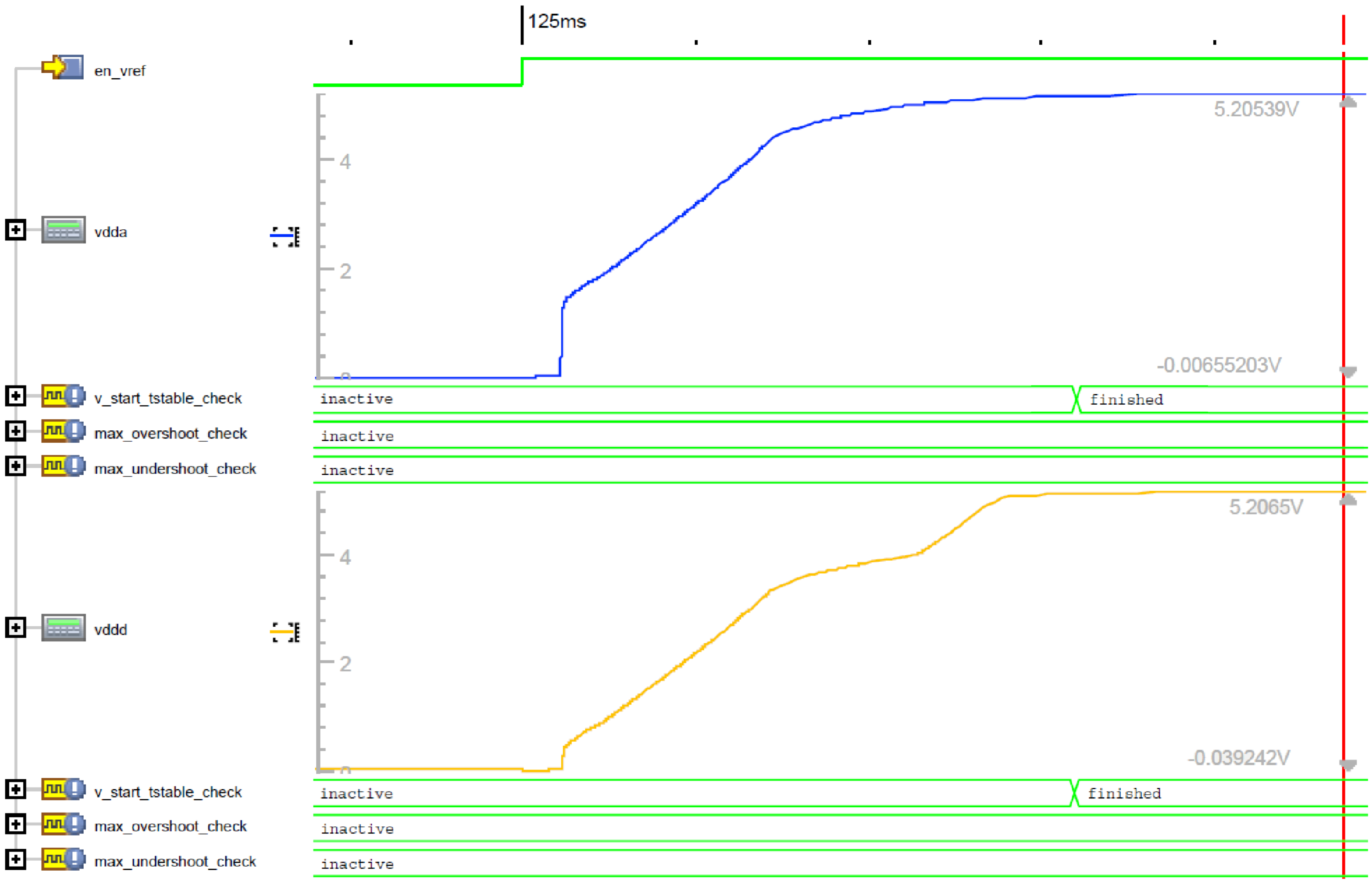


Figure 22: Detail of vdda and vddd at time 125 ms with start-up assertions.

Apart from running the simulation with assertions shown in Appendix A, other clocking methods mentioned in chapter 3.2 were also used. The source codes are shown in Appendices B and C.

The simulation results can be seen in Figure 23, where assertion trios show different results. It is obvious, that assertions evaluated by defined 50 kHz clock cannot detect errors that occur for short time periods. For instance, assertion delta_osc_int, which monitors voltage difference between vdd_osc and vdd_int failed six times since time 100 ms without detection.

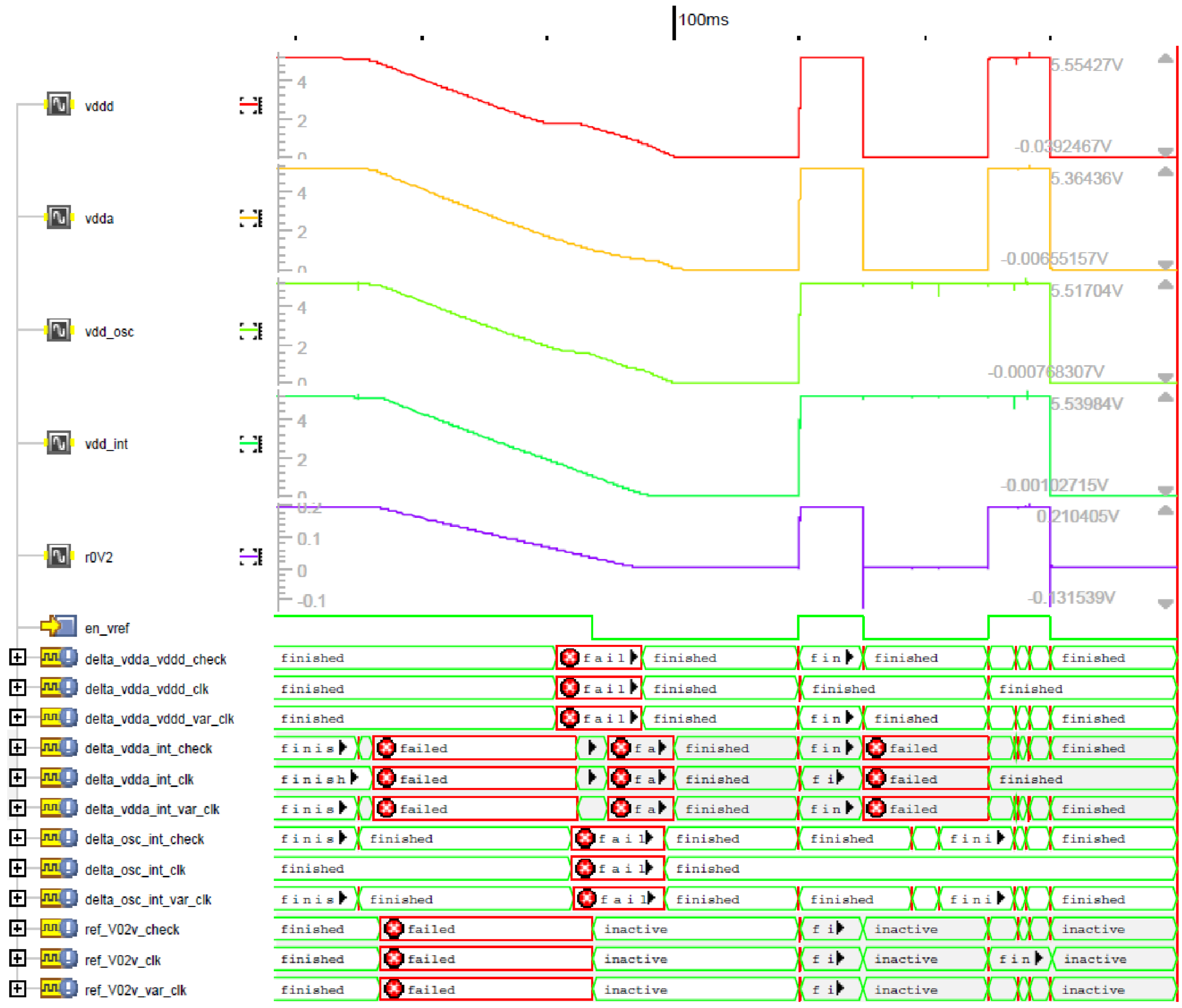


Figure 23: Simulation results for different assertion clocking methods

Table 1: Assertion failures from simulations

assertion name	errors detected				
	cross	variable clock	clock 50 kHz	clock 1 MHz	clock 2 MHz
vdda_max_check	1	1	0	0	0
vdda_max_ena0_check	5	5	3	4	4
vddd_max_check	1	1	0	1	1
vddd_max_ena0_check	6	6	3	4	3
vdd_osc_max_check	24	24	4	9	10
vdd_int_max_check	24	24	5	9	9
delta_osc_int_check	14	14	2	3	5
delta_vdda_int_check	20	20	5	8	7
delta_vdda_vddd_check	12	12	5	8	8
delta_vddd_osc_check	20	20	7	10	14
ref_I1u_check	13	13	4	6	6
ref_I5u_check	18	18	4	6	7
ref_V5v_check	8	8	4	6	5
ref_V02v_check	9	9	4	6	3

Table 1 shows list of assertions and their failure count for all used assertion clocking methods. Assertions evaluated using cross function or variable clock detected all errors while 2 MHz clock detected less than half at almost 40 % simulation time increase. Variable clock brought 14 % increase in time, however it is probable, that this slowdown might be greater while monitoring some rapidly changing signal. The most reasonable method of assertion evaluation seems to be the cross function, which minimally affected the simulation time and detected all errors.

Table 2: Simulation times for different assertion clocking methods.

assertion clocking	simulation time	simulation time increase	errors detected
	(s)	(%)	(%)
no assertions	617	-	0
cross	622	1	100
variable clock	702	14	100
clock 50 kHz	642	4	29
clock 1 MHz	721	17	46
clock 2 MHz	845	37	47

4 CONCLUSION

This master thesis deals with modern verification methods of mixed-signal integrated circuits. The thesis is divided into two parts. The first theoretical part deals with the challenges of mixed-signal verification, explains the difference between analog and digital domains and lists some recently featured verification methods that can be used in mixed-signal verification, such as real number modeling, assertion-based verification and metric-driven verification.

The practical part focuses on assertion-based verification. At first, PSL and SVA descriptive languages are compared on an example of voltage monitor assertion. Verilog-AMS PSL was chosen as a more suitable language for mixed-signal applications, due to its support of analog expressions in arguments. Several methods of assertion clocking were also discussed.

The practical part further contains several examples of created assertions usable in verification of mixed-signal circuits, such as voltage level monitor, voltage difference monitor, voltage start-up monitor, transition monitor or setup and hold time monitor. These assertions are then used in verification of power supply block from power supply control circuit.

The created verification code discovered 175 errors during simulation of the tested block. These errors were mostly caused by voltage peaks during load switching. Additional simulations were run with the purpose of comparing different types of assertion clocking in terms of detected errors and simulation time. The results show, that the analog cross function is the most reasonable way of assertion clocking in analog circuit, for it detected all errors at minimal simulation time increase.

The assertion based verification method proved to be useful tool for debugging analog or mixed-signal circuits, because it leads to quicker identification of bugs that might otherwise be overlooked during manual waveform inspection. The properties captured as assertions have varied from the very simple to the reasonably complex and some of the violations would not be immediately apparent from waveform inspection. Inspection of assertion status makes it easier to identify and debug issues in the correct context.

5 REFERENCES

[1] NIZIC, Mladen. Mixed-Signal Design Trends and Challenges. *Mixed-Signal Methodology Guide: Advanced Methodology for AMP IP and SOC Design, Verification and Implementation*. San Jose: Cadence Design Systems, Inc., 2012, s. 1-10. ISBN 978-1-300-03520-6.

[2] KARNANE, Kishore, BALASUBRAMANIAN, Sathishkumar. Solutions for Mixed-Signal SoC Verification: New techniques that are making advanced SoC verification possible. *Cadence Design Systems* [online]. 2012 [cit. 2013-12-07].

Dostupné z: http://www.cadence.com/rl/Resources/white_papers/ms_soc_verification_wp.pdf

[3] Mixed-signal SOC verification using analog behavioral models. WANG, Qi. CADENCE DESIGN SYSTEMS. *EDN Network* [online]. 2012 [cit. 2013-12-08]. Dostupné z: http://www.edn.com/design/integrated-circuit-design/4392246/1/Mixed-signal-SOC-verification-using-analog-behavioral-models-

[4] VOGELSONG, Ronald. AMS Behavioral Modeling. *Mixed-Signal Methodology Guide: Advanced Methodology for AMP IP and SOC Design, Verification and Implementation*. San Jose: Cadence Design Systems, Inc., 2012, s. 25-70. ISBN 978-1-300-03520-6.

[5] BHATTACHARYA, Prabal, O'RIORDAN, Don. Mixed-Signal Verification Methodology. *Mixed-Signal Methodology Guide: Advanced Methodology for AMP IP and SOC Design, Verification and Implementation*. San Jose: Cadence Design Systems, Inc., 2012, s. 1-10. ISBN 978-1-300-03520-6.

[6] GOERING, Richard. DAC Panel: Users Describe Mixed-Signal Verification Challenges, Solutions. *Cadence Design Systems* [online]. 2011 [cit. 2013-12-08]. Dostupné z: http://www.cadence.com/community/blogs/ii/archive/2011/06/13/dac-panel-users-describe-mixed-signal-verification-challenges-solutions.aspx

[7] Walter Hartong, Scott Cranston. *Real Valued Modeling for Mixed Signal Simulation*, Cadence Design Systems , 2009 [cit. 2013-12-08]. Dostupné z:

http://www.cadence.com/rl/Resources/application_notes/real_number_appNote.pdf

[8] MEHTA, Ashok. *Systemverilog assertions and functional coverage: Guide to Language, Methodology and Applications*. New York: Springer, 2013, p. cm. ISBN 978-146-1473-237.

[9] GOERING, Richard. DVCon Paper: Assertion-Based Verification For Mixed-Signal Designs. *Cadence Design Systems* [online]. 2011 [cit. 2013-12-08]. Dostupné z: http://www.cadence.com/Community/blogs/ii/archive/2011/03/10/dvcon-assertion-based-verification-serves-mixed-signal-designs.aspx

[10] BHATTACHARYA, Prabal, O'RIORDAN, Don. Assertion-based verification in mixed-signal design. *EE Times* [online]. 2011 [cit. 2013-12-08]. Dostupné z: http://www.eetimes.com/document.asp?doc_id=1279150

[11] Property Specification Language Reference Manual. [cit. 2013-13-08]. Dostupné z: http://www.eda.org/vfv/docs/PSL-v1.1.pdf

[12] Mixed Signal Assertion Based Verification. *Cadence Design Systems* [online]. 2010 [cit. 2013-12-08]. Dostupné z: http://www.cadence.com/rl/Resources/conference_papers/ATC-122%20Slides.pdf

[13] Verilog Analog/Mixed-Signal (AMS) Working Group. *Accellera Systems Initiative* [online]. [cit. 2013-12-08].

Dostupné z: http://www.accellera.org/activities/committees/verilog-ams/

[14] 1800-2012 - IEEE Standard for SystemVerilog-Unified Hardware Design, Specification, and Verification Language. IEEE Xplore [online]. [cit. 2013-12-08].

Dostupné z: http://standards.ieee.org/findstds/standard/1800-2012.html

[15] MILNE, Andy, ROBERTS, Damian. Utilizing Digital Techniques for Analog and Mixed-Signal Verification. *Synopsis* [online]. 2011 [cit. 2013-12-08]. Dostupné z: http://www.synopsys.com/tools/verification/amsverification/capsulemodule/customsim-utidigitaltech-wp.pdf

[16] KHAN, Neyaz, KASHAI Yaron, FANG Hao. Metric-driven verification of mixed-signal designs. [cit. 2013-12-08].

Dostupné z: http://events.dvcon.org/2011/proceedings/papers/03_4.pdf

[16] HAVLICEK, John, WOLFSTHAL Yaron, FANG Hao. PSL and SVA: Two standard assertion languages addressing completementary engineering needs. [cit. 2013-12-08].

http://www.inf.pucrs.br/~calazans/graduate/SDAC/PSL%20and%20SVA_two%20standard%20assertion%20languages%20addressing%20complementary%20engineering%20needs.pdf

[17] Cadence Verilog-AMS Language Reference Version 13.1. Cadence Design Systems, Inc., June 2013.

[18] SANTONJA, Regis. Re-usable continuous-time analog SVA assertions.[cit. 2014-15-04].

Dostupné z: http://www.slideshare.net/regis_santonja/re-usable-continuoustime-analog-sva-assertions-slides

LIST OF ABBREVIATIONS

SoC – System on Chip

IP – Intellectual Property

IC – Integrated Circuit

SPICE – Simulation Program with Integrated Circuit Emphasis

RTL – Register Transfer Level

ABV – Assertion-Based Verification

MDV – Metric-Driven Verification

UVM – Universal Verification Methodology

IPC – Inter-Process Communication

RF – Radio Frequency

PSL – Property Specification Language

SVA – System Verilog Assertions

SV-AMS –System Verilog-Analog Mixed-Signal

LIST OF APPENDICES

Appendix A: Voltage supply block assertions clocked by cross function.

```
//Verilog-AMS HDL for "cn19Proj", "wm_supply_assert_cross" "verilogams"

`include "constants.vams"
`include "disciplines.vams"

module wm_supply_assert_cross ( vdda, vddd, vdd_osc, vdd_int, en_vref,
r5V0, r0V2, n5u, n1u );
input      vdda, vddd, vdd_osc, vdd_int, en_vref, r5V0, r0V2, n5u, n1u;
electrical vdda, vddd, vdd_osc, vdd_int, r5V0, r0V2, n5u, n1u;
logic      en_vref;

///////////////////////////////////////////////////
parameter real vdda_max            = 5.3;
parameter real vddd_max            = 5.4;
parameter real vdd_osc_max         = 5.2;
parameter real vdd_int_max         = 5.2;
parameter real vdd_max_ena0        = 30m;
parameter real delta_vdda_vddd     = 200m;
parameter real delta_vdd_osc_int   = 200m;
parameter real delta_vdd_int_vdda  = 100m;
parameter real delta_vdd_osc_vddd  = 100m;
parameter real vref_5_min          = 4.995;
parameter real vref_5_max          = 5.005;
parameter real vref_02_min         = 0.195;
parameter real vref_02_max         = 0.205;
parameter real iref_5u_min         = 4.75u;
parameter real iref_5u_max         = 5.25u;
parameter real iref_1u_min         = 0.950u;
parameter real iref_1u_max         = 1.050u;
parameter real tresh               = 5m;
parameter real treshI              = 15n;

///////////////////////////////////////////////////
//psl vdda_max_check: assert always
//   ((en_vref == 1) -> (V(vdda) < vdda_max))
//@(cross(V(vdda) - tresh, 0) or cross(V(vdda) - vdda_max, 0)
// or en_vref);
//
//psl vddd_max_check: assert always
//   ((en_vref == 1) -> (V(vddd) < vddd_max))
//@(cross(V(vddd) - tresh, 0) or cross(V(vddd) - vddd_max, 0)
// or en_vref);
//
//psl vdda_max_ena0_check: assert always
//   ((en_vref == 0) -> (V(vdda) < vdd_max_ena0))
//@(cross(V(vdda) - vdd_max_ena0, 0) or en_vref);
//
//psl vddd_max_ena0_check: assert always
//   ((en_vref == 0) -> (V(vddd) < vdd_max_ena0))
//@(cross(V(vddd) - vdd_max_ena0, 0) or en_vref);
```

```
//psl vdd_osc_max_check: assert never
//   (V(vdd_osc) > vdd_osc_max)
//@(cross(V(vdd_osc) - tresh, 0) or cross(V(vdd_osc) - vdd_osc_max, 0));
//
//psl vdd_int_max_check: assert never
//   (V(vdd_int) > vdd_int_max)
//@(cross(V(vdd_int) - tresh, 0) or cross(V(vdd_int) - vdd_int_max, 0));
//
//psl delta_vdda_vddd_check: assert never
//   (( V(vdda) - V(vddd) > delta_vdda_vddd ) ||
//   ( V(vddd) - V(vdda) > delta_vdda_vddd ))
//@(cross((V(vdda) - V(vddd)) - delta_vdda_vddd, 0)
//or cross((V(vddd) - V(vdda)) - delta_vdda_vddd, 0));
//
//psl delta_osc_int_check: assert never
//   (( V(vdd_osc) - V(vdd_int) > delta_vdd_osc_int ) ||
//   ( V(vdd_int) - V(vdd_osc) > delta_vdd_osc_int ))
//@(cross((V(vdd_osc) - V(vdd_int)) - delta_vdd_osc_int, 0)
//or cross((V(vdd_int) - V(vdd_osc)) - delta_vdd_osc_int, 0));
//
//psl delta_vdda_int_check: assert never
//   (( V(vdda) - V(vdd_int) > delta_vdd_int_vdda ) ||
//   ( V(vdd_int) - V(vdda) > delta_vdd_int_vdda ))
//@(cross((V(vdda) - V(vdd_int)) - delta_vdd_int_vdda, 0)
//or cross((V(vdd_int) - V(vdda)) - delta_vdd_int_vdda, 0));
//
//psl delta_vddd_osc_check: assert never
//   (( V(vddd) - V(vdd_osc) > delta_vdd_osc_vddd ) ||
//   ( V(vdd_osc) - V(vddd) > delta_vdd_osc_vddd ))
//@(cross((V(vddd) - V(vdd_osc)) - delta_vdd_osc_vddd, 0)
//or cross((V(vdd_osc) - V(vddd)) - delta_vdd_osc_vddd, 0));
//
//psl ref_V5v_check: assert always
//   ((en_vref == 1) -> ((V(r5V0) >= vref_5_min) &&
//   (V(r5V0) <= vref_5_max)))
//@(cross(V(r5V0) - tresh, 0) or cross(V(r5V0) - vref_5_min, 0)
//or cross(V(r5V0) - vref_5_max, 0) or en_vref);
//
//psl ref_V02v_check: assert always
//   ((en_vref == 1) -> ((V(r0V2) >= vref_02_min) &&
//   (V(r0V2) <= vref_02_max)))
//@(cross(V(r0V2) - tresh, 0) or cross(V(r0V2) - vref_02_min, 0)
//or cross(V(r0V2) - vref_02_max, 0) or en_vref);
//
//psl ref_I5u_check: assert always
//   ((en_vref == 1) -> ((I(vdda, n5u) >= iref_5u_min) &&
//   (I(vdda, n5u) <= iref_5u_max)))
//@(cross(I(vdda, n5u) - treshI, 0) or cross(I(vdda, n5u) - iref_5u_min, 0)
//or cross(I(vdda, n5u) - iref_5u_max, 0) or en_vref);
//
//psl ref_I1u_check: assert always
//   ((en_vref == 1) -> ((I(vdda, n1u) >= iref_1u_min) &&
//   (I(vdda, n1u) <= iref_1u_max)))
//@(cross(I(vdda, n1u) - treshI, 0) or cross(I(vdda, n1u) - iref_1u_min, 0)
//or cross(I(vdda, n1u) - iref_1u_max, 0) or en_vref);

endmodule
```

Appendix B: Voltage supply block assertions clocked by variable clock.

```
//Verilog-AMS HDL for "cn19Proj", "wm_supply_assert_clock_var" "verilogams"

`include "constants.vams"
`include "disciplines.vams"
`timescale 1ns / 10ps

module wm_supply_assert_clock_var ( vdda, vddd, vdd_osc, vdd_int, en_vref,
r5V0, r0V2, n5u, n1u);
input      vdda, vddd, vdd_osc, vdd_int, en_vref, r5V0, r0V2, n5u, n1u;
electrical vdda, vddd, vdd_osc, vdd_int, r5V0, r0V2, n5u, n1u;
logic      en_vref;

///////////////////////////////////////////////////var.clock
real clk_a;
reg clk = 0;

always @(absdelta(clk_a, 5m, 100p, 1m))
   if ((clk_a == 1.0) || (clk_a == 0.0)) clk = ~clk;
analog begin
   clk_a = 1.0 - clk_a;
end

///////////////////////////////////////////////////
parameter real vdda_max           = 5.3;
parameter real vddd_max           = 5.4;
parameter real vdd_osc_max        = 5.2;
parameter real vdd_int_max        = 5.2;
parameter real vdd_max_ena0       = 30m;
parameter real delta_vdda_vddd    = 200m;
parameter real delta_vdd_osc_int  = 200m;
parameter real delta_vdd_int_vdda  = 100m;
parameter real delta_vdd_osc_vddd  = 100m;
parameter real vref_5_min         = 4.995;
parameter real vref_5_max         = 5.005;
parameter real vref_02_min        = 0.195;
parameter real vref_02_max        = 0.205;
parameter real iref_5u_min        = 4.75u;
parameter real iref_5u_max        = 5.25u;
parameter real iref_1u_min        = 0.950u;
parameter real iref_1u_max        = 1.050u;
parameter real tresh              = 5m;
parameter real treshI             = 15n;

///////////////////////////////////////////////////
//psl default clock = (posedge clk);
//
//psl vdda_max_var_clk: assert always
// ((en_vref == 1) -> (V(vdda) < vdda_max));
//
//psl vddd_max_var_clk: assert always
// ((en_vref == 1) -> (V(vddd) < vddd_max));
```

```
//psl vdda_max_ena0_var_clk: assert always
// ((en_vref == 0) -> (V(vdda) < vdd_max_ena0));
//
//psl vddd_max_ena0_var_clk: assert always
// ((en_vref == 0) -> (V(vddd) < vdd_max_ena0));
//
//psl vdd_osc_max_var_clk: assert never
// (V(vdd_osc) > vdd_osc_max);
//
//psl vdd_int_max_var_clk: assert never
// (V(vdd_int) > vdd_int_max);
//
//psl delta_vdda_vddd_var_clk: assert never
// (( V(vdda) - V(vddd) > delta_vdda_vddd ) ||
// ( V(vddd) - V(vdda) > delta_vdda_vddd ));
//
//psl delta_osc_int_var_clk: assert never
// (( V(vdd_osc) - V(vdd_int) > delta_vdd_osc_int ) ||
// ( V(vdd_int) - V(vdd_osc) > delta_vdd_osc_int ));
//
//psl delta_vdda_int_var_clk: assert never
// (( V(vdda) - V(vdd_int) > delta_vdd_int_vdda ) ||
// ( V(vdd_int) - V(vdda) > delta_vdd_int_vdda ));
//
//psl delta_vddd_osc_var_clk: assert never
// (( V(vddd) - V(vdd_osc) > delta_vdd_osc_vddd ) ||
// ( V(vdd_osc) - V(vddd) > delta_vdd_osc_vddd ));
//
//psl ref_V5v_var_clk: assert always
// ((en_vref == 1) -> ((V(r5V0) >= vref_5_min) &&
// (V(r5V0) <= vref_5_max)));
//
//psl ref_V02v_var_clk: assert always
// ((en_vref == 1) -> ((V(r0V2) >= vref_02_min) &&
// (V(r0V2) <= vref_02_max)));
//
//psl ref_I5u_var_clk: assert always
// ((en_vref == 1) -> ((I(vdda, n5u) >= iref_5u_min) &&
// (I(vdda, n5u) <= iref_5u_max)));
//
//psl ref_I1u_var_clk: assert always
// ((en_vref == 1) -> ((I(vdda, n1u) >= iref_1u_min) &&
// (I(vdda, n1u) <= iref_1u_max)));

endmodule
```

Appendix C: Voltage supply block assertions clocked by defined clock.

```
//Verilog-AMS HDL for "cn19Proj", "wm_supply_assert_clock" "verilogams"

`include "constants.vams"
`include "disciplines.vams"
`timescale 1ns / 10ps

module wm_supply_assert_clock ( vdda, vddd, vdd_osc, vdd_int, en_vref, r5V0,
r0V2, n5u, n1u);
input      vdda, vddd, vdd_osc, vdd_int, en_vref, r5V0, r0V2, n5u, n1u;
electrical vdda, vddd, vdd_osc, vdd_int, r5V0, r0V2, n5u, n1u;
logic      en_vref;

///////////////////////////////////////////////////2Mhz clock

reg clk;
initial clk=0;
always #250 clk=~clk;

///////////////////////////////////////////////////
parameter real vdda_max            = 5.3;
parameter real vddd_max            = 5.4;
parameter real vdd_osc_max         = 5.2;
parameter real vdd_int_max         = 5.2;
parameter real vdd_max_ena0        = 30m;
parameter real delta_vdda_vddd     = 200m;
parameter real delta_vdd_osc_int   = 200m;
parameter real delta_vdd_int_vdda  = 100m;
parameter real delta_vdd_osc_vddd  = 100m;
parameter real vref_5_min          = 4.995;
parameter real vref_5_max          = 5.005;
parameter real vref_02_min         = 0.195;
parameter real vref_02_max         = 0.205;
parameter real iref_5u_min         = 4.75u;
parameter real iref_5u_max         = 5.25u;
parameter real iref_1u_min         = 0.950u;
parameter real iref_1u_max         = 1.050u;
parameter real tresh               = 5m;
parameter real treshI              = 15n;

///////////////////////////////////////////////////
//psl default clock = (posedge clk);
//
//psl vdda_max_clk: assert always
// ((en_vref == 1) -> (V(vdda) < vdda_max));
//
//psl vddd_max_clk: assert always
// ((en_vref == 1) -> (V(vddd) < vddd_max));
```

```
//psl vdda_max_ena0_clk: assert always
// ((en_vref == 0) -> (V(vdda) < vdd_max_ena0));
//
//psl vddd_max_ena0_clk: assert always
// ((en_vref == 0) -> (V(vddd) < vdd_max_ena0));
//
//psl vdd_osc_max_clk: assert never
// (V(vdd_osc) > vdd_osc_max);
//
//psl vdd_int_max_clk: assert never
// (V(vdd_int) > vdd_int_max);
//
//psl delta_vdda_vddd_clk: assert never
// (( V(vdda) - V(vddd) > delta_vdda_vddd ) ||
// ( V(vddd) - V(vdda) > delta_vdda_vddd ));
//
//psl delta_osc_int_clk: assert never
// (( V(vdd_osc) - V(vdd_int) > delta_vdd_osc_int ) ||
// ( V(vdd_int) - V(vdd_osc) > delta_vdd_osc_int ));
//
//psl delta_vdda_int_clk: assert never
// (( V(vdda) - V(vdd_int) > delta_vdd_int_vdda ) ||
// ( V(vdd_int) - V(vdda) > delta_vdd_int_vdda ));
//
//psl delta_vddd_osc_clk: assert never
// (( V(vddd) - V(vdd_osc) > delta_vdd_osc_vddd ) ||
// ( V(vdd_osc) - V(vddd) > delta_vdd_osc_vddd ));
//
//psl ref_V5v_clk: assert always
// ((en_vref == 1) -> ((V(r5V0) >= vref_5_min) &&
// (V(r5V0) <= vref_5_max)));
//
//psl ref_V02v_clk: assert always
// ((en_vref == 1) -> ((V(r0V2) >= vref_02_min) &&
// (V(r0V2) <= vref_02_max)));
//
//psl ref_I5u_clk: assert always
// ((en_vref == 1) -> ((I(vdda, n5u) >= iref_5u_min) &&
// (I(vdda, n5u) <= iref_5u_max)));
//
//psl ref_I1u_clk: assert always
// ((en_vref == 1) -> ((I(vdda, n1u) >= iref_1u_min) &&
// (I(vdda, n1u) <= iref_1u_max)));

endmodule
```