



DEGREE PROGRAMME IN ELECTRICAL ENGINEERING

MASTER'S THESIS

MIXED-SIGNAL VERIFICATION OF ANALOG IP USING SCHEMATIC MODEL GENERATOR AND SYSTEMVERILOG

Author	Juho Tarkiainen
Supervisor	Juha Kostamovaara
Second Examiner	Tarmo Ruotsalainen
Technical Advisor	Lauri Linnansaari

April 2018

Tarkiainen J. (2018) Mixed-Signal Verification of Analog IP Using Schematic Model Generator and SystemVerilog. University of Oulu, Degree Programme in Electrical Engineering. Master's Thesis, 45 p.

ABSTRACT

Verification is one of the most important aspects of designing an integrated circuit. However, the verification by simulating the device level netlist has become problematic as the mixed-signal circuits have become more complex during the years and therefore, the simulation has become very time consuming. This has been alleviated by creating models of the circuits that represent their behaviour. Sometimes the designer may be responsible for the model creation. If the project schedule is tight, there is a danger that the model is either not maintained well enough or even not created. This may compromise the whole verification process.

Cadence Design Systems introduced Schematic Model Generator (SMG) tool to help in creating SystemVerilog Real Number Models (RNM) in a Graphical User Interface (GUI). The tool provides a way for engineers with little or no coding experience to create abstract models of their designs.

In this thesis, the possibility of using SMG to create SystemVerilog-RN (Real-Number) models of analog blocks for mixed-signal verification is investigated. The thesis covers examples of the model creation and verification process for two different analog Intellectual Properties (IP). The aim was to create models which offer enough abstraction of the analog behaviour by utilizing real value numbers to represent electrical variables. Also, the interoperability with SPICE netlist and compatibility with Mentor Graphics Questa environment was examined.

The models were successfully created but some limitations of SMG were noticed, which led to the use of mixing-and-matching handwritten SystemVerilog with the SMG generated code. Nevertheless, SystemVerilog proved to be a decent option for creating mixed-signal models to simulate the effects of both analog and digital worlds in an abstract manner.

Key words: mixed-signal, verification, SystemVerilog, Real Number Models, Schematic Model Generator

Tarkiainen J. (2018) Schematic Model Generatorin ja SystemVerilogin hyödyntäminen analogia-IP-lohkon sekasignaaliverifiointissa Oulun yliopisto, sähkötekniikan tutkinto-ohjelma. Diplomityö, 45 s.

TIIVISTELMÄ

Verifioiminen on tärkeä osa integroitujen sekasignaaliipiirien suunnittelua. Simuloiminen komponenttitason kuvausta käyttämällä on usein ongelmallista piirien monimutkaisuuden ja simulaatioaikojen takia. Tätä ongelmaa on pyritty ratkaisemaan luomalla malleja, jotka kuvaavat piirin toimintaa abstraktilla tasolla. Tällaisen mallin tekeminen voi joskus olla suunnitteluinsinöörin vastuulla. Projektiaikataulun ollessa tiukka, tehty malli voi jäädä päivittämättä tai sitä ei luoda ollenkaan, mikä voi pahimmassa tapauksessa vaarantaa koko piirin verifiointiprosessin.

Cadence Design Systems on kehittänyt Schematic Model Generator (SMG) työkalun, jolla voidaan luoda SystemVerilog RNM –malleja (Real Number Models) käyttäen GUI:ta (Graphical User Interface). Työkalun avulla insinöörit, joilla on vain vähän tai ei yhtään ohjelmointitaitoa, kykenevät itse luomaan abstrakteja malleja piireistään.

Tässä diplomityössä käydään läpi mahdollisuutta käyttää SMG:tä korkean tason SystemVerilog-RN (Real Number) käyttäytymismallien luomiseksi. Diplomityössä käydään läpi mallin tekeminen ja verifiointi kahden erillisen esimerkin avulla. Työn tarkoituksena oli luoda mallit, jotka esittäisivät analogia-IP:eiden (Intellectual Property) toiminnan sopivan abstraktilla tasolla käyttämällä reaalityyppisiä sähköisten suureiden esittämiseksi. Mallien toiminta testattiin myös SPICE-komponenttitason kuvauksen kanssa sekä Mentor Graphicsin Questan simulaatioympäristössä.

Mallit luotiin onnistuneesti, mutta SMG:ssä havaittiin joitain puutteita mallien tekovaiheessa, mikä johti itse tehdyn SystemVerilog-koodin sekoittamiseen SMG:llä luodun koodin sekaan. Siitä huolimatta SystemVerilog osoittautui varteenotettavaksi vaihtoehdoksi sekasignaalmallien luomiseen.

Avainsanat: sekasignaali, verifiointi, SystemVerilog, Real Number Models, Schematic Model Generator

TABLE OF CONTENTS

ABSTRACT

TIIVISTELMÄ

TABLE OF CONTENTS

FOREWORD

LIST OF ABBREVIATIONS AND SYMBOLS

1.	INTRODUCTION	7
1.1.	Mixed-Signal Integrated Circuit	7
1.2.	Top-Down Design Methodology	8
2.	VERIFICATION AND MODELLING	10
2.1.	Verification	10
2.2.	Hardware Description Language Options	12
3.	SYSTEMVERILOG	14
3.1.	History	14
3.2.	Digital Hardware Description Language in Mixed-Signal Modelling	14
3.3.	Motivation	15
4.	SCHEMATIC MODEL GENERATOR	17
4.1.	Basic Concept	17
4.2.	Target Users and Usage Options	20
5.	MODEL CREATION	21
5.1.	Target Blocks	21
5.2.	Model Creation	22
5.2.1.	Low Drop-out Regulator	22
5.2.2.	DCDC	25
5.3.	Model Testing	28
5.3.1.	Test Stimulus	28
5.3.2.	Assertions	31
5.3.3.	Simulation	32
5.4.	Simulation Results and Model Accuracy	33
5.5.	Model Maintenance	34
5.6.	Design Effort in the Model Generation	34
6.	COMPATIBILITY OF DIFFERENT MODELS	36
6.1.	Simulation with Transistor Level Description	36
6.2.	Simulation in Questa Environment	37
7.	DISCUSSION	40
7.1.	SystemVerilog-RN Behavioural Model vs. Transistor Level Design	40
7.2.	Pros and Cons of SMG	40
7.3.	Suggested Workflow	41
7.4.	Future of SMG	42
8.	SUMMARY	43
9.	REFERENCES	44

FOREWORD

This Master's Thesis was written at Nordic Semiconductor Finland ASA with the purpose of studying alternative and possible future options in analog block behavioural modelling. I want to thank the whole company for letting me study such an interesting topic. Especially I want to thank Dr. Tarmo Ruotsalainen, Lauri Linnansaari, Joni Jäntti and Isto Hyyryläinen for close support during the process of writing this thesis. In addition, I would like to express my gratitude to the whole team for helping me out even with the simplest problems I faced during the research process and professor Juha Kostamovaara for supervising this thesis.

Last but not least, thank you my friends and family for supporting me.

Oulu, April 12th, 2018

Juho Tarkiainen

LIST OF ABBREVIATIONS AND SYMBOLS

AC	Alternating Current
ADC	Analog-to-Digital Converter
ADE	Analog Design Environment
AMS	Analog Mixed-Signal
ASIC	Application Specific Integrated Circuit
BBT	Building Block Text
DCDC	Direct Current to Direct Current
DUT	Design Under Test
FIR	Finite Impulse Response
E2L	Electrical-to-Logic
E2R	Electrical-to-Real
EDA	Electronic Design Automation
HDL	Hardware Description Language
IC	Integrated Circuit
IoT	Internet of Things
IP	Intellectual Property
LDO	Low Drop-out Regulator
MIMO	Multiple-Input and Multiple-Output
PLL	Phase-Locked Loop
PWM	Pulse Width Modulation
RNM	Real Number Models
SMG	Schematic Model Generator
SoC	System-on-Chip
UDT	User-Defined Types
ULP	Ultra Low Power
VCO	Voltage Controlled Oscillator
VHDL	Very High Speed Integrated Circuit HDL

1. INTRODUCTION

System-on-Chips (SoC) are widely used in today’s electronic systems. Their design has become increasingly complicated which has resulted in the difficulty of identifying the design bugs before manufacturing. The design verification may take even 80 percent or more of the overall time, thus the development of verification techniques in different levels of abstraction to detect bugs is indispensable. [1]

On the market side, telecommunication systems, automotives, Internet of Things (IoT) and other high speed data applications have increased the demand on mixed-signal integrated circuits (IC). The design cycle time for such SoCs may be even around 18 to 24 months. Delays due to bugs in the system may delay the project by another 6 to 12 months. This pushes towards well defined mixed-signal design environments to meet the time-to-market and other different challenges of the SoC designs. [2]

1.1. Mixed-Signal Integrated Circuit

Mixed-signal IC is a device which operates in both digital and analog domains, presenting either the digital or analog information in either form. To clarify, one can describe a mixed-signal device using the matrix shown in Figure 1, in which i.e. a device that performs an analog operation with a digital function or processes digital data with an analog function, is a mixed-signal device. [3] For example, B could be an analog-to-digital converter (ADC) and C could be a finite impulse response (FIR) filter.

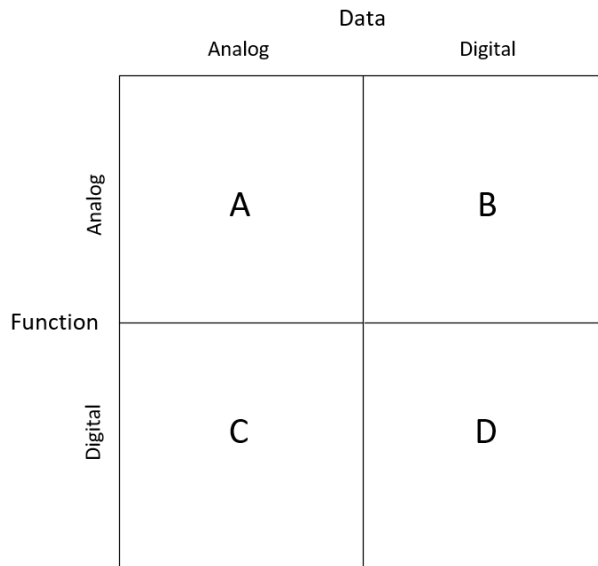


Figure 1. Matrix illustrating the boundaries of analog and digital domains. A mixed-signal system contains and mixes both analog and digital signals and functions (B and C).

The mixed-signal circuit design flow can be described by being either “Analog on Top” or “Digital on Top”. If the design is analog-centric and contains only small or

medium amounts of digital logic, it can be said that the design process is schematic driven. The top level is created using a schematic (e.g. in Cadence Design Systems Virtuoso environment), making it analog. The digital content is designed in a separate environment using the digital design flow and then synthesized and imported. If the majority of the mixed-signal design consists of digital logic, the top level is not anymore schematic driven but netlist driven. Digital content and the digital top level are netlisted and the separate analog IP content is imported or modelled. [4]

1.2. Top-Down Design Methodology

Top-down design is a method in which a behavioural model of the system is first written and simulated. Behavioural model is a high level description of the system created using Hardware Description Language (HDL). The behavioural model is developed and tuned until it meets the requirements of the actual design, making it an executable specification and giving the possibility to compare the actual design against the model later on. Traditionally, the behavioural models are created manually but as this is a time consuming task, there is a need to increase the level of automation in the process. [5] [6]

In order to increase the innovation in analog and mixed-signal designs, new concepts of architecture are required, which need to be proven first at an early phase of the design process. For complex analog systems, behavioural models need to be utilized to reliably simulate and verify the functionality of the complete system before the transistor level implementation. [7]

The behavioural models of individual IPs located inside the system are used to simulate different things depending on the focus of the model. During the architectural exploration, the behavioural models enable optimizing the interaction between different IPs, e.g. control signals and handshakes. It is also possible to implement electrical variables (current, voltage) or their real value presentations in the model to e.g. model the current consumption during different operation modes. The focus of the model determines, which HDL-language should be used for modelling as different languages enable the user to model the IP at different levels of abstraction.

Using top-down approach to design a mixed-signal circuit gives great benefit when validating the specification of the IC. When designing a complex system, a common way is to split the whole system into a number of separate modules and assign the modules to group of designers. These separate models and their specification parameters need to be closely studied as this enables the designers to identify the fundamental parameters of the design. This helps the project to stay on the given budget and schedule as the problems are identified early on. [8]

The top-down method is aimed towards designs that have strict performance requirements due to e.g. commercial markets that are performance driven and have very high volumes. The opposite approach to top-down approach is so called bottom-up, in which the design is cost driven. This means that custom components are only made when it produces more cost effective approach than using off-the-shelf components. The whole system is then built using the chosen parts and the overall performance of the system is known only at the end of the design process. [5] Today,

it is more common that the systems are built top-down due to the strict requirements and manufacturing volumes.

V-curve describing the top down design methodology can be seen in Figure 2. After the starting system requirements are clear, the system must be created using models due to lack of actual design blocks. When the top level system functionality is clear, the individual sub-systems, e.g. a Phase-locked Loop (PLL), located inside the system must be examined and their requirements need to be set to meet the top level requirements at the end. In order to meet the sub-system requirements, the sub-system components, e.g. a Voltage Controlled Oscillator (VCO) in PLL must be defined. The actual design process can start after the requirements of every level (system, sub-system and component) are clear. The design process starts from the component level and continues up to the system level. One could say that the requirements are set top-down and the actual design is done bottom-up. After each level is built, it must be checked against its set requirements before continuing to higher level. Finally, the built top level can be compared against the created top level model to check if the system meets the set requirements.

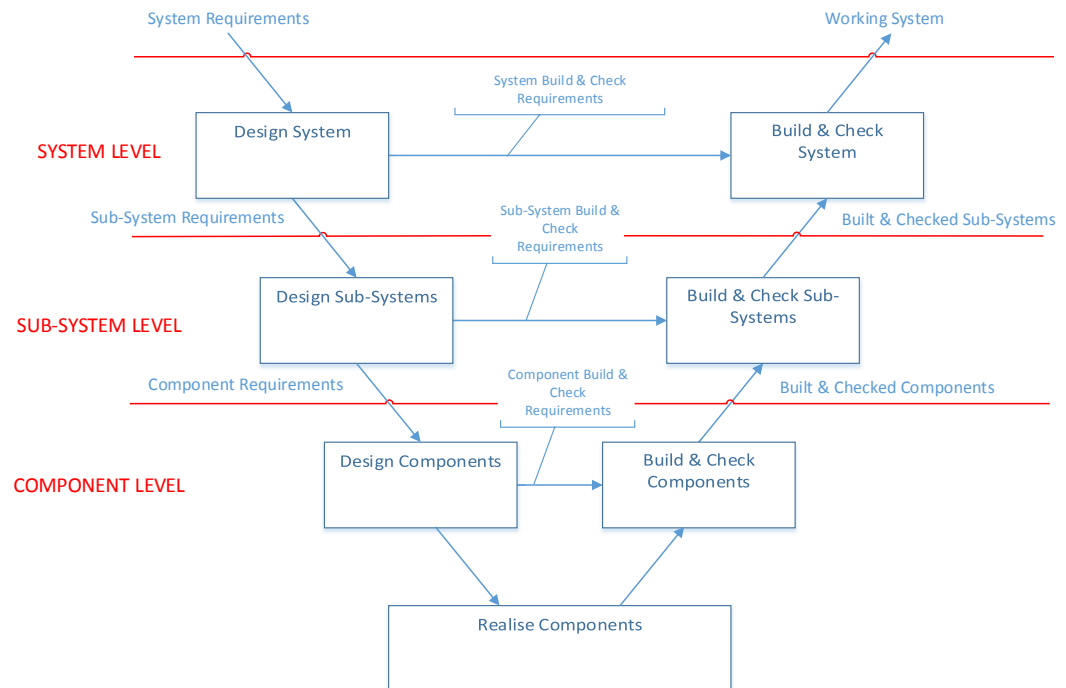


Figure 2. V-curve describing top-down design methodology.

2. VERIFICATION AND MODELLING

Verification of electronics has become increasingly important as the complexity of the systems has increased. There exists many different types of verification, including manufacturing verification, functional verification and timing verification. This thesis focuses on functional verification.

The goal of functional verification is to ensure that the design is working as intended. Before, functional verification was done when a prototype of the design had been manufactured. In case of an error, the prototype would be modified to fix the error. For this to work, the internal operation of the prototype had to be observable. In addition, it had to be possible to fix the found problems on the same prototype afterwards. Clearly, this is not always possible for today's systems as the discrete components are not that widely used anymore and for Application Specific Integrated Circuits (ASIC) and SoCs, the ability to observe the internal operation is limited. Also, the cost of making a modification may be measured in millions of dollars and may have impact on the project schedule. This leads to need of today's functional verification which aims to verification of the design before the prototype is even build. [9]

2.1. Verification

Functional verification of the design using simulation attempts to create a virtual prototype by collecting as much relevant information of the components in the design as possible and how they are connected in the whole system. This set of information is then transformed into a correct format, making it a model of the system. The model should show how the actual design should behave and help to note the discrepancies between the model and the design. When problems are found in the design, the model can be used to help locate the source of the problem and fix it. This iterative process is run until no more or only few problems are found and the prototype can be built. Using models in the verification process enables faster debugging as all nodes are observable and the real life limitations, such as loading effects are not changing the behaviour of the observed node. This makes it possible to do faster, and therefore cheaper iterations. Also, one of the biggest benefits of using models is that they allow the software development to begin much earlier, which often shows areas where the design could be improved. [5]

Behavioural models offer the highest level of abstraction. The high level of abstraction means that the purpose of the model is to only simulate what happens on the boundary of the sub-model e.g. input and output. Behavioural model of an IP is not supposed to provide detail of how the actual work is done. The benefit of sacrificing the model detail accuracy is the speed of creation, modification and simulation which enables them to be used in architectural exploration, performance modelling or hardware/software code sign. [5] Also, one big advantage of behavioural models is the increased possibility to use the created IP model later in other systems. For generic IPs, such as Low Drop-Out (LDO) regulators and Pulse Width Modulators (PWM), it is likely that the IP model can be reused with only little additional effort. This is possible since behavioural models are not technology dependent. The correspondence between the analog design abstraction and technology dependency is illustrated in Figure 3. [10] The abstraction of the design

description is high at system level, where the system is built using “black boxes”. Each “black box” fulfils a certain action required by the system but doesn’t consider how the action is executed. At low abstraction level, single system modules are designed at physical level using a certain technology, which leads to the design being tightly bind to the selected technology.

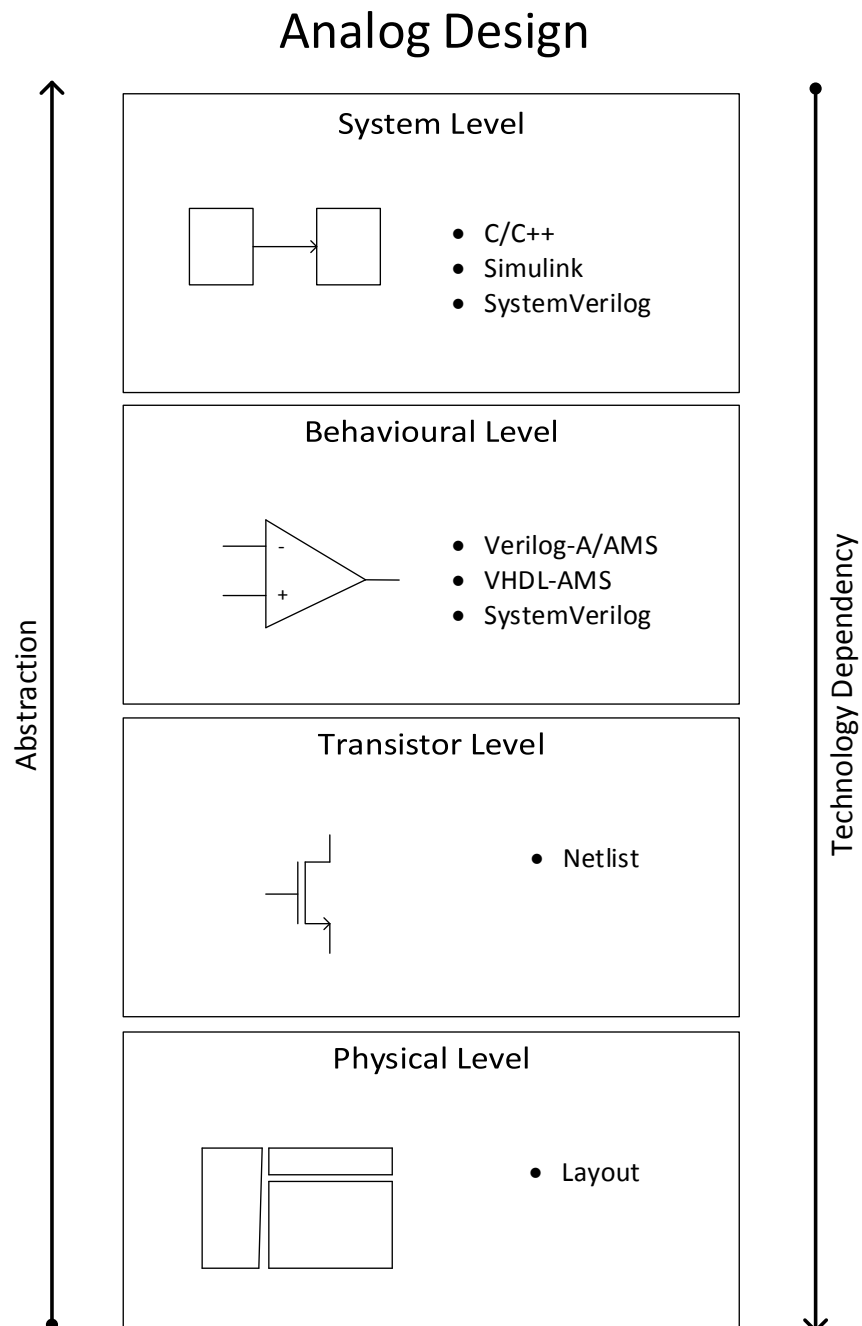


Figure 3. Technology dependency versus the level of abstraction in analog design.

2.2. Hardware Description Language Options

HDLs are programming languages that are used for modelling the behaviour of devices and processes and they can be divided into three separate parts: digital, analog and mixed-signal. Digital HDLs are based on event-driven techniques and a discrete model of time. Analog HDLs use differential and/or algebraic equations that provide solutions varying continuously with time. Mixed-signal HDLs support both. [11] This thesis focuses on HDLs that can be utilized to create analog and mixed-signal behavioural models.

Depending on the desired level of model abstraction and the model usage, there exists different HDL options (e.g. SystemVerilog, Verilog-AMS, Wreal, VHDL-AMS) to model an analog IP for mixed-signal purposes. Currently, SystemVerilog is widely used digital HDL. The popularity to use SystemVerilog also for analog design purposes is increasing due to real number value and User Defined Types (UDT) support.

SystemVerilog is inherited from Verilog and can be used for both hardware description and verification. For analog behavioural modelling purposes, SystemVerilog can be utilized to represent electrical variables, such as voltage and current (i.e. output of a system) using real numbers. In case only real value representation is insufficient, one can take advantage of UDTs. UDTs allow the user to describe analog, Kirchoff's laws obeying behaviour on a single net in discrete manner. The single net can have multiple drivers and it can combine voltage, current and resistive effect of all the drivers. For verification purposes, SystemVerilog offers the user possibility to take advantage of assertions and metric-coverage. Assertions are user created warnings that are triggered when the system works incorrectly and metric-coverage is a measure that determines how much of the design specification has been exercised in the verification. Assertions and metric-coverage are both already widely used in digital design verification.

Verilog-AMS is a HDL used for behavioural modelling of analog, discrete and mixed-signal systems. Verilog-A is the subset of Verilog-AMS which describes the analog behaviour obeying the Kirchhoff's Potential and Flow Laws (KPL and KFL). [12] Verilog-AMS combines Verilog and Verilog-A with some additional features. Verilog and Verilog-A are related but still different languages which run on different simulators. Verilog-A runs on circuit simulators, Verilog on logic simulators and Verilog-AMS on mixed-signal simulators. [13] The benefit of using Verilog-A is that the model can be simulated using circuit simulator instead of mixed-signal simulator, however the model can then describe only analog behaviour. For mixed-signal purposes, one still needs to use mixed-signal simulator.

Wreal is an extension that was donated by Cadence Design Systems to the Accelera Systems Initiative and it is a datatype that was created for Verilog-AMS. Some EDA vendors have utilized the datatype to support SystemVerilog real number modelling capabilities for analog mixed-signal modelling purposes. Wreal aims to provide accurate analog mixed-signal simulation by describing the analog behaviour using real-number values in discrete time. A wreal net supports multiple drivers just like wire-net type but the difference is that the drivers of wreal nets are real numbers rather than logic levels. The downside of Wreal compared to SystemVerilog UDT is that even though Wreal is able to resolve a multiple driver net value, the result of Wreal is only 1-dimensional (voltage or current), while SystemVerilog UDT

extensions enable simulating current, voltage and impedance on a single net concurrently.

VHDL-AMS is an extension to the existing Very High Speed Integrated Circuit Hardware Description Language (VHDL). Using VHDL-AMS, one can create behavioural models at varying level of abstraction. The user can create models for individual parts, i.e. inductors and capacitors at the desired level of detail. This may be of interest in case the model requires more accurate simulation of electrical effects, such as closed-loop feedback. For certain simulations, i.e. lower level architecture exploration, the more accurate electrical behaviour simulation could show problems in certain control schemes. However, simulating these effects would mean that the model is at lower level than a behavioural model. One should always consider which effects to include in the model and which ones should be excluded to guarantee that the model can be used for its purpose. [14]

3. SYSTEMVERILOG

SystemVerilog, officially IEEE Std 1800-2005 standard, is a widely used hardware modelling description and verification language. It is a set of extensions to IEEE Std 1363-2005 Verilog Standard, also known as Verilog-2005. The extensions can be generalized in two primary categories [15]:

- Enhancements to hardware modelling in both efficiency and abstraction level
- Enhancements to verification and assertions in order to create efficient and race-free testbenches

3.1. History

Major part of SystemVerilog was released as an Accelera standard in June 2002 under the name of SystemVerilog 3.0. Accelera is a non-profit organization which supports the development and use of EDA (Electronic Design Automation) languages. SystemVerilog was the third generation of Verilog language family that started with Verilog-1995 defined by Phil Moorby in the early 1980s. The SystemVerilog 3.0 release allowed the EDA vendors to begin adding the SystemVerilog extensions to existing simulators, compilers and other engineering tools. This initial release of the SystemVerilog aimed at extending the synthesizable constructs of Verilog and to allow modelling hardware in higher abstraction level. [15]

A major update to SystemVerilog was released in May 2003. Referred to as SystemVerilog 3.1, this release added plentiful verification capabilities to SystemVerilog. Accelera kept on refining the SystemVerilog 3.1 standard by working closely with EDA companies so that the specifications would be implemented as intended. A few additional modelling and verification constructs were defined in May 2004 release of SystemVerilog 3.1a. [15]

Right after SystemVerilog 3.1a was ratified, Accelera donated the SystemVerilog standard to IEEE Standard Association (IEEE-SA), which was the association responsible for Verilog 1364 standard. Accelera worked alongside with the IEEE to make a request for a new standard and to review and standardize the SystemVerilog extensions to Verilog. In November 2005, the official IEEE 1800-2005 standard was released to the public. [15]

The development of SystemVerilog has progressed further after the initial IEEE 1800-2005 release to the public. IEEE 1800-2009 standard (also referred as SV-2009) brought support for real variables and IEEE 1800-2012 (SV-2012) added net types and interconnects, which enable e.g. modelling of complex information on a single net. Both standards added essential features to increase the chances for using SystemVerilog for modelling purposes.

3.2. Digital Hardware Description Language in Mixed-Signal Modelling

The development of mixed-signal HDL, such as Verilog-AMS, has enabled engineers to describe the analog and mixed-signal behaviour of circuits at higher

level of abstraction than by just using conventional circuit simulators. This has enabled far more efficient way of simulating mixed-signal circuits than before [16].

One of the main advantages of adopting digital HDL's (e.g. SystemVerilog) to create mixed-signal behavioural models of analog blocks is the possibility for top-down approach. The possibility to mix low and high level description in different parts of a model increases flexibility, speed and the number of options to use the model for top-down design purposes. For example, in the case of a Phase-Locked Loop (PLL), phase detector architecture can be modelled in a more detailed manner using a physical model, whereas the critical and complex loop filter can be simplified using a mathematical model describing only behaviour. [8]

SystemVerilog provides tools to describe the system in an abstract enough manner to create models for mixed-signal verification purposes. Real-type values can be utilized to e.g. represent the model output voltage of the Design Under Test (DUT). Verification possibilities (assertions, random test input generation) are among the main advantages of using SystemVerilog over other alternative HDLs. Also, cross-compatibility in mixed-signal simulators (Virtuoso AMS, Questa ADMS) and UDTs, which allow defining new data types and modelling the design functionality in fewer lines of code are advantages. [15] The UDTs can be utilized to simulate electrical behavior in discrete time domain by assigning real-type values to their electrical counterparts (voltage, current, impedance). [17]

3.3. Motivation

Increasing complexity in telecom, automotive and consumer electronics has made it difficult for the traditional analog transient analysis to verify the system. In e.g. telecommunications, the number of different operation modes has increased in RF transceivers after LTE-Advanced (4G) introduced carrier aggregation and Multiple-Input and Multiple-Output (MIMO). One can expect in the future that the system complexity and the frequency increase in 5G will also increase the need of creating efficient models for verifying different operation modes and signal paths of the system. This increases the motivation to use HDLs, i.e. SystemVerilog with UDT and digital simulators, which are able to run hundreds of test cases in finite amount of time and are easy for SoC teams to integrate in their verification flow. [18]

The top level of the system usually consists of multiple analog blocks. Even if a single analog block is fast to simulate using a transistor level description, the top level may be too slow to simulate multiple times in a row and correct mistakes after every run. High level behavioural models can be utilized to build the system top level and run the simulation faster than the transistor level description. This enables the hardware and verification teams to quickly locate the incorrect top level connections and wrong control signals. Also, it is possible to divide the system into separate parts, in which one part is simulated using the transistor level description and the simulation time consuming part using HDL models. [19]

IPs with high abstraction models for top system level use increase the chance to reuse them in future. Sub-blocks that are generic or offer a key-feature may be reused in future projects. The advancing technology may change the transistor level design description but the behavioural model is not technology dependent. This increase in flexibility, simulation speed and reusability chance is illustrated in Figure 4. [10] The increase in simulation speed also speeds up the development of testbenches. The

testbench can be created first using the behavioural model and the actual transistor level design can be run in the ready testbench afterwards.

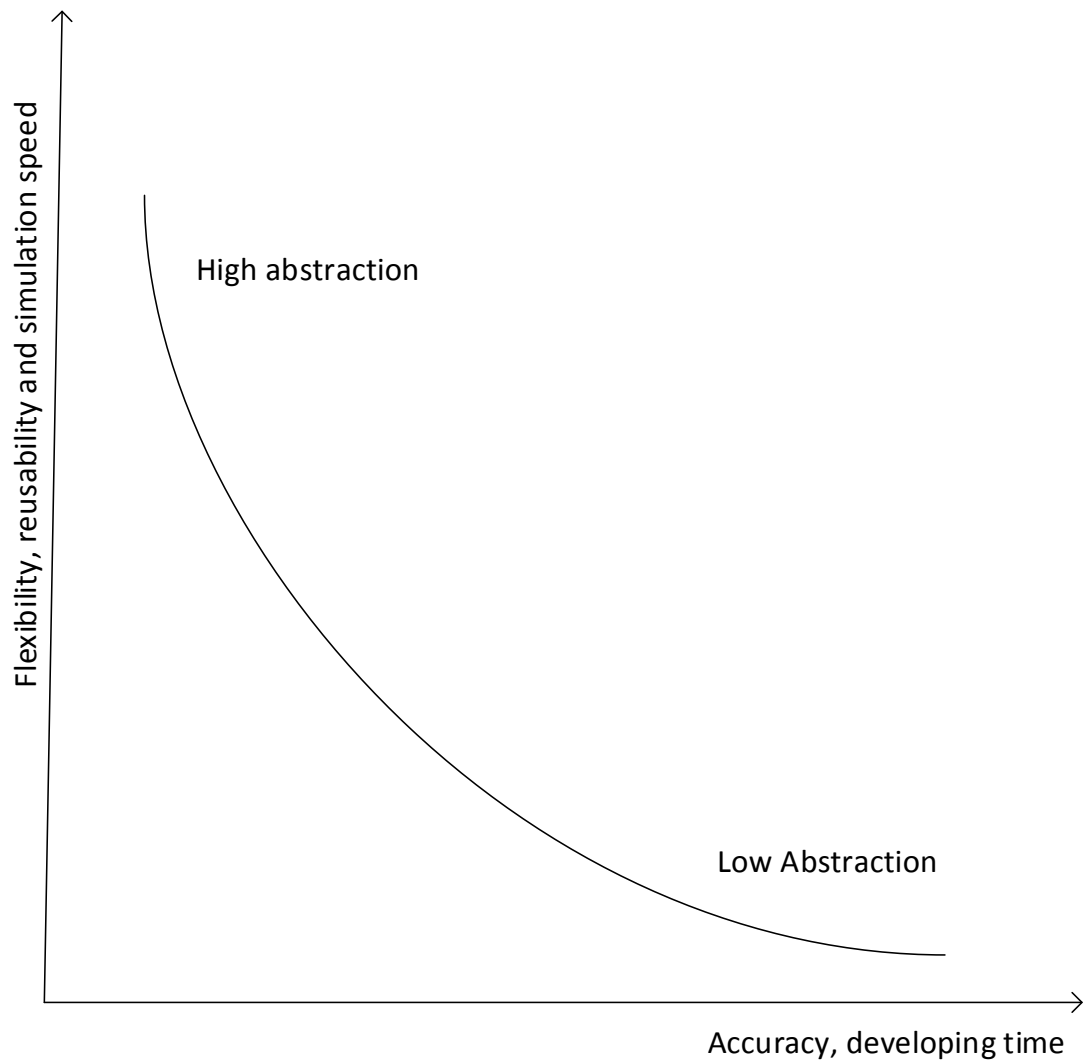


Figure 4. Curve describing the relation between the flexibility, reusability, simulation speed, accuracy and developing time in high and low abstraction models.

4. SCHEMATIC MODEL GENERATOR

Schematic Model Generator (SMG) is a tool developed by Cadence Design Systems to help creating behavioural models using currently SystemVerilog, Verilog-AMS or Wreal languages. There is also support for the SystemVerilog UDTs, which can be utilized to i.e. simulate impedance, voltage and current on a single net using real-type values. SMG provides a graphical environment to build models using a pre-made Building Block Texts (BBT) library, which can be considered as off-the-shelf parts to use in a schematic like environment, each having a specific function they perform. For example, the library offers BBTs that perform normal arithmetic operations, such as multiplication and subtraction, which can be utilized to e.g. manipulate real-type values representing an electrical parameter inside the model. For logic and electrical behavioural modelling, one can take advantage of the basic logic port and i.e. slew rate BBTs, which are also offered in the library.

4.1. Basic Concept

After the requirements for a model/system are clear, the model creation using the SMG begins by creating a model schematic using BBTs instead of real components. Simple example of an analog multiplexer controlled by a single-bit digital signal can be seen in Figure 5. This SystemVerilog example uses real-type value to represent the output voltage. The multiplexer BBT itself is an off-the-shelf part included in the SMG library with user controlled settings such as initialization of the output.

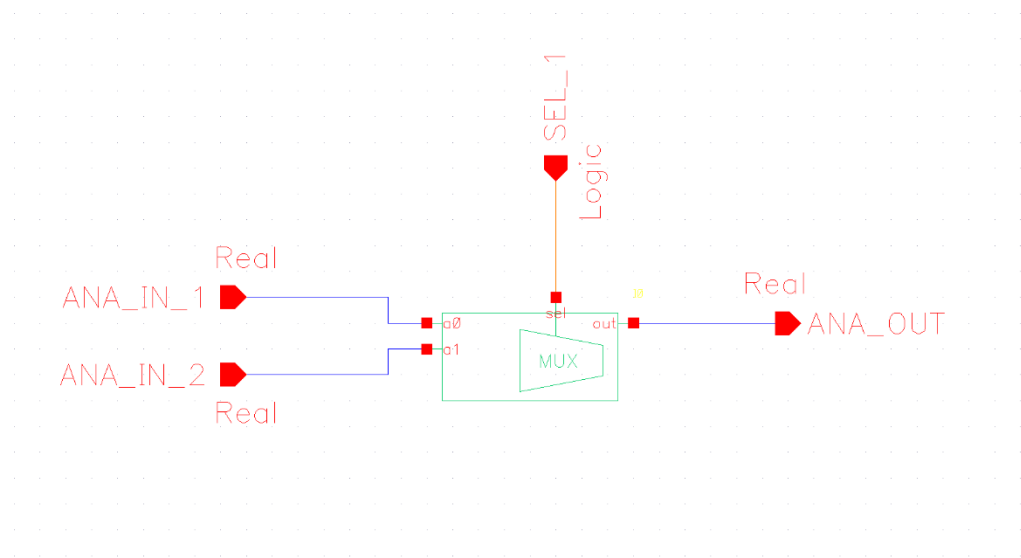


Figure 5. Model Schematic created for an analog multiplexer behavioural model utilizing real-type values.

SMG can generate a model description of the created model schematic in user selected language (SystemVerilog, Verilog-AMS, Wreal). In this example, a SystemVerilog file was generated, which can be seen in Figure 6. By creating a separate testbench and simulating in e.g. Cadence Design Systems Analog Design Environment (ADE) simulation environment, one can test the generated

SystemVerilog model by providing it normal electrical input test stimulus. During the elaboration phase, the Virtuoso Analog Mixed-Signal (AMS) simulator recognizes electrical-to-logic and electrical-to-real boundaries and inserts an E2L (Electrical-to-Logic) or E2R (Electrical-to-Real) module at the port to convert the electrical input by the user set rules.

```

7 // This model makes use of external definitions from one or more model files
8 // These files can be found in $CDSHOME/tools/dfII/etc/smg/bbtDefinitions.
9 // Add this directory to the simulator search path or to the local hdl.var file being used.
10 // E.g. in hdl.var: DEFINE NCVLOGOPTS -incdir $CDSHOME/tools/dfII/etc/smg/bbtDefinitions
11 `timescale 1ns/1ps
12 `define SMGTIMEUNIT 1e-9
13 `define SMGTIMEPRECISION 1e-12
14 `include "smgDefinitions.sv"
15
16 module ANALOG_MUX(ANA_IN_1,ANA_IN_2,ANA_OUT,SEL_1);
17                                     // Pin directions
18     input    ANA_IN_1,ANA_IN_2,SEL_1;
19     output   ANA_OUT;
20                                     // Pin types
21     real    ANA_IN_1,ANA_IN_2,ANA_OUT;
22     logic   SEL_1;
23                                     // Variable Declarations
24     integer selector_I0;             // Inst I0
25     real ANA_OUT_reg;               // Inst I0
26                                     // Digital Block
27     always begin                     // Inst I0
28         selector_I0 = SEL_1;        // Inst I0
29         if (selector_I0 == 1'bz)   // Inst I0
30             ANA_OUT_reg = `wrealZState; // Inst I0
31         else                         // Inst I0
32             case (selector_I0)      // Inst I0
33                 0 : ANA_OUT_reg = ANA_IN_1; // Inst I0
34                 1 : ANA_OUT_reg = ANA_IN_2; // Inst I0
35                 default: ANA_OUT_reg = `wrealXState; // Inst I0
36             endcase                // Inst I0
37         @(ANA_IN_1, ANA_IN_2, SEL_1); // Inst I0
38     end                             // Inst I0
39     assign ANA_OUT = ANA_OUT_reg;   // Inst I0
40
41
42 endmodule
43
44
45
46

```

Figure 6. The SMG generated SystemVerilog code of the model schematic seen in Figure 5.

The created simulation testbench can be seen in Figure 7. ANA_IN_1 input was set to 5V, ANA_IN_2 to 2.5V and SEL_1 switches from logical zero to one at 5ms and back to zero at 7ms. The obtained simulation results are visible in Figure 8. The results show that the ANA_OUT switches from 5 to 2.5 at 5ms and back to 5 at 7ms, proving that the example model works. One should note that the output is only a real value representation of the output voltage, not actual voltage that could drive e.g. resistor load. This would require use of UDT in the model to convert the real value output to an electrical one. The whole model design process using SMG is simplified in Figure 9.

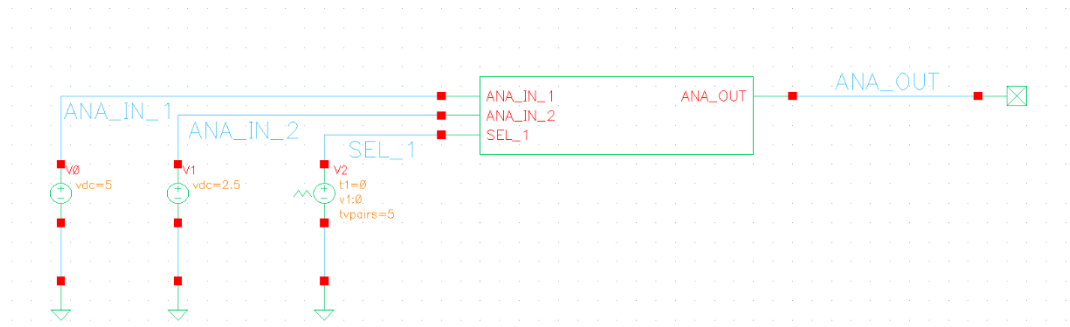


Figure 7. The testbench to test the created SMG generated SystemVerilog model.

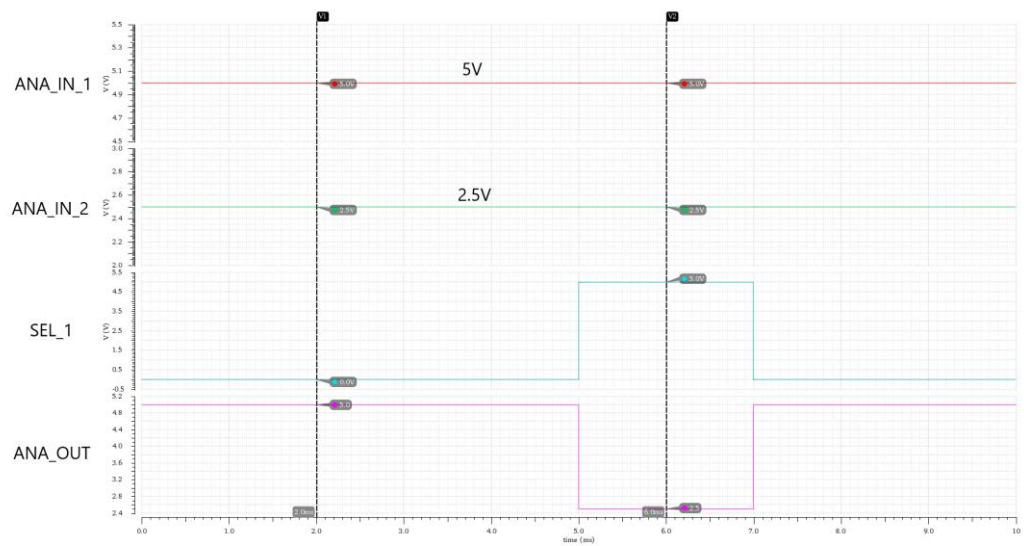


Figure 8. The simulation results of the analog multiplexer SystemVerilog model.

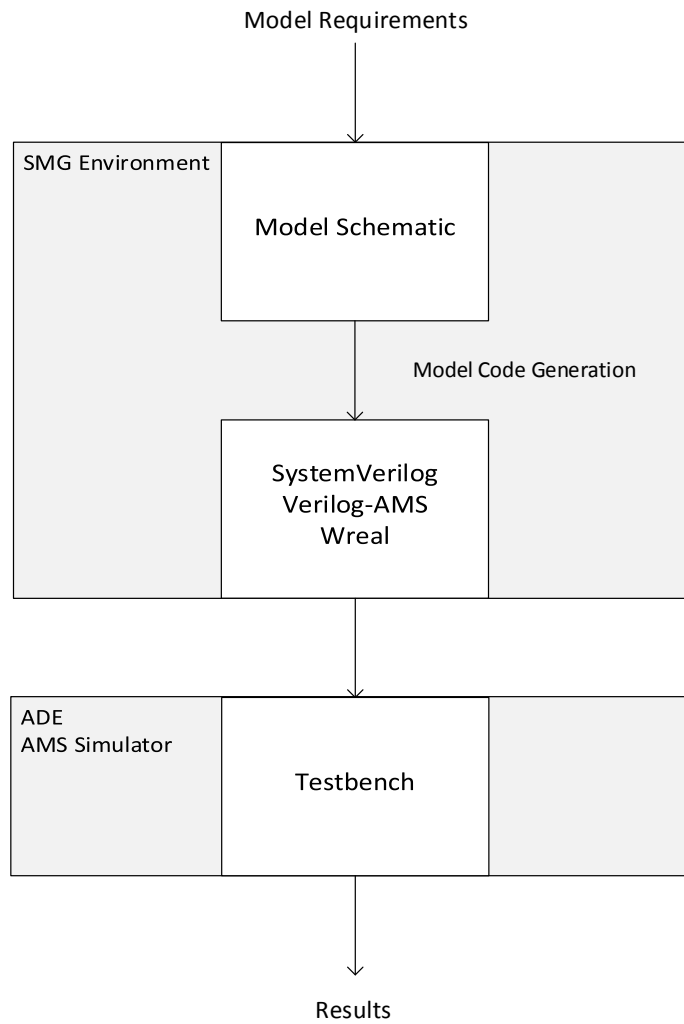


Figure 9. The analog behavioural model designing process using Schematic Model Generator simplified.

4.2. Target Users and Usage Options

SMG is aimed for analog and AMS designers who have knowledge of the Virtuoso environment but lack the experience and knowledge of behavioural modelling languages. Modelling experts as well as digital and verification engineers may benefit from the SMG technology too, even though text editors may provide more free environment for them. In the future, the requirement to use UDT may come from the need to model the RF paths in RF domain i.e. amplitude and frequency of the signal. SMG aims to provide a solution for everyone to be able to create such models. [18]

SMG generated code can be modified by hand. This gives the possibility to model some parts of the system using SMG and other parts with handwritten code. Mixing-and-matching may benefit experienced users or modelling experts in speeding up their model development. [18]

5. MODEL CREATION

SMG was utilized to create two separate models of different complexity using SystemVerilog-RN. Virtuoso version IC6.1.7-64b.500.12 was used to run SMG. The more complex model required timing accurate behavioral modelling whereas it was sufficient for the simpler model to only focus on modelling basic checks (e.g. supply range and bias) and the programmability, such as output voltage programming. By creating two different analog blocks, the advantages and limits of SMG could be thoroughly tested.

For both of the blocks, the most important aspect was to model the response to different control signals as precisely as possible to simulate the actual behavior of the IP. Transient effects, such as output voltage ripple, were left out due to the fact that they load the simulator for no reason as it was desired to keep the model as computationally light as possible.

5.1. Target Blocks

The first target analog block was a Low Drop-Out regulator (LDO) with a programmable output voltage, high impedance output mode and short circuit protection. Important electrical parameters to include were the minimum and maximum supply voltage, drop-out voltage and the input bias current and reference voltage accuracy. The design parameters that were targeted for modelling can be seen in Table 1.

Table 1. The target parameters of the LDO SystemVerilog model

Parameter	Information
Supply range	The model should give output only if the minimum required supply voltage exists
Programmable output	The output should be programmable according to the LDO datasheet
Input bias current and reference voltage checking	The programmed output should be valid only if the given bias currents and reference voltages are correct
High-Z mode	The output should be in high-z state if the high-z control signal is high
Pull-down state	The output should be in pulled-down state if the pull-down control signal is high
Drop-out voltage	One of the LDO main electrical parameters

Second target block was a Direct Current to Direct Current (DCDC) buck-converter (step-down converter) with Ultra Low Power (ULP) operating mode. The DCDC contained programmable output voltage, Pulse Width Modulation (PWM) frequency setting, trimming and different operation modes. Important analog parameters to model were the supply voltage range and the bias and voltage reference accuracy. Also, the buck-converter external coil behaviour had to be

modelled without overloading the simulator. The operation during the ULP-mode relied on the energy stored inside the coil, but clearly, modelling this electrical parameter too accurately would degrade the simulator performance, giving only little more benefit in inspecting the correct performance of the DCDC. The mode transitions which were either synchronous or asynchronous and relied on external digital control needed to be accurately modelled. To test the model, the external controller block had to be created to respond to the requests that the DCDC made. The target model parameters of the DCDC can be seen in Table 2.

Table 2. The target parameters of the DCDC SystemVerilog model

Parameter	Information
Supply range	The model should give output only if the minimum required supply voltage exists
Programmable output	The output should be programmable according to the DCDC datasheet
Input bias current and reference voltage checking	The programmed output should be valid only if the given bias currents and reference voltages are correct
Programmable PWM frequency	The PWM frequency must be programmable
PWM-to-ULP transition	Transition from the PWM-mode to the ULP-mode must be implemented as accurately as possible
ULP-to-PWM transition	Transition from the ULP-mode to the PWM-mode must be implemented as accurately as possible

5.2. Model Creation

5.2.1. Low Drop-out Regulator

LDO is a device used to generate lower voltage output of a higher supply voltage. In order to guarantee higher efficiency, it is best used for low current applications. The advantage of using LDO over a switching regulator is that it does not emit frequency disturbances to the receiving end and does not require an external, bulky coil.

The LDO modelling process began by modelling the electrical parameters, including the supply range, the bias current checking and the drop-out voltage. The main advantage of using real-type values is that one can perform normal arithmetic operations. Thus, the current supply voltage can be represented by performing a simple operation

$$V_{supply} = AVDD - AVSS$$

By comparing the obtained supply value with the minimum supply voltage stated in the LDO requirement documentation, one can determine if the supply level is

sufficient enough to power up the LDO. If the supply level is not high enough, the output of the model is pulled down to zero. On the SMG model schematic, this whole operation is done by first subtracting the negative supply level from the positive supply level, then comparing the result with the minimum requirement and finally multiplexing either the supply level or zero to the output for further usage. This can be seen in Figure 10.

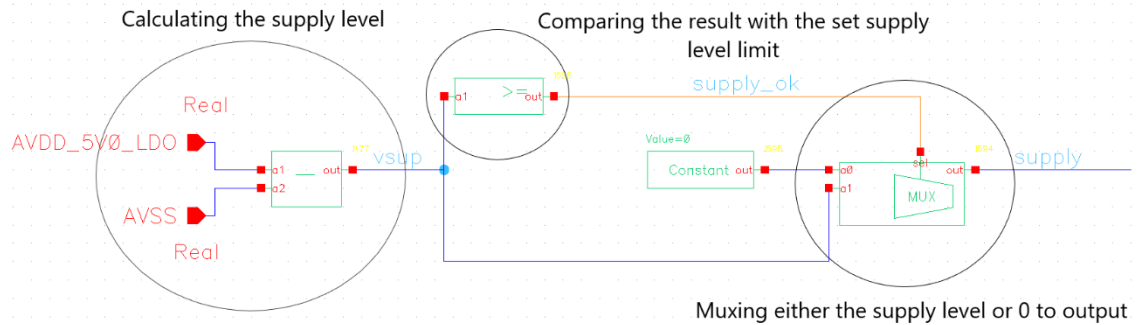


Figure 10. Simple way of checking the minimum supply voltage requirement.

Next, the bias current level checking had to be modelled. Bias currents are often used to generate different references. This time a 10uA bias current was used to generate a reference voltage of 0.65V using a 65kΩ resistor. Again, this was easily done by performing a multiplication operation, which can be seen in Figure 11.

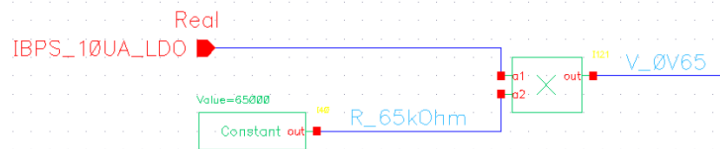


Figure 11. Example of generating real value representing internal reference voltage.

The simplest way of modelling the output voltage programming is an array of real value presentations of different output voltages that are just multiplexed according to the given register bit value. The LDO featured 6-bit selection for the output voltage, meaning total of 64 different outputs. This big array of values is rather painful to create in the model schematic without using a specific BBT to add an external text file including the control values for different output voltage levels. The external text file approach would be ideal considering the maintenance and readability of the model. However, this approach generated parts in the SystemVerilog code that were not part of the current SystemVerilog standard. To make the generated SystemVerilog code as compatible with other simulators as possible, the output voltage programming was done with the set of different real

values connected to a multiplexer BBT. This straight forward, but clumsy approach can be seen in Figure 12.

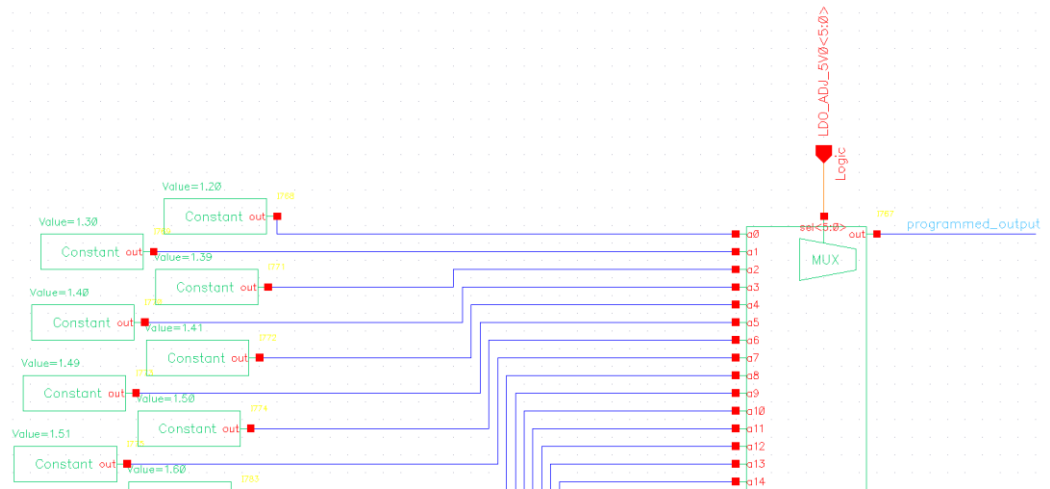


Figure 12. Multiplexing the real-type value representation of the LDO output voltage according to the given control bit value.

The generated internal references had to be compared against the specified ones to determine if all the internal references were correct and the output could be set accordingly to the target level. The model should somehow indicate that the inputs it has been given are valid or invalid. In case of e.g. wrong bias current level or input reference voltage, the output level should differ from the desired output voltage. Also, if the supply voltage was lower than the set output voltage plus the drop-out voltage, the target value should not reach the set output voltage level.

Finally, the main output control had to be implemented. The output assigning was implemented in a way that the target value is continuously measured. The target value itself is the result of the programmed output voltage, the supplies and the references. The set target value then continues to the output through multiplexers which check if the output was programmed to high-z state or pulled down state. High-z state was implemented as a negative value to clearly show the difference to a normal output. The final output value is created by driving the calculated real-type value through a slew rate BBT which makes the changes in the output gradual. The output control can be seen in Figure 13 and the symbol of the LDO can be seen in Figure 14.

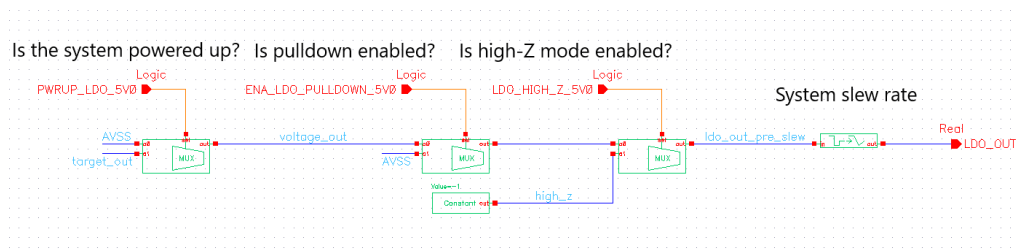


Figure 13. The LDO model schematic output control.

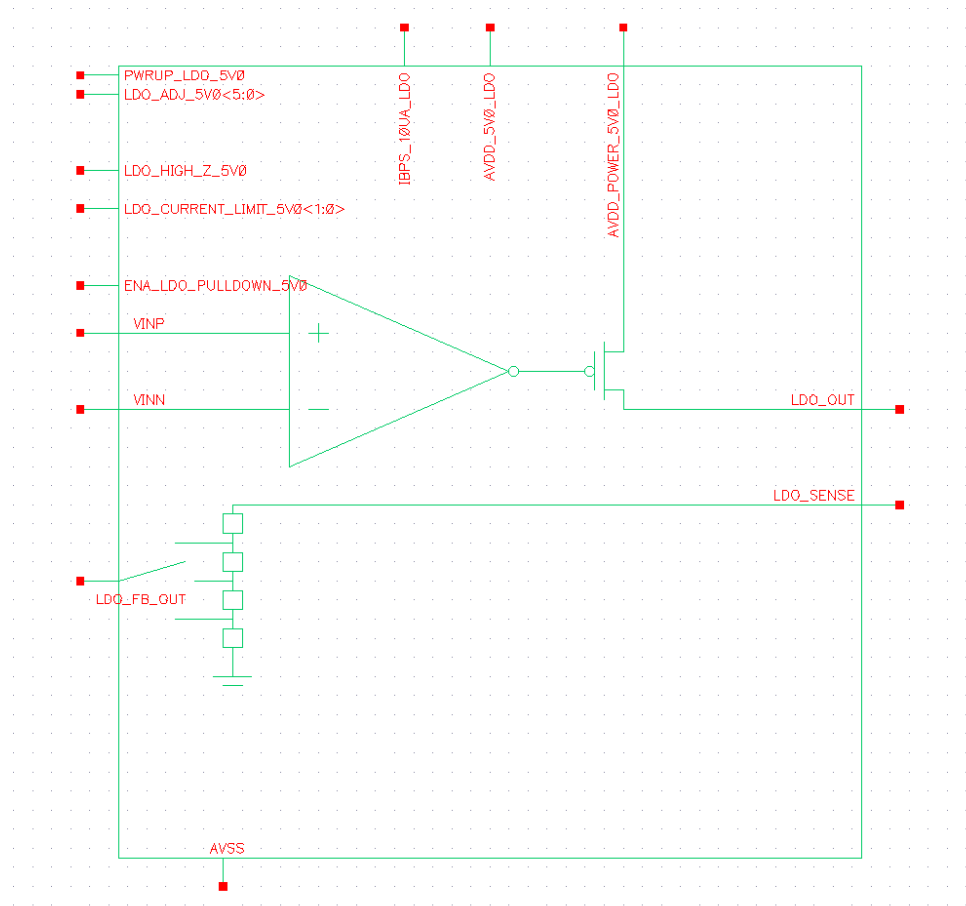


Figure 14. The symbol of the created LDO SystemVerilog RN model.

5.2.2. DCDC

The target DCDC was a buck-converter, which is also known as a step-down converter. Buck-converter is a device used to generate lower voltage out of a higher supply voltage, just like LDO. The advantage of using buck-converter over an LDO is that buck-converter is able to deliver low voltages with better efficiency than an LDO.

The DCDC IP features two main operation modes, which are the PWM-mode and the Ultra-Low Power-mode (ULP). In PWM-mode, the external coil of the converter is refreshed every clock cycle and in the ULP-mode the coil is only refreshed when the output voltage is below the target value. The DCDC was specified to send syncing pulse in PWM frequency to one of its status outputs during the PWM-mode. This output indicates the frequency of the programmable PWM signal and that the DCDC is in the PWM-mode. The DCDC was specified to always start up in the PWM-mode, after which it can switch to the ULP-mode after receiving the correct control bits from the external controller. During the ULP-mode, the DCDC occasionally requests for control signals and references from the external master block to keep the external coil refreshed. Transition back to the PWM-mode is done by providing the correct control bits. In case the references or control bits are inaccurate or incorrect, the DCDC output will not settle to a correct level or the mode transition does not take place. The simplified pseudo state chart describing the DCDC behaviour can be seen in Figure 15.

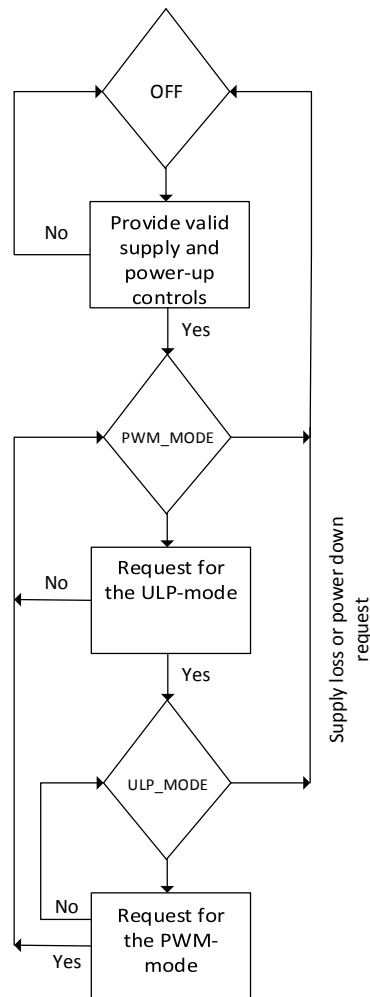


Figure 15. Simplified pseudo state chart describing the DCDC IP behaviour.

The DCDC model creation process began with the same procedure as with the LDO i.e. creating supply range, bias and reference voltage checks. The mode transitions required more control signals than the LDO model. In order for the model to be valid, the timing of the control signals had to be exact to represent the actual behaviour. The logical behaviour was first implemented in the model schematic using single logic gates and D-flip-flops, which lead into a non-working and non-maintainable model. The problem with the handmade logic is that if even a small part of the system behaviour is changed, a big part of the model schematic may have to be changed. Also, the chance for human error increases and debugging may become really demanding. This is clearly unacceptable as the main reason for using SMG was to provide a fast and easy way to create models.

To get around this problem, the combinational logic can be created using handwritten SystemVerilog code and mixing it with the SMG generated code. There exist at least two options to do so. One option is to add the handwritten SystemVerilog code to the SMG generated SystemVerilog file afterwards. The downside to this approach is that every time the code is regenerated, the handwritten parts need to be re-added as SMG always creates a completely new file. An alternative way is to create own BBTs that can be placed in the model schematic. In this case, the handwritten SystemVerilog is included with the BBTs and the mixing

with the SMG generated SystemVerilog code is done in the SMG model file generation phase. The latter was the chosen option for the DCDC model.

The handwritten SystemVerilog code was first used to create a state machine BBT to keep track on the current state of the system and change the behaviour of other blocks in the system. Some of the state machine state transition were synchronous and took advantage of the PWM-signal inside the model to count clock cycles for requesting signals from the external control block and expecting an answer in a given time. During the ULP-mode, when there is no PWM-signal, the asynchronous transitions were controlled by the external controller block. The asynchronous transitions were triggered by only certain events and the DCDC internal PWM clock signal took over the control of the state changes after switching back to the PWM-mode from the ULP-mode. This assures that the model state changes are controlled.

In order to simulate the effects taking place especially during mode transitions, the comparator inside the DCDC had to be modelled more precisely. Even though the comparator did not include a lot of combinational logic, it was best to handwrite the SystemVerilog code. The comparator measures the difference between the target voltage and the output to indicate if the output level is high enough. In case the output level is too low, the comparator triggers and causes the DCDC to take action. The low threshold of the comparator is dependent on the operating mode. In the ULP-mode, the output normally falls slightly below the low threshold level, after which the DCDC requests for a coil refresh. If the refresh is not given, the output level continues falling.

The simulation was required to stay as computationally light as possible. Therefore, the ripple voltage caused by the change of the charge in the output capacitor was included as a separate input signal to the comparator BBT and not as actual output signal that would be visible as alternating current (AC) signal on top of the output direct current (DC) signal in the simulation.

The SMG was used to generate a single SystemVerilog file of the model schematic including both off-the-shelf BBTs and self-created BBTs. Parts of the handwritten SystemVerilog among the SMG generated code can be seen in Figure 16.

```

always begin
    dvlstp_I897 = (1e6)* $mg_min( $mg_abstime-tlast_I897,1e-09); // Inst I897
    dvlstp_I897 = (10000)* $mg_min( $mg_abstime-tlast_I897,1e-09); // Inst I897
    tlast_I897 = $mg_abstime; // Inst I897
    if (CAP_CHARGING == `wrealXState) begin // Inst I897
        lastValue_I897 = CHARGE; // Inst I897
        CHARGE = `wrealXState; // Inst I897
        @(CAP_CHARGING); // Inst I897
    end // Inst I897
    else if (CAP_CHARGING == `wrealZState) begin // Inst I897
        @(CAP_CHARGING); // Inst I897
    end // Inst I897
    else if ((CAP_CHARGING == CHARGE && CAP_CHARGING - CHARGE <= dvlstp_I897) || (CAP_CHARGING <= CHARGE && CHARGE - CAP_CHARGING <= dvlstp_I897)) begin // Inst I897
        CHARGE = CAP_CHARGING; // Inst I897
        @(CAP_CHARGING); // Inst I897
    end // Inst I897
    else if (CAP_CHARGING > CHARGE) begin // Inst I897
        CHARGE = CHARGE+dvlstp_I897; // Inst I897
        #(1.0); // Inst I897
    end // Inst I897
    else if (CAP_CHARGING < CHARGE) begin // Inst I897
        CHARGE = CHARGE-dvlstp_I897; // Inst I897
        #(1.0); // Inst I897
    end // Inst I897
    else if (CHARGE == `wrealXState || CHARGE == `wrealZState) begin // Inst I897
        CHARGE = lastValue_I897; // Inst I897
        #(1.0); // Inst I897
    end // Inst I897
end // Inst I897
always @(TARGET_VOLTAGE or SW1 or CHARGE) // Inst I898
begin // Inst I898
    output_with_ripple = SW1 + CHARGE; // Inst I898
    voltage_difference = TARGET_VOLTAGE - output_with_ripple; //Calculate the voltage difference to determine if the HCOMP has triggered // Inst I898
end // Inst I898
always @(power_up_status or supply_status) //Check that the HCOMP is turned on // Inst I898
begin // Inst I898
    if(power_up_status == '1'b1 && supply_status == '1'b1) // Inst I898
        begin // Inst I898
            hcomp_on = '1'b1; // Inst I898
        end // Inst I898
    else // Inst I898
        begin // Inst I898
            hcomp_on = '1'b0; // Inst I898
        end // Inst I898
end // Inst I898
always_comb //Check, if the comparator hysteresis threshold has been crossed // Inst I898
begin // Inst I898
    if(ULP_MODE_5V0 == '1'b1) // Inst I898
        begin // Inst I898
            if(hcomp_on == '1'b1) // Inst I898
                begin // Inst I898
                    if(voltage_difference >= 18e-3) // Inst I898
                        begin // Inst I898
                            lower_hysteresis_cross = '1'b1; // Inst I898
                            upper_hysteresis_cross = '1'b0; // Inst I898
                        end // Inst I898
                    else if(voltage_difference <= -18e-3) // Inst I898
                        begin // Inst I898
                            lower_hysteresis_cross = '1'b0; // Inst I898
                            upper_hysteresis_cross = '1'b1; // Inst I898
                        end // Inst I898
                end // Inst I898
            end // Inst I898
        end // Inst I898
    end // Inst I898
end // Inst I898

```

Figure 16. Mix-and-match of handwritten SystemVerilog with the off-the-shelf BBTs in the generated SystemVerilog RN code of the DCDC.

5.3. Model Testing

5.3.1. Test Stimulus

The models had to be tested in order to validate their correspondence to the specified performance. The testbench could have been implemented by inserting a voltage source to each input pin of the model. However, the timing of the test signals, especially for the DCDC model, would have been nearly impossible to implement and one could not be sure if the wrong behaviour was caused by the bug in the model or the testbench inaccuracy. In order to create test stimulus that described the digital control accurately, the test stimulus was created as a separate SystemVerilog file and attached to the testbench.

The creation of multiple test cases easily is one of the main advantages for using SystemVerilog. One could easily create a test to e.g. go through all the LDO programmable output values. Especially the DCDC model testing benefits from creating test tasks to switch operation mode from one to another. The DCDC model testbench can be seen in Figure 17. The symbol of the test stimulus is the same as for the DCDC but the direction of the input and output ports is flipped, therefore creating a perfect interface.

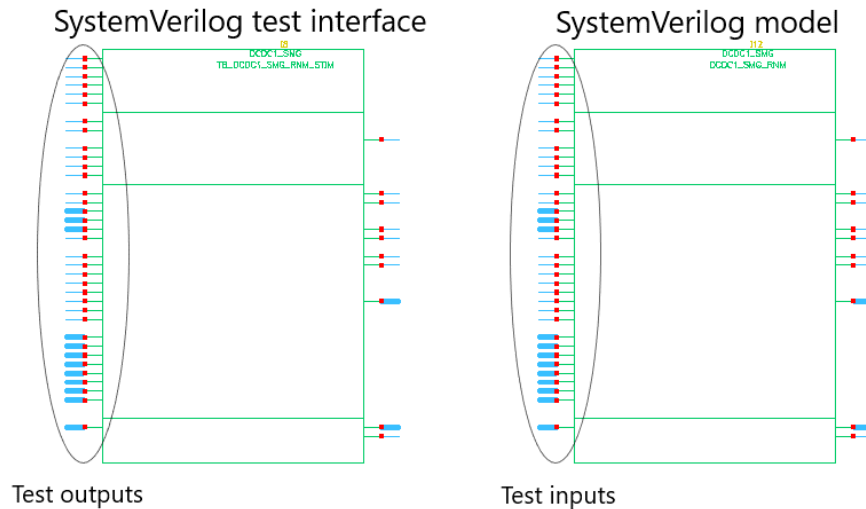


Figure 17. The DCDC model simulation testbench. Left side block provides the test stimulus to the SystemVerilog model (right side block).

Creating the test stimulus began with writing different tasks that made the model change its behaviour accordingly. In Table 3 and Table 4 one can see the created tasks for the LDO and DCDC models. It was important to test all the possible output options and that the state changes took place under control. Use of tasks made the test pattern more readable and maintainable, as if the design and the model was to change, only the task would have to be changed. Parts of the written test stimulus code can be seen in Figure 18.

Table 3. The LDO test stimulus tasks

Task	Description
Initialize	Initializes all the ports
Supplies up	Sets the supplies up
Supplies down	Sets the supplies down
Program all	Programs all the necessary values, such as the output voltage, the current limit and the bias trimming
Test all target values	Goes through all the target values
Test all shutdown methods	Tests every shutdown method, including the pulldown and the high-z state

Table 4. The DCDC test stimulus tasks

Task	Description
Initialize	Initializes all the ports
Supplies up	Sets the supplies up
Bias up	Sets the required bias current up
Power down	Powers everything down
Program all	Programs all the necessary nominal values, including the output voltage, the PWM frequency, the slew rate and the trim values
Ramp up	Goes through the start-up sequence to ramp up the DCDC output voltage
PWM-to-ULP	Goes through the pattern to change the state from the PWM-mode to the ULP-mode
ULP-to-PWM	Goes through the pattern to change the state from the ULP-mode to the PWM-mode
ULP refresh request	Sets the required biases, references and enable signals when needed

```

////////////////////////////////////
////////MODE CHANGES
////////////////////////////////////

task ramp_up(); //Ramp up DCDC1 output voltage
$display("Ramp up DCDC1 output voltage", $time);

@(posedge clk)
begin
    DCDC1_PWM = 1'b1;
    PWRUP_5V0 = '1;
    EN_HPMODE_5V0[0] = 1'b1;
    HP_BIAS_5V0 = 1'b1;
    REFRSH_ULP_REF_FBCK_5V0 = 1'b1;
    VREF0V65_HP_DCDC = 0.65;
    IBPSR_5U_DCDC = 5e-6;
    IBNSR_5U_DCDC = 5e-6;
end

@(posedge clk)
begin
    EN_HPMODE_5V0[3:1] = 3'b111;
    HP_REFRSH_5V0 = 1'b1;
end

@(posedge clk); //Wait for two clock cycles
@(posedge clk);

@(posedge clk)
begin
    EN_HPMODE_5V0[4] = 1'b1;
    EN_SFTSTRT_5V0 = 1'b1;
    PWM_MODE_5V0 = 1'b1;
end

endtask

```

Figure 18. The DCDC output ramp-up task.

5.3.2. Assertions

It may be insufficient to only trust one to see the errors in the model simulation results. SystemVerilog has an advantage of enabling using assertions that give the user a warning when something goes wrong. The assertion module was created as a separate module to be added in the testbench. This assertion module captures the signal traffic between the model and the test stimulus module. The assertion module can then determine if the model response was correct to the given signals. The SMG offers a way to create assertions, so technically the assertion module could have been created in the SMG graphical environment. Writing them by hand showed to be more convenient for more complex checks. For simple checks, such as supply checking, the SMG was applicable. One should note that the assertions could have been included in the model itself instead of using a separate module but this time, the separate module was used to illustrate the possibility of using assertions. In Figure 19, one can see the updated DCDC testbench and in Figure 20, parts of the written assertion module for the DCDC model.

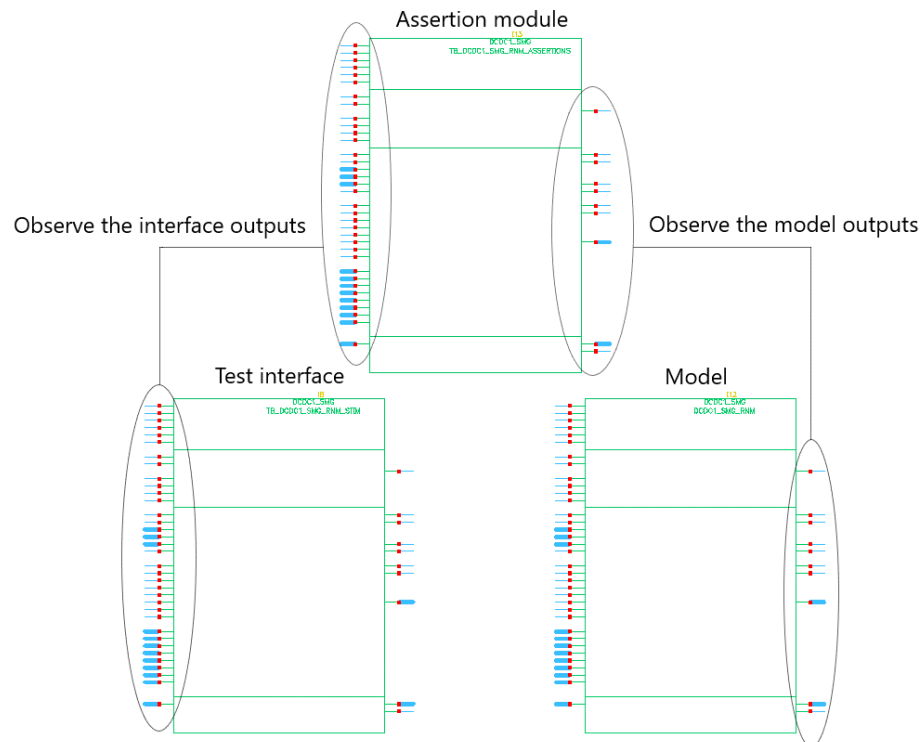


Figure 19. The updated DCDC model testbench. The assertion module (the top block) observes the test inputs and the model outputs.

```

always @(posedge PWM_MODE_5V0)
  PWM_MODE_TRANSITION: assert((EN_HPMODE_5V0[4:0] == 5'b11111 && REFRSH_ULP_REF_FBCK_5V0 == 1'b1 && EN_
    (REFRSH_ULP_REF_FBCK_5V0 == 1'b1 && EN_HPMODE_5V0[4:0] == 5'b11111 && BYPASS_SFT
    $display("DCDC: Correct control bits in PWM-mode transition given");
  else
    $warning("DCDC: Incorrect control bits in PWM-mode transition given!");

always @(posedge ULP_MODE_5V0)
  ULP_MODE_TRANSITION: assert (REFRSH_ULP_REF_FBCK_5V0 == 1'b0 && HP_REFRSH_5V0 == 1'b0 && HP_BIAS_5V0
    PGOOD_5V0 == 1'b1)
    $display("DCDC: Correct control bits in ULP-mode transition given");
  else
    $warning("DCDC: Incorrect control bits in ULP-mode transition given!");

```

Figure 20. Example of the created assertions for the DCDC model.

5.3.3. Simulation

First, the LDO model was tested. The test pattern is planned so that all the LDO programmable output values are checked as well as all the shutdown methods (pull-down, high-z, power down). In Figure 21, one can see the LDO model output. It can be seen that the testbench first programs a random output voltage, then goes through all the possible output values from the top to bottom and back to the top. Finally, the testbench tests the pull-down and high-z mode. According to the visual inspection, the LDO model works as intended.

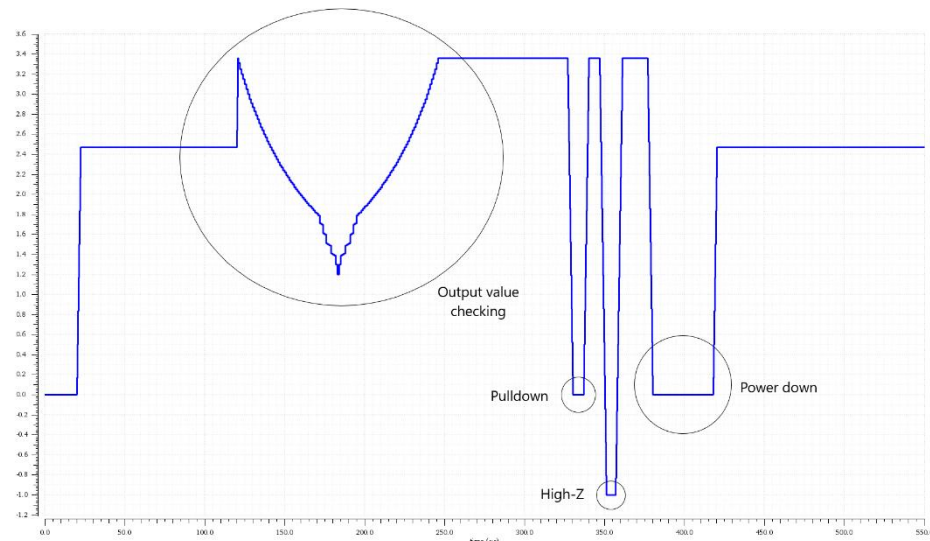


Figure 21. The LDO model output.

Next, the DCDC model was tested. In Figure 22, Figure 23 and Figure 24 one can see the behaviour of the model at certain important moments. In Figure 22, the DCDC model output is normally ramped up and the DCDC model enters the PWM-mode. After a while, the testbench sets new control signals and the DCDC model requests for the ULP-mode from the external controller. This leads to a visible change in the output, which can be seen in Figure 23. After being in the ULP-mode for a while, the testbench sets new control signals again and the DCDC model correctly requests for a change to the PWM-mode. According to the assertions and visual inspection, the DCDC model was working correctly at the given level of model abstraction.

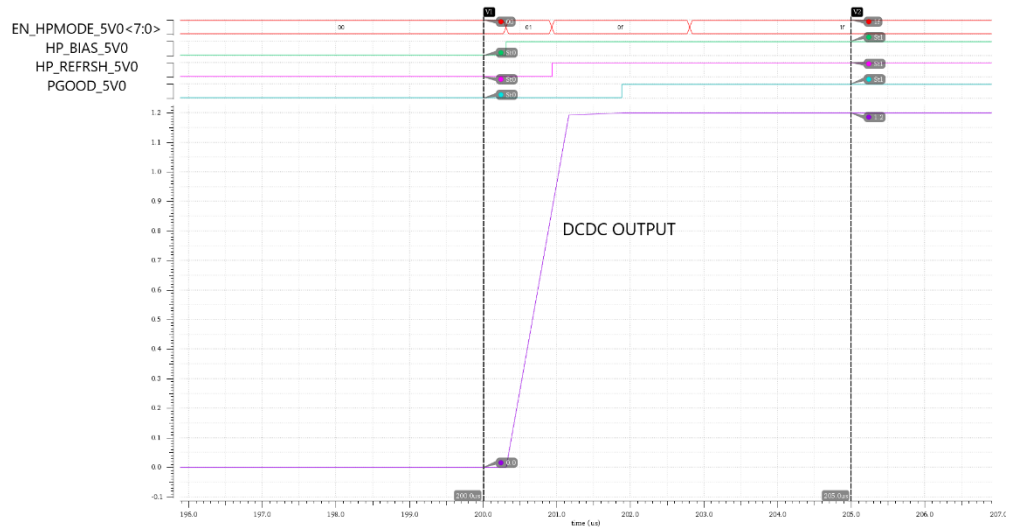


Figure 22. The DCDC model output voltage ramp up.

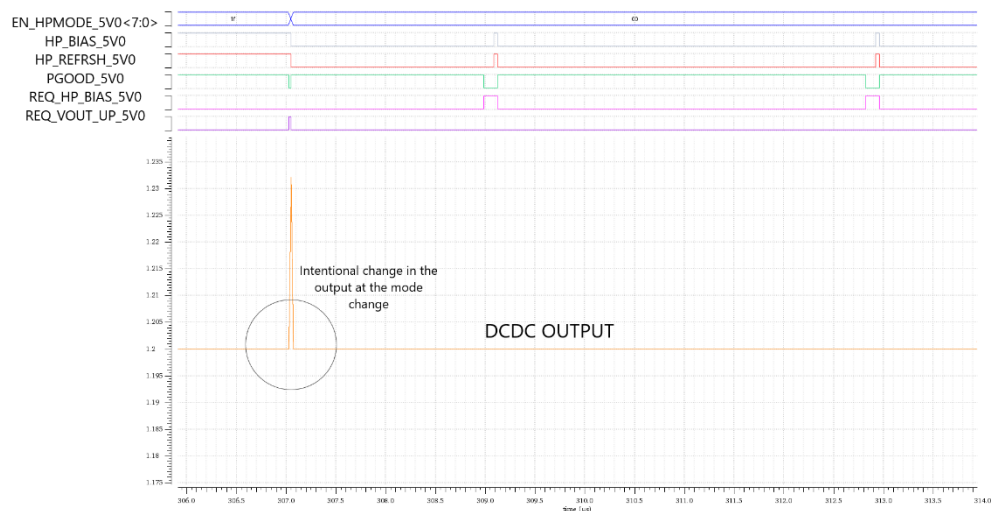


Figure 23. The DCDC model transition from the PWM-mode to the ULP-mode.

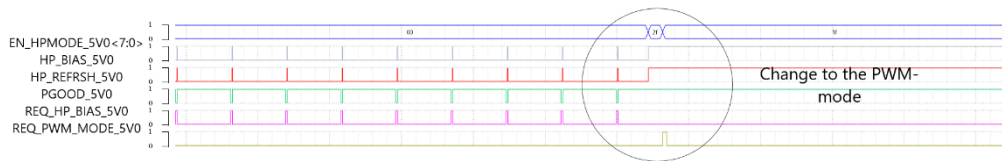


Figure 24. The DCDC model transition from the ULP-mode to the PWM-mode.

5.4. Simulation Results and Model Accuracy

Both the models aimed to model the actual design in an abstract manner. Both succeed to model the most important aspects, such as the response to external digital control, as they operate correctly only if they are given the correct control bits. This is important so that the software team can take advantage of the models for the software development. Misunderstanding in the operation could cause the software

development to go slightly wrong and delay the verification phase when testing the transistor level implementation with the related software.

Both the models lack most of the electrical parameters that can be measured in the transistor level description. Both have only real-type outputs which means that the outputs are not capable of driving any real load. If the electrical behaviour was modelled precisely, driving different loads would be visible in the change of the performance, especially for the DCDC as the energy stored in the output capacitor and the coil would decrease faster when driving a low ohmic load [20]. This would mean that in the ULP-mode the DCDC would request for the coil refresh more frequently. Different load would also affect the output voltage ripple, a parameter which is completely missing from the DCDC model. The LDO had a programmable output current limit and as the model is not capable of driving “real” current, the model doesn’t change its behaviour even though different current limit values can be programmed.

The LDO and DCDC transistor level descriptions use feedback to determine the current output voltage level. The feedback was left out in both models as it would just load the simulator unnecessarily. The feedback stability issues could not be inspected with the behavioural model anyway which is one of the most important aspects of the feedback.

5.5. Model Maintenance

The LDO model was created by using only the SMG graphical interface and the DCDC by mixing and matching the SMG generated code and the handwritten SystemVerilog. The SMG environment was sufficient in creating a maintainable model schematic for the LDO as it did not have a complex control scheme with accurate timing. Even if the number of controls increased or changed, it would be easy to add the changes in the existing model schematic.

For the DCDC, using just the SMG library provided BBTs would have resulted in a non-maintainable model schematic. In case the model’s combinational logic was made using the off-the-shelf BBTs, in other words single logic ports, a simple change in the control scheme could affect big parts of the model schematic. Behavioural models are often used for architectural exploration and to let the software development to begin in an early phase. This means it is likely that faults are found in the design and as a result, changes to original plan may be needed. Handwritten code is far more adaptable in the matter of maintaining the model as even big parts of the control scheme can be changed just by changing i.e. the state machine. SMG did not provide a way to simply create state machines except by adding handwritten SystemVerilog. A finite-state machine generator would be a great addition for the tool.

5.6. Design Effort in the Model Generation

Considering the design effort, the LDO model creation may have benefitted from the use of SMG compared to creating the model completely by hand. SMG generated all the SystemVerilog framework around the model and the user could just focus on the modelling. The model was reasonably quickly developed and the model accuracy was likely at the same level as it would have been if the model was handwritten. It is

clear that this model could have been created by a user not familiar with SystemVerilog syntax.

The DCDC model creation possibly did not benefit from the use of SMG. Only the simplest parts of the model could be created in the SMG environment, including i.e. supply and reference checks. Most of the time spent was on the handwritten BBTs to cover the complex parts of the model. This led into thinking that the whole model could have been created in a separate SystemVerilog-file without the use of SMG and included to the testbench.

6. COMPATIBILITY OF DIFFERENT MODELS

Today, the designed systems are often a result of cooperation between different teams. Therefore it is an important aspect that the models are interoperable between the teams involved with the design, meaning that the model should be compatible with different simulation environments in order to enable fully validating both the model and the design. Also, it must be reasonably easy to compare the model against the actual design. Therefore, the LDO model was simulated in the same testbench with the transistor level design to locate the design discrepancies and the DCDC model in the Mentor Questa digital simulation environment to examine the cross-compatibility of the SMG generated SystemVerilog models.

6.1. Simulation with Transistor Level Description

The LDO model interoperability with transistor level description was examined by creating a new testbench to use with the LDO SystemVerilog model and transistor level design. Spectre simulator does not support SystemVerilog. One could simply use AMS simulator to simulate both the SystemVerilog model and the transistor level description in a single simulation run but the most convenient approach was to create separate testcases in the single testbench. Therefore, the testbench is the same for both the transistor level design and the model but the user determines which one is run at the time. By separating the transistor level design and the model from each other, they can be run in parallel using their corresponding, dedicated simulators (spectre and AMS) which makes the simulation faster than running a single larger simulation using just AMS simulator.

Even though the transistor level design and SystemVerilog model may both operate properly when it comes to state changes and programmability, one could possibly miss errors such as a wrong output value. Like mentioned earlier, the validated model can be utilized as an executable specification. The testbench was set to measure the differences between the model and transistor level design outputs. In Figure 25, one can see the results of a simple test. First, the testbench programs a random LDO output value and then tests the pulldown state and high-z state. The results show a problem with the transistor level design as its output is not pulled down to ground level in the pulldown state! Other than that, the design passes the simple test, which can be seen in the ADE testbench output value comparison seen in Figure 26. The model utilizes negative output value to represent the high-z state.

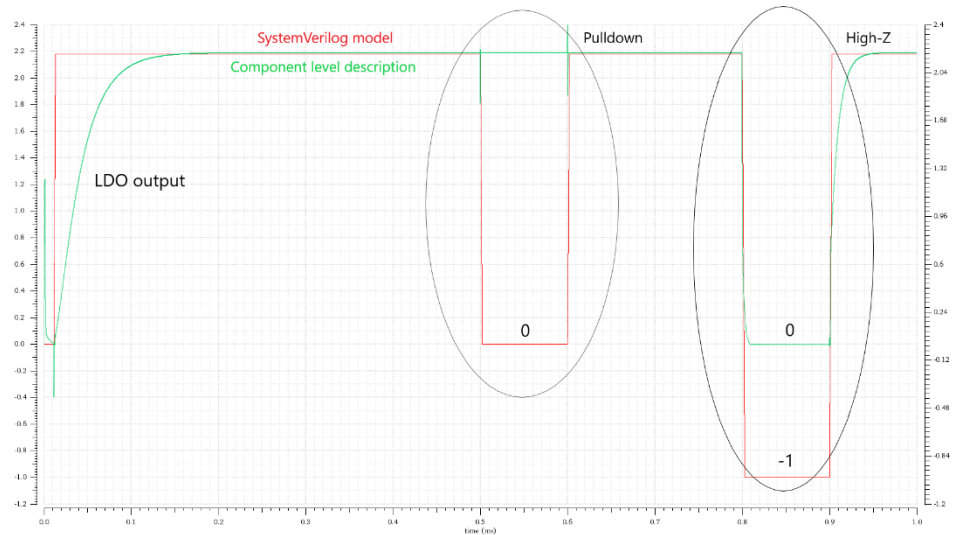


Figure 25. Comparison between the SystemVerilog model (red) and the component level description (green). The output of the transistor level design is not pulled down to ground level at the pull-down state (time 550us), which may have been caused by an error in the analog design.

TRANSIENT_SPICE	/LDO_OUT						
TRANSIENT_SPICE	LDO_OUTPUT_VALUE_STARTUP	info			2.189	2.189	2.189
TRANSIENT_SPICE	LDO_OUTPUT_VALUE_PULLDOWN	< 50m	fail		2.191	2.191	2.191
TRANSIENT_SPICE	LDO_OUTPUT_VALUE_HIGH_Z	info			-14.42u	-14.42u	-14.42u
TRANSIENT_MODEL	/LDO_OUT						
TRANSIENT_MODEL	LDO_OUTPUT_VALUE_STARTUP	info			2.179	2.179	2.179
TRANSIENT_MODEL	LDO_OUTPUT_VALUE_PULLDOWN	range -0.01 0.01	pass		488.9u	488.9u	488.9u
TRANSIENT_MODEL	LDO_OUTPUT_VALUE_HIGH_Z	range -1.01 -0.99	pass		-999.5m	-999.5m	-999.5m
TRANSIENT_MODEL	MODEL_VS_SPICE_STARTUP_DIFFERENCE	< 50m	pass		9.673m	9.673m	9.673m
TRANSIENT_MODEL	MODEL_VS_SPICE_PULLDOWN_DIFFERENCE	< 50m	fail		2.191	2.191	2.191
TRANSIENT_MODEL	MODEL_VS_SPICE_HIGH_Z_DIFFERENCE	info			999.5m	999.5m	999.5m

Figure 26. Assembler setup to compare the LDO transistor level design and the SystemVerilog model.

6.2. Simulation in Questa Environment

The SMG generated SystemVerilog model and the created stimulus and assertion modules were taken to the Questa environment. In order to simulate, the simulator required knowledge of how to connect the provided model, assertion and stimulus modules. For one to be truly sure that the simulation matches the analog environment simulation, both the digital and analog simulation environments should share the same top level netlist. Therefore, the top level netlist of the modules generated by the Cadence Design Systems AMS simulator was utilized in Questa environment to connect the modules. The difference between the two environments can be seen when going through the top netlist of the AMS simulator file. The AMS simulator top netlist utilizes “wire” net type to connect the modules to each other. This is not a mistake but it matters how the simulator treats the wire between the modules. Single wire between two logic-type or real-type ports generated only warnings in Questa during the compilation phase but the DCDC also included a real-type analog bus, which gave an error during the simulation. The wire-type bus had to be defined to real-type by hand. Also, the rest of the wire-type connections were either defined as logic or real depending on the connecting port to get rid of all the warnings during

the compilation. This was not a big change to do but prevented from using unchanged top netlist after making changes to the model. The modified top netlist can be seen in Figure 27.

```

module TB_DCDC_SMG_RNM;

// Buses in the design

logic      [7:0] EN_HPMODE_5V0;
logic      [7:0] EN_ULPMODE_5V0;
logic      [2:0] PRG_SAWGEN_5V0;
logic      [3:0] PRG_VOUT_5V0;
logic      [7:0] PWRUP_5V0;
logic      [7:0] TESTMODE_5V0;
logic      [2:0] TRM_SAWGEN_5V0;
logic      [3:0] TRM_VOUT_ULP_5V0;
logic      [2:0] PRG_P_SLEW_5V0;
logic      [2:0] PRG_N_SLEW_5V0;
logic      [3:0] OUT_CALBRT_REG_5V0;
logic      [3:0] TRM_ULP_FEEDBACK_5V0;
logic      [3:0] TRM_VREF_DCDC_5V0;

real ATST  [1:0];

logic      PGOOD_5V0,REQ_HP_BIAS_5V0,SYNCOUT_5V0,EN_HYST_H_5V0,REFRESH_ULP_REF_FBCK_5V0,SYNCIN_5V0,HP_REFR
real      AVDD_POWER_5V0,AVSS_POWER,SW1,AVDD_5V0,AVSS,DVDD_5V0,DVSS,IBNSR_5U_DCDC,IBPS_7N_DCDC,VO1,VREF0

TB_DCDC1_SMG_RNM_STIM  I8 ( .TRM_SAWGEN_5V0(TRM_SAWGEN_5V0[2:0]),

```

Figure 27. Part of the modified top netlist.

The final issue preventing from directly using the generated SystemVerilog was a difference in how the simulators treated the real-type nets and pins. Cadence Design Systems supports the floating net state “X” and the high impedance net state “Z” for real-type values. These states are not part of the current official SystemVerilog standard. Some BBTs that were in the LDO and DCDC model schematic included definitions of floating and high impedance net states and therefore, they were generated in the SystemVerilog file by SMG. Today, the SMG environment doesn’t provide a switch to turn this option off. Even though this non-standard SystemVerilog works without changes in ADE environment, the lines describing real-net behaviour in X- or Z-state need to be adjusted to comply with the SystemVerilog standard in order to ensure cross-compatibility in other simulation environments.

After making the required changes, the SMG generated model was ready for the simulation in Questa environment. In Figure 28, one can see the Questa simulation results of the DCDC model. The results show that it was possible to transfer the model simulation setup with small changes to different simulation environment and run the simulation using the same simulation testbench.

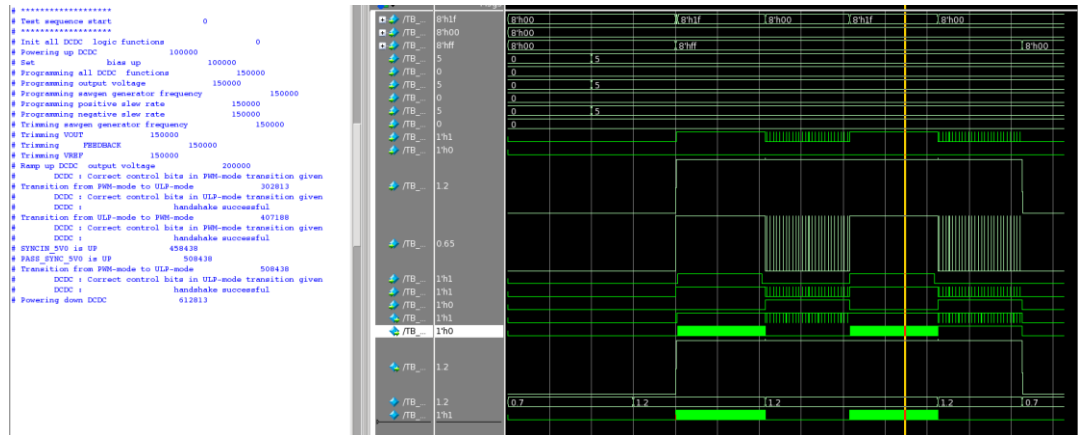


Figure 28. Simulation results of the DCDC SystemVerilog model in the Questa environment.

7. DISCUSSION

SMG definitely has provided an alternative way for designing analog block behavioural models. However, SMG may need some adjustments to be an option for advanced users. The thesis examples showed that SystemVerilog-RN has potential in designing behavioural models for analog blocks. The benefit of using behavioural models can be seen in the simulation times and the flexibility they offer. Even for a single analog IP, the simulation time was considerably shorter than for the transistor level description.

7.1. SystemVerilog-RN Behavioural Model vs. Transistor Level Design

It is clear that a behavioural model created using SystemVerilog-RN cannot have the same amount of detail as a transistor level design. However, the benefit of using a behavioural model comes from the fact that the simulation time is reasonably shorter. The short simulation time increases the possibilities of utilizing the model for software debugging, if the model itself is validated and working. The fast-to-simulate behavioural model can be used to find software or design bugs (e.g. connectivity bugs) by running multiple simulations in a row. The difference in the simulation time of the LDO transistor level design and SystemVerilog-RN model can be seen in Table 5, in which the LDO SystemVerilog model and the transistor level description were simulated in the ADE environment. Spectre-simulator was used for the transistor level description and the AMS simulator for the SystemVerilog model. For high frequency systems that are heavy to simulate, the difference between simulation times could be even greater.

Table 5. The LDO simulation time comparison in ADE

Low-Dropout Regulator	Total simulation time:
Transistor Level Description	3min 59sec
SystemVerilog-RN	44sec

The possibilities which UDTs offer seem very interesting. However, this thesis was limited to only behavioural modelling without the use of UDT. It is possible and even likely that the usage of UDT will grow in the future. The possibility to use fast digital solvers and simulate electrical effects is a clear advantage. Compared to e.g. VHDL-AMS, which also can be used to create electrical behavioural models, SystemVerilog offers more flexibility. Also, the fact that SystemVerilog is widely used today in digital design and verification means that adapting SystemVerilog to analog design flow is an interesting idea and likely improves the whole mixed-signal chip design flow. In addition, there has been discussion about the upcoming SystemVerilog-AMS language standard, which aims to extend SystemVerilog analog and mixed-signal design and verification capabilities. [21]

7.2. Pros and Cons of SMG

SMG enables designers to create their own models without even knowing the actual model language. Especially with the possible increase of popularity to utilize SystemVerilog and UDT in analog behavioural modelling in the future, the analog

designers would have to learn another language in order to create efficient system describing models. Using SMG, the analog designers could skip the language learning part and just focus on the model creation. [18] Exploring the premade BBTs in the SMG GUI often gave ideas of what and how to model certain things. The model being in the form of a schematic gave better view of the whole model, what parts are missing and what parts should be improved. Even for a designer not familiar with behavioural modelling, getting involved with modelling was relatively easy.

For designers who are already familiar with SystemVerilog, SMG might quickly become too restrictive to use. One who knows SystemVerilog can always use a text editor basically free of charge. This is not true for SMG as it requires a CAD licence to operate. One big problem with model schematic was with logic. Complex combinational and sequential logic required inside the model, e.g. state machines, may be in some cases nearly impossible for a human mind to craft without errors. Currently, SMG does not seem to provide an easy option to solve the problem expect by handling the complex logic with handwritten SystemVerilog.

SMG generated code is totally compatible with ADE simulation environment. However, the generated code is not a ready-out-of-package solution for using it outside Cadence Design Systems environment as the generated code may include code that does not fully comply with the current SystemVerilog standard. This complicated the code re-usage outside ADE and there was no other way to affect this expect by modifying the SystemVerilog file or leaving the BBTs that generated such code out of the model schematic.

7.3. Suggested Workflow

It is important to determine the model abstraction level and the amount of control in the system. If the model requires a detailed description of the electrical behaviour, e.g. closed-loop simulations, it should be considered if SystemVerilog offers too high level abstraction, even with the use of UDT. SMG can be utilized to generate models very fast if the system is not too complex. If the system requires different states and a lot of timed control, it should be considered if SMG is usable and if the model creation should be given to a designer more familiar with SystemVerilog modelling.

The created model should be tested thoroughly. SystemVerilog verification capabilities e.g. random input generation, assertions and functional coverage should be utilized as much as possible to validate the model. Random-based approach is already widely used in digital design verification and it can speed up the verification process and offer a better functional coverage than directed approach. Figure 29 illustrates how the random-based approach reaches the 100% functional coverage of the DUT sooner than the directed approach. Initially it takes more time to develop a random-based testbench than a directed approach testbench but the DUT is exercised more effectively. [11] The examples of this thesis represented the directed approach method to test the functionality of the models.

The model benefits from being as interoperable as possible. If the model is created using SMG, it should be considered if the Cadence Design Systems provided possibilities outside the SystemVerilog standard are necessary. The generated code can be easily modified to comply with SystemVerilog standard by e.g. running a

script. One must keep in mind that the software and/or digital development may not take place in Cadence Design Systems supported environment.

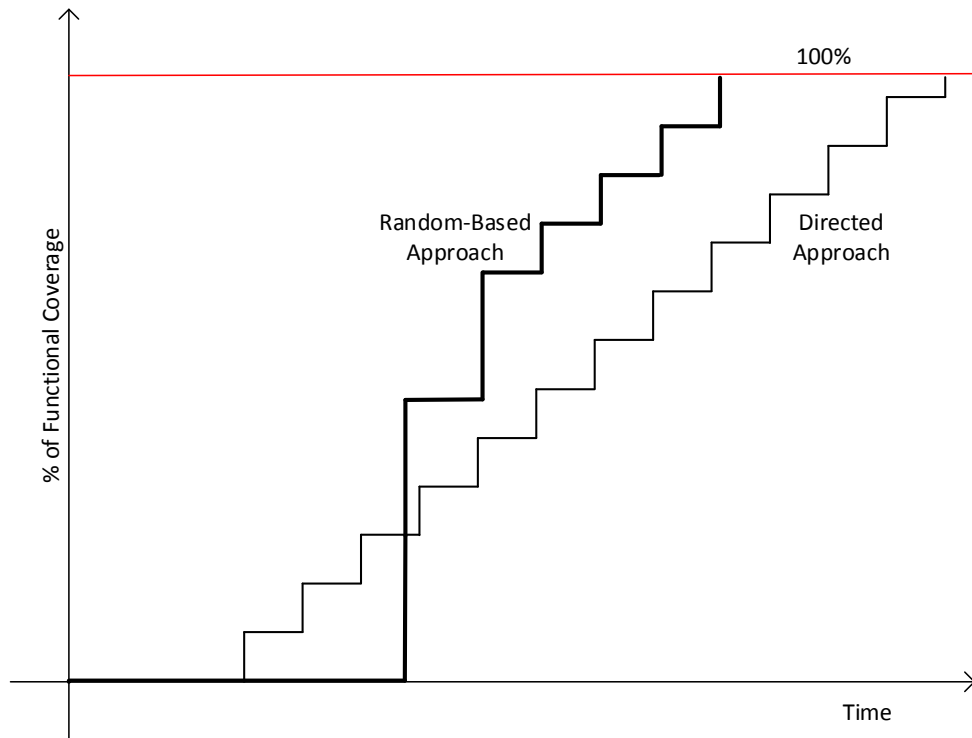


Figure 29. Functional coverage based on the testbench approach.

7.4. Future of SMG

SMG is a good new tool for model generation. However, in order to truly benefit from it, it should offer more to advanced users to make it more appealing. There should be a reason for the SystemVerilog and mixed-signal modelling knowing engineers to learn SMG.

SMG can be utilized to more complex designs by mixing-and-matching handmade SystemVerilog with the generated code. This option should be improved somehow, e.g. by not forcing the user to always create a BBT that performs certain action in the current model schematic and is not usable in any other design, which eventually may lead to the design library filling up with single-use BBTs.

8. SUMMARY

This thesis covered a concept of verifying mixed-signal circuits and why the mixed-signal verification is playing a big role in design of an integrated circuit. Mixed-signal verification using behavioural models was illustrated by introducing SystemVerilog-RN to model the system in an abstract manner that is often required for heavy simulations, e.g. LTE/5G simulations.

Schematic Model Generator tool by Cadence Design Systems was presented by creating a sample model to introduce the basic idea and the possibilities the tool offered. Furthermore, an LDO and a DCDC behavioural models were created using SystemVerilog-RN by utilizing SMG. SMG provided a fast and simple way to create a behavioural model for the LDO whereas for the more complex DCDC, the limitations of SMG were noted. In order to create a valid model for the DCDC, it was necessary to mix handwritten SystemVerilog with the SMG generated code.

Both models were validated by creating a testbench and an assertion module to capture the traffic between the behavioural model and the testbench to ensure correct functionality. According to assertions and visual inspection, both models succeeded to model the analog behaviour in the given abstraction. Entirely in the SMG environment created LDO model was likely maintainable in the future and the design effort was sufficient, whereas the DCDC model required a lot more design effort and the end result was not as satisfying as for the LDO model.

The compatibility with component level description and other simulation environments was examined. The LDO model and transistor level design were simulated in the same testbench and the results were used for evaluating the design performance. The DCDC model was modified in order to run the model in Mentor Graphics Questa environment. In the last part, the obtained results were concluded and the use of SystemVerilog-RN and SMG to create behavioural models was discussed.

9. REFERENCES

- [1] M. Fujita, Ghosh, Indradeep, M. Prasad. (2007) Verification Techniques for System-Level Design, 240 pages.
- [2] H.G. Bakeer, O. Shaheen, H. M. Eissa, M. Dessouky. (2007) Analog, Digital and Mixed-Signal Design Flows, 2007 2nd International Design and Test Workshop, pp. 247-252.
- [3] M. Baker. (2003) Demystifying Mixed Signal Test Methods.
- [4] https://www.cadence.com/content/cadence-www/global/en_US/home/solutions/mixed-signal-solutions/mixed-signal-implementation.html, read February 14th, 2018.
- [5] R. Munden. (2005) ASIC and FPGA Verification: A Guide to Component Modelling, 316 pages.
- [6] R. Sommer, I. Rugen-Herzig, E. Hennig, U. Gatti, P. Malcovati, F. Maloberti, K. Einwich, C. Clauss, P. Schwarz, G. Noessing. (2002) From system specification to layout: seamless top-down design methods for analog and mixed-signal applications, Design, Automation and Testing in Europe Conference and Exhibition, 2002, pp. 884-891.
- [7] J. Wang, L. Siek, R. Filippi, K.A. Ng (2007) A Top-Down Design Verification Based on Reuse Modular and Parametric Behavioral Modeling for Subranging Pipelined Analog-to-Digital Converter, 2007 International Symposium on Integrated Circuits, pp. 378-381
- [8] R. Wilton. (1997) Developments in and applications of mixed-signal HDL tools, IEE Colloquium on Mixed-Signal AHDL/VHDL Modelling and Synthesis (Ref. No: 1997/331), (November 1997).
- [9] A. Meyer. (2004) Principles of Functional Verification, 206 pages.
- [10] P. Elo. (2004) Mixed Signal IP reuse, University of Oulu, Department of Electrical and Information Engineering. Master's Thesis, 79 pages.
- [11] M. Pakkala. (2008) SystemVerilog-based Environment for RTL Module Verification, University of Oulu, Department of Electrical and Information Engineering. Master's Thesis, 79 p.
- [12] Accelera. (2009) Verilog-AMS Language Reference Manual – Analog & Mixed-Signal Extensions to Verilog-HDL, 392 pages.
- [13] <https://verilogams.com/refman/overview.html>, read January 24th, 2018.
- [14] P.J. Ashenden, G.D. Peterson, D.A. Teegarden. (2003) The System Designer's Guide to VHDL-AMS: Analog, Mixed-Signal, and Mixed-Technology Modelling.
- [15] S. Sutherland, S. Davidmann, P. Flake. (2006) SystemVerilog For Design: A Guide to Using SystemVerilog for Hardware Design and Modelling, Second Edition. Springer, 418 pages.
- [16] D.I. Long. (1997) Behavioural modelling of mixed signal circuits using PWL waveforms, IEEE Colloquium on Mixed-Signal AHDL/VHDL Modelling and Synthesis (Ref. No: 1997/331), (November 1997).
- [17] Cadence. (2014) Electrical Equivalent Mixed-Signal Library, Product Version 6.1.6, (August 2014).

- [18] Cadence. (2017) Cadence User Conference 2017, Full-chip Software and IC Co-verification Methodology Using SystemVerilog Real Number Models (May 2017)
- [19] O. Varkki. (1998) Integroidun analogiaelektroniikan HDL-A mallinnus sekamuotoelektroniikan suunnittelu ympäristössä, Oulun yliopisto, sähkötekniikan osasto, 57s.
- [20] R.A Shagger. (2007) Fundamentals of power electronics with MATLAB, 384 pages.
- [21] <http://www.eda.org/activities/working-groups/systemverilog-ams>, read March 27th, 2018.