EQUIVALENCE CHECKING OF DESIGNS MODELED IN SIMULINK IMPLEMENTED IN LOW LEVEL

by

Muharrem Orkun Sağlamdemir

B.S., Electrical & Electronics Engineering, Boğaziçi University, 2005

M.S., Electrical & Electronics Engineering, Boğaziçi University, 2007

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Electrical & Electronics Engineering

Boğaziçi University

2018

EQUIVALENCE CHECKING OF DESIGNS MODELED IN SIMULINK IMPLEMENTED IN LOW LEVEL

APPROVED BY:

Prof. Günhan Dündar (Thesis Supervisor)	.....
Assoc. Prof. Alper Şen (Thesis Co-supervisor)	.....
Assoc. Prof. Arda Deniz Yalçınkaya	.....
Assist. Prof. İsmail Faik Başkaya	.....
Assoc. Prof. H. Fatih Uğurdağ	.....
Assist. Prof. Mustafa Berke Yelten	.....

DATE OF APPROVAL: 05.01.2018

⋆

To My Family

who are my Pole Star...

who are my Beating Heart...

♡

ACKNOWLEDGEMENTS

My PhD education was really a long journey due to my full time job. Completing that journey would not be possible without the support of many people.

Firstly, I want to thank Prof. Günhan Dündar for supervising my PhD thesis, in addition to my undergraduate project and MSc thesis. His gentle approach has motivated me to reach the end of this research. He has been a role model for me and it is an honor for me to be a student of him. I want to express my gratitude to Prof. Alper Şen for co-supervising my PhD thesis. His approach to search for the better has motivated me a lot to make this research better. His recommendations on advancing my skills helped me on becoming a better researcher. I would like to thank Prof. H. Fatih Uğurdağ and Prof. İsmail Faik Başkaya for taking part in my thesis progress jury and thesis jury. I am thankful for their eagerness to help me in determining presentation dates. I want to thank Prof. Arda Deniz Yalçınkaya and Prof. Mustafa Berke Yelten for being part of my thesis jury.

I want to thank Gönenç Berkol and Engin Afacan for their support in the evolutionary computation and process variation analysis parts of the research, respectively.

I want to thank my brother Ömer Yetik for his support throughout my PhD study. The feeling of having him on my side is invaluable. I want to thank Şefika, Abbas and Hüseyin Yetik for their support.

Last but not least, I want to express my deepest gratitude to my beloved family, Tuğba, Belgin and Yahya Sağlamdemir. They were my brain in doing my research, they were my heart in keeping up my motivation. If their support were not on my side, this research would not be possible. I am grateful to my beloved sister Tuğba for being my road friend in this journey. I am thankful for having such wonderful parents and wonderful sister.

ABSTRACT

EQUIVALENCE CHECKING OF DESIGNS MODELED IN SIMULINK IMPLEMENTED IN LOW LEVEL

We propose methodologies for checking the equivalence of analog and digital Simulink models and their low level implementations. For digital methodology, we develop a flow utilizing commonly used tools used in digital integrated circuit design. Digital model can be designed by built-in Simulink libraries or user-defined functions. These models are converted to register-transfer level (RTL) implementations by usage of either Matlab's HDL Coder tool or Real Time Workshop followed by Catapult. Manual RTL implementation is compared to the converted RTL via Formality tool to decide equivalence. For the analog methodology, equivalence is decided by comparing predetermined performance parameters from simulations of model and the circuit level implementation. Simulations are done for different input parameters, which characterize the design. By means of this, a space of input parameters is found, where the model and the circuit level implementation are equivalent. For the analog methodology, we utilize Matlab's optimizer firstly, then we develop an evolutionary computation approach, which is a modified version of SPEA2 algorithm. We utilize Quasi Monte Carlo method to generate the samples, which are the input parameter set. This makes it possible to reach the result with less samples. Finally, we integrate process variation analysis to the equivalence checking methodology. To the best of our knowledge, the methodologies we propose for both digital and analog models are the first for checking equivalence for Simulink. Also the analog equivalence checking methodology with evolutionary computation can be applied to non-Simulink models. We validate digital methodology on designs distributed with Matlab package and Advanced Encryption Standard (AES). Proposed analog methodologies and process variation analysis are validated on inverter, operational amplifier and buck converter.

ÖZET

SİMULİNKTE MODELLENEN TASARIMLARIN ALT SEVİYEDEKİ GERÇEKLEMELERİYLE DENKLİKLERİNİN SINANMASI

Bu çalışmada analog ve sayısal Simulink modellerinin alt seviyedeki gerçeklemeleriyle denkliklerini sınamaya yarayan yöntemler önerilecektir. Sayısal yöntem, sayısal tümleşik devre tasarımında sıklıkla kullanılan araçlardan faydalanılarak geliştirilmiştir. Simulink'te sayısal modeller, mevcut kütüphane veya kullanıcı tanımlı fonksiyonlar aracılığıyla tasarlanır. Bu modeller yazmaç aktarım seviyesindeki gerçeklemeye Matlab'ın HDL Coder aracıyla veya Real Time Workshop sonrasında Catapult kullanılması ile dönüştürülür. Yazmaç aktarım seviyesinde yazılmış bir tasarımın önceki aşamada yine bu seviyeye dönüştürülmüş haldeki tasarımla denklik kıyaslaması Formality kullanılarak yapılır. Analog yöntemde ise, denkliğe model ve devre seviyesinde yapılan benzetimlerden elde edilen başarım değişkenlerinin kıyaslanması ile karar verilmektedir. Bu benzetimler tasarımı ifade eden farklı giriş değişkenleri için çalıştırılır. Bu sayede, model ve devre seviyesi tasarımın denk olduğu giriş değişkenleri uzayı elde edilir. Analog yöntemde, ilk aşamada Matlab'ın en iyileyicisi kullanılmıştır, bir sonraki aşamada SPEA2 yönteminden devşirilen bir evrimsel hesaplama yöntemi geliştirilmiştir. Son aşamada giriş değişkenleri kümesi olan örneklem oluşturulmasında Yarı Monte Carlo yönteminden faydalanılmıştır. Bu yöntem sayesinde daha az örnek kullanılarak sonuca ulaşılması sağlanmıştır. Son olarak, süreç değişimleri analizini yöntemimize ekledik. Bildiğimiz kadarıyla, Simulinkteki analog ve sayısal modeller için önerdiğimiz yöntemler alanda ilktir. Ayrıca, evrimsel hesaplama kullanan yöntemimiz Simulink dışında tasarlanmış modellere de uygulanabilir. Sayısal yöntemimiz İleri Şifreleme Standardı ve Matlab sürümüyle dağıtılan örnekler üzerinde; analog yöntemler ve süreç değişimleri analizi ise evirici, işlevsel yükselteç ve buck çevirici üzerinde doğrulanmıştır.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iv
ABSTRACT v
ÖZET vi
LIST OF FIGURES x
LIST OF TABLES xviii
LIST OF SYMBOLS xxi
LIST OF ACRONYMS/ABBREVIATIONS xxiv
1. INTRODUCTION 1
1.1. Outline 3
1.2. Contributions 4
2. RELATED WORK 7
2.1. Digital Approach 7
2.1.1. C and C++ 9
2.1.2. SystemC and TLM 10
2.1.3. Simulink 11
2.2. Analog Approach 17
2.2.1. Analog Model Checking 17
2.2.2. Analog Equivalence Checking 20
2.3. Process Variations 27
3. DIGITAL APPROACH 31
3.1. Introduction 31
3.2. Methodology 33
3.2.1. Simulink Background 33
3.2.2. Building of the Methodology 33
3.2.3. A Simple Simulink Example 35
3.3. Experiments 40
3.3.1. Configurable FIR Filter 40
3.3.2. Least Mean Squares (LMS) Filter 41
3.3.3. AES 44

3.4. Conclusion . . . 57
4. ANALOG APPROACH UTILIZING OPTIMIZER OF MATLAB . . . 60
4.1. Introduction . . . 60
4.2. Methodology . . . 61
4.2.1. Building of the Methodology . . . 61
4.2.2. Implementation of the Methodology . . . 62
4.3. Experiments . . . 63
4.3.1. Inverter . . . 64
4.3.2. Operational Amplifier (Opamp) . . . 70
4.3.3. Buck Converter . . . 74
4.4. Conclusion . . . 77
5. ANALOG APPROACH WITH EVOLUTIONARY COMPUTATION . . . 79
5.1. Introduction . . . 79
5.2. Methodology . . . 80
5.2.1. Building of the Methodology . . . 80
5.2.2. Proposed Approach . . . 82
5.3. Experiments . . . 84
5.3.1. Inverter . . . 85
5.3.2. Operational Amplifier (Opamp) . . . 88
5.3.3. Buck Converter . . . 92
5.4. Conclusion . . . 98
6. VARIABILITY ANALYSIS ON ANALOG APPROACH . . . 99
6.1. Introduction . . . 99
6.2. Methodology . . . 101
6.2.1. Inputs of Analysis . . . 102
6.2.2. Simulations . . . 104
6.2.3. Metric Creation for Selection of Individuals . . . 105
6.2.4. Selection of Successful Individuals . . . 106
6.2.5. Crossover and Mutation . . . 107
6.2.6. Equivalence Threshold Application and Finding Boundary . . . 108
6.3. Experiments . . . 108

6.3.1. Inverter 108
6.3.2. Operational Amplifier (Opamp) 115
6.3.3. Buck Converter 119
6.4. Discussion 124
6.5. Conclusion 125
7. CONCLUSIONS AND FUTURE WORK 127
REFERENCES 130

LIST OF FIGURES

Figure 2.1. Examples on sequential modifications in [1]. 8

Figure 2.2. Example for illustrating a compare point extraction in [2]. 11

Figure 2.3. A Simulink model. 12

Figure 2.4. Simulink based ASIC design and verification environment in [3]. . 13

Figure 2.5. Integrated ESL environment and SoC flow in [4]. 14

Figure 2.6. The architecture of the translation validation tool in [5]. 15

Figure 2.7. Overview of the structure of the C code generated for a Simulink model [6]. 16

Figure 2.8. Model checking flow [7]. 18

Figure 2.9. Linearization of PLL in the study [2]. 21

Figure 2.10. The structure of equivalence checking in [8–10]. 22

Figure 2.11. The methodology in [11]. 22

Figure 2.12. Complete state space-covering input stimuli generation and verification flow in [7]. 23

Figure 2.13. The equivalence checking procedure by means of complete state space-covering stimuli application [7]. 24

Figure 2.14. Signal mapping between the behavioral and electrical domains [12]. 24

Figure 2.15. Optimization based equivalence checking flow in [12]. 26

Figure 2.16. Effect of importance sampling on density function [13]. 28

Figure 3.1. Formal equivalence checking methodology. 35

Figure 3.2. Case study with built-in Simulink, Stateflow and S-function blocks. 35

Figure 3.3. Two state subsystem and its state diagram. 36

Figure 3.4. S-function builder I/O setting window. 36

Figure 3.5. S-function builder functionality setting window. 37

Figure 3.6. Simulink block for S-function. 37

Figure 3.7. Hierarchy for rtl.v file generated by Catapult. 38

Figure 3.8. Logic implemented by wrapper:core from Catapult. 38

Figure 3.9. Reference design window on Formality. 39

Figure 3.10. Implementation design window on Formality. 39

Figure 3.11. Formality result window. 40

Figure 3.12. The top level of configurable FIR filter Simulink model. 41

Figure 3.13. Stateflow part of the Simulink model for configurable FIR. 42

Figure 3.14. Datapath part of the Simulink model for configurable FIR. 42

Figure 3.15. Formality window for configurable FIR. 43

Figure 3.16. The system model of LMS filter. 43

Figure 3.17. LMSx10 blocks of LMS filter. 43

Figure 3.18. LMS tap blocks of LMS filter. 44

Figure 3.19. Formality window for LMS filter. 44

Figure 3.20. Pseudocode for AES. 46

Figure 3.21. Illustration for subbytes transformation. 46

Figure 3.22. Illustration for shift rows transformation. 47

Figure 3.23. Pseudocode for key expansion. 48

Figure 3.24. The hierarchy of the manual implementation for AES 128 in [14]. . 48

Figure 3.25. AES128_Key_Expand block diagram. 49

Figure 3.26. AES128_RCon block diagram. 49

Figure 3.27. S-function builder for mix column transformation. 50

Figure 3.28. S-function builder code for mix column transformation. 50

Figure 3.29. RTW S-function for mix columns. 51

Figure 3.30. Schematic for Catapult output for mix column transformation. . . 51

Figure 3.31. Formality window for mix column transformation. 52

Figure 3.32. AES cipher top showing interface. 53

Figure 3.33. The AES cipher top. 54

Figure 3.34. The AES key expand module of AES cipher top. 55

Figure 3.35. The AES 128 rcon module of AES key expand module. 56

Figure 3.36. A snapshot of HDL workflow advisor. 56

Figure 3.37. Modelsim simulation snapshot of manual RTL. 57

Figure 3.38. Simulink simulation of AES model. 58

Figure 4.1. Analog equivalence checking methodology. 62

Figure 4.2. Simulink inverter model. 66

Figure 4.3. Simulink inverter model block. 67

Figure 4.4. Simulink inverter model with input generator. 68

Figure 4.5. Results for the inverter. 70

Figure 4.6. Circuit diagram for two stage operational transconductance amplifier [15]. 71

Figure 4.7. Band limited opamp from SimScape library. 72

Figure 4.8. Operational amplifier configuration for frequency domain check. . 72

Figure 4.9. Circuit diagram for frequency domain check for operational amplifier. 73

Figure 4.10. Voltage sensor block. 73

Figure 4.11. Results for the opamp. 74

Figure 4.12. Circuit diagram for buck converter. 75

Figure 4.13. Simulink model of the buck converter. 76

Figure 4.14. Results for the buck converter. 77

Figure 5.1. Analog equivalence boundary search methodology. 81

Figure 5.2. Flow diagram of the proposed approach. 83

Figure 5.3. Boundary based pareto dominance. 84

Figure 5.4. Simulink inverter model with input generator. 85

Figure 5.5. Successful points for inverter Monte Carlo run. 86

Figure 5.6. Boundary for inverter with population size 99 and generation number 30 and successful Monte Carlo points. 87

Figure 5.7. Simulink and HSpice simulations for the points at the center, on the edge and on the outside for inverter. 89

Figure 5.8. Circuit diagram for two stage operational transconductance amplifier [15]. 90

Figure 5.9. Boundary for opamp with population size 99 and generation number 30. 91

Figure 5.10. Simulink and HSpice simulations for the points at the center, on the edge and on the outside for opamp. 92

Figure 5.11. Simulink model of the buck converter. 93

Figure 5.12. Circuit topology for buck converter. 94

Figure 5.13. Boundary in 2d for buck with population size 99 and generation number 30 and r_l=400. 95

Figure 5.14. Boundary for buck with population size 99 and generation number 30. 96

Figure 5.15. Simulink and HSpice simulations for the points at the center, on the edge and on the outside for buck converter. 97

Figure 6.1. Flow diagram for process variations-effect analysis for analog behavioral equivalence boundary computation. 103

Figure 6.2. Illustration for boundary number assignment algorithm. 106

Figure 6.3. Modeling of chromosomes and generation of offsprings. 107

Figure 6.4. Simulink inverter model with input generator. 109

Figure 6.5. Equivalence boundaries for inverter before process variations-effect analysis and for different equivalence values with population size 99 and generation number 20. 111

Figure 6.6. The archive for 5-generation run filtered for yield greater than or equal to 70%. 112

Figure 6.7. Boundary for inverter with population size 99 and generation number 20. 113

Figure 6.8. 70% equivalence boundary for inverter with population size 99 and generation Number 5. 113

Figure 6.9. Circuit diagram for two stage operational transconductance amplifier [15]. 115

Figure 6.10. Operational amplifier configuration for frequency domain check. . 116

Figure 6.11. Boundary for opamp with population size 99 and generation number 30. 118

Figure 6.12. 70% equivalence boundary for opamp with population size 99 and generation number 5. 118

Figure 6.13. Simulink model of the buck converter. 120

Figure 6.14. Circuit diagram for buck converter. 120

Figure 6.15. Boundary for buck converter with population size 99 and generation number 30. 122

Figure 6.16. 70% equivalence boundary for buck converter with population size 99 and generation number 5. 122

LIST OF TABLES

Table 2.1. Declarations of parameters and functions in the optimization problem definition in [12]. 25

Table 3.1. C functions belonging to S-functions. 38

Table 3.2. Some metrics for the experiments. 58

Table 4.1. Algorithm for extraction of performance parameter error percentages for current input parameter values. 64

Table 4.2. Algorithm for finding the optimal input parameter set. 64

Table 4.3. Algorithm for finding the upper/lower threshold value for each input parameter. 65

Table 4.4. Matrices and vectors in the definition of $fminimax()$ function. . . 69

Table 5.1. Parameters for inverter example. 86

Table 5.2. Locations of points at the center, on the boundary and on the outside for inverter. 88

Table 5.3. Parameters for opamp example. 91

Table 5.4. Locations of points at the center, on the boundary and on the outside for opamp. 92

Table 5.5. Parameters for buck converter example. 94

Table 5.6. Locations of points at the center, on the boundary and on the outside for buck converter. 96

Table 6.1. Parameters for inverter example. 109

Table 6.2. Volume encapsulated in the equivalence boundary for inverter. . . 112

Table 6.3. Locations of points at the center, on the boundary and on the outside for inverter. 114

Table 6.4. Classification of 49 points on the 70% equivalence boundary according to percentage equivalence value for inverter. 114

Table 6.5. Parameters for opamp example. 116

Table 6.6. Volume encapsulated in the equivalence boundary for opamp. . . . 117

Table 6.7. Locations of points at the center, on the boundary and on the outside for opamp. 117

Table 6.8. Classification of 45 points on the 70% equivalence boundary according to percentage equivalence value for opamp. 119

Table 6.9. Parameters for buck converter example. 121

Table 6.10. Volume encapsulated in the equivalence boundary for buck. 123

Table 6.11. Locations of points at the center, on the boundary and on the outside for buck converter. 123

Table 6.12. Classification of 26 points on the 70% equivalence boundary according to percentage equivalence value for buck converter. 124

LIST OF SYMBOLS

A	Inequality matrix
A_β	Process variation parameter for current factor
A_{eq}	Equality matrix
A_{VT}	Process variation parameter for threshold voltage
b	Inequality vector
b_{eq}	Equality vector
c	Inequality constraint vector
c_{eq}	Equality constraint vector
C_{int}	Internal Capacitance
C_{load}	Load capacitance
Clk	Clock
Cut_off	Cut-off frequency
D_n	Star discrepancy
f_{B-E}	Function for mapping behavioral input to electrical domain
F	Objective function vector
$Fall_T$	Fall time
g_{E-B}	Function for mapping electrical output to behavioral domain
GF	Galois field
h_{err}	Error function for behavioral domain
k_n	Transconductance parameter for NMOS transistor
k_p	Transconductance parameter for PMOS transistor
l	Length of the transistor
lb	Lower bound vector
I_B	Input in behavioral domain
I_E	Input in electrical domain
L_n	Length of N type Metal Oxide Transistor
L_p	Length of P type Metal Oxide Transistor
Max_V	Maximum output voltage

Min_V	Minimum output voltage
N_b	Block size
N_r	Number of rounds in Advanced Encryption Standard
O_B	Output in behavioral domain
O_E	Output in electrical domain
$Prop_hl$	Propagation delay high to low
$Prop_lh$	Propagation delay low to high
$Prop_M$	Propagation delay mean
pw	Duty ratio of switching pulse waveform
Q_{SP}	Function mapping electrical domain inputs to outputs
r_{cap}	Capacitor series resistance
r_{ind}	Inductor series resistance
r_l	Load resistance
r_{sw}	Switch resistance
rst	Reset
R_{in}	Input resistance
R_n	On resistance for NMOS Transistor
R_{out}	Output resistance
R_p	On resistance for PMOS Transistor
$Ripple_V$	Ripple voltage
$Rise_T$	Rise time
$Slew_hl$	Slew rate high to low
$Slew_lh$	Slew rate low to high
t_{fall}	Fall time
t_{ox}	Oxide thickness of the transistor
t_{rise}	Rise time
ub	Upper bound vector
v_{th}	Threshold voltage of the transistor
V_0	Initial output voltage
V_{dd}	Supply voltage
V_{dot}	Maximum slew rate

V_{HK}	Total variation in Koksma-Hlawka theorem
V_m	Switching voltage
V_T	Threshold voltage
w	WIdth of the transistor
W_n	Width of N type Metal Oxide Transistor
W_p	Width of P type Metal Oxide Transistor
$(W/L)_n$	Width to Length Ratio for N type Metal Oxide Transistor
$(W/L)_p$	Width to Length Ratio for P type Metal Oxide Transistor
x	Input parameter vector
β	Current factor

LIST OF ACRONYMS/ABBREVIATIONS

2D	Two Dimensional
AEC	Analog Equivalence Checking
AES	Advanced Encryption Standard
AMS	Analog and Mixed Signal
ANSI	American National Standards Institute
ASIC	Application Specific Integrated Circuit
ASL	Analog Specification Language
ATPG	Automatic Test Pattern Generation
BDD	Binary Decision Diagram
BMC	Bounded Model Checking
CAD	Computer Aided Design
CEC	Combinational Equivalency Checking
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processing Unit
CTL	Computation Tree Logic
DAE	Differential Algebraic Equation
DATS	Discrete Analog Transition Structure
DC	Direct Current
DFG	Data Flow Graph
EC	Equivalence Checking
ESL	Electronic System Level
FIR	Finite Impulse Response
FSM	Finite State Machine
FPGA	Field Programmable Gate Array
GB	Giga Byte
GF	Galois Field
GUI	Graphical User Interface
HDL	Hardware Description Language

HLM	High Level Model
HLS	High Level Synthesis
I/O	Input Output
I/O-EFA	Input Output Extended Finite Automata
IEEE	Institute of Electrical and Electronics Engineers
IBM	International Business Machines
IC	Integrated Circuit
IP	Intellectual Property
ISCAS	International Symposium on Circuits and Systems
KLE	Karhunen-Loeve Expansion
LDS	Low Discrepancy Sequences
LHPN	Labeled Hybrid Petri Nets
LMS	Least Mean Squares
LTL	Linear Temporal Logic
MBD	Model Based Design
MC	Monte Carlo
MNA	Modified Node Analysis
NAND	Not And
NIST	The National Institute of Standards and Technology
NMOS	N Type Metal Oxide Semiconductor
OPAMP	Operational Amplifier
PEP	Potentially Equivalent Pairs
PLL	Phase Locked Loop
PMOS	P Type Metal Oxide Semiconductor
PSL	Property Specification Language
QMC	Quasi Monte Carlo
RAM	Random Access Memory
RTL	Register Transfer Level
RTW	Real Time Workshop
RV	Random Variable
S-Function	System Function

SAT	Satisfiability Problem of Boolean Formulas
SDE	Statistical Difference Equation
SEC	Sequential Equivalence Checking
SERE	Sequential Extended Regular Expressions
SLM	System Level Model
SMT	Satisfiability Modulo Theories
SoC	System-on-a-Chip
SPEA2	Strength Pareto Evolutionary Algorithm 2
SPICE	Simulation Program with Integrated Circuit Emphasis
SQP	Sequential Quadratic Programming
SRAM	Static Random Access Memory
SSTA	Statistical Static Timing Analysis
STA	Static Timing Analysis
STL	Signal Temporal Logic
TLC	Target Language Compiler
TLM	Transaction Level Modeling
US	United States
UVM	Universal Verification Methodology
VHDL	Very High Speed Integrated Circuit Hardware Description Language
XOR	Exclusive Or

1. INTRODUCTION

In IC Design industry, it is a widely known fact that verification activities consume most of the time in a project [16]. Abstraction is an important paradigm to reduce verification costs. High level design methodologies such as model based design (MBD) starts with an abstract model of the design.There is a trend to make whole IC design process model based [17]. In early days of IC Design, specifications were text based and were not executable. However, driven by the increase in complexity of designs, time-to-market pressure and design-team integration, executable model description became the initial starting point of an IC Design project. Those driving factors are also pushing designers to higher and higher levels of abstractions, since verification is faster and architectural optimizations are enabled at higher levels of abstraction [18].

Also, there is another driving factor facilitating development of High Level Models (HLM), namely High Level Synthesis (HLS). HLS is the process of synthesizing the RTL implementation from a high-level system level model. By means of high-level synthesis, RTL code is generated automatically, which makes the RTL implementation free of human errors. Also, since RTL implementation time is just the run-time of the synthesizer, design time decreases. More than 30 of the top semiconductor and system companies in the world have adopted HLS by 2010 [19]. System-on-a-Chip (SoC) tape outs, which contain HLS-developed IPs have exceeded 50 in 2009 [19]. Mainly blocks including intensive mathematical operations are synthesized from their high level model rather than implementing them in RTL.

From the above mentioned advantages and new arriving concept of HLS, it is clear that much more time will be spent on High Level Modeling in the IC Design Industry. However, equivalence of synthesized RTL should be compared to the corresponding high level model. Equivalence check is needed for other design projects where the System Level Model (SLM) exists, even if the low level design is obtained manually. Thus, equivalence of SLM code to RTL code needs to be checked. Also, in the course of a project, multiple high level models may exist for different usage scenarios and these

models have to be consistent with each other [20].

In today's IC Design Industry, RTL implementations are not always developed in house. From the perspective of design and reuse, many companies acquire Intellectual Properties (IP) for their projects. In this case, there needs to be a checking mechanism to assure functional equivalency between SLM and the acquired IP, as well.

Combinational Equivalence Checking (CEC) has been used to check for equivalency between RTL and Gate-Level Netlist for a long time [21]. In this case, there is a mapping between the elements of two designs, such as inputs, outputs, flip-flops, etc. However, it is difficult to obtain a strict mapping between SLM and RTL designs. Hence, the concept of equivalency checking is an open question in academia and industry for the case of checking SLM code and RTL implementation. There exist studies on equivalence checking of SLMs written in C and RTL designs [22–26]. In addition to C, also there are several studies on the languages like C++ and SystemC [2, 27, 28].

A popular high level modeling language is Simulink. Simulink is a powerful modeling language with its extensive model library. In addition to its library, it makes building user-defined models easy. Furthermore, Simulink provides dependable toolboxes like Optimization toolbox, which may be utilized during exploration of design space in modeling process. However, equivalence checking (EC) for Simulink has not been studied in the literature. Because of this, working on EC for Simulink is valuable and it will be a novel contribution to literature. In this thesis, we develop EC solutions for high level designs developed using Simulink and RTL digital designs.

Simulink is also a widely used environment to model Analog and Mixed Signal (AMS) systems at a high level. In that respect, checking the equivalence of Simulink models and their low circuit level counterparts is another area to be explored. To the best of our knowledge, there is no study in the literature on the equivalence checking of Simulink models of AMS systems and their low circuit level counterparts. Furthermore, there exists only a few works on EC between high level models other than Simulink such as Verilog-AMS and circuit level counterparts [9, 12, 29, 30]. In this thesis, we will

also develop EC solutions for high level analog designs developed using Simulink and circuit level designs.

There are various approaches to analog equivalence checking. One of these approaches generates a linear transfer function for both model and implementation [29, 30]. Another approach generates stimuli for both model and the implementation which cover the state space completely [7]. These approaches suffer from state explosion problem as run time gets too large when a small increase is seen in design complexity. There is also a semi-formal Analog Equivalence Checking (AEC) approach that works using a constrained input set [12]. They use Verilog-AMS as high level models.

In this thesis, we develop AEC techniques for high level digital and analog models and their low level implementations. For analog techniques, we propose two techniques: one with $fminimax$ function from optimization toolbox of Matlab [31], the other with an evolutionary computation approach, which is a modified version of SPEA2 [32]. Finally, we integrate process variation analysis to our AEC approach with evolutionary computation.

1.1. Outline

In Chapter 2, we present our literature survey on digital equivalence checking, analog equivalence checking and process variations-effect analysis, respectively. In Chapter 3, we propose a methodology for a checking the formal equivalence between digital Simulink models and their HDL counterparts. This chapter presents also the validation of our methodology on standardized, publicly distributed complex designs. In Chapter 4, we present an analog equivalence checking methodology for Simulink models and their circuit level implementations (SPICE). This methodology utilizes $fminimax$ function from Matlab optimization toolbox and provides lower and upper bounds for parameters characterizing the inputs and outputs of design. In Chapter 5, we propose a simulation-based analog equivalence boundary search methodology for high level Simulink models and their low level HSpice counterparts. As an enhancement to our study in Chapter 4, we develop our tool-independent optimizer utilizing

evolutionary computation in Chapter 5. In Chapter 6, we present our methodology, which analyzes the effects of process variations on analog equivalence checking. The methodology in this chapter is extended from the methodology in Chapter 5 and it generates process variations-aware equivalence boundary. Methodologies presented in Chapters 4, 5, and 6 are validated by inverter, operational amplifier and buck converter examples.

1.2. Contributions

The contributions of this thesis are listed below in a categorized form according to the corresponding chapter:

Chapter 3: Digital Approach, which is first published in [33]

- We propose a digital formal equivalence checking methodology for digital Simulink models and their HDL counterparts.
- We validate our methodology utilizing standardized and publicly distributed complex designs.
- We design the first Advanced Encryption Standard (AES) Simulink model, which only includes Simulink library elements.

Chapter 4: Analog Approach Utilizing Optimizer of Matlab, which is first published in [34]

- We present the first semi-formal analog equivalence checking methodology for Simulink models and their circuit level implementations using $fminimax$ optimization function from Matlab.
- We provide the upper and lower thresholds for input parameters for a design, which characterize the inputs and the outputs of the design, where the model and the implementation are equivalent.
- We validate our methodology on inverter, operational amplifier and buck converter examples.

Chapter 5: Analog Approach with Evolutionary Computation, which is first published in [35]

- We improve our AEC methodology by building an evolutionary computation based optimizer, which is a modified version of SPEA2 [32]. To the best of our knowledge, this is the first time evolutionary computation is applied to analog equivalence checking problem.
- We propose a simulation-based analog equivalence boundary search methodology for high level Simulink models and their low level HSpice counterparts.
- Our approach is capable of finding the equivalence boundary in the space defined by the parameters.
- Our approach reduces the necessary number of simulations when compared with Monte Carlo method.
- We validate our methodology on inverter, operational amplifier and buck converter examples.

Chapter 6: Variability Analysis on Analog Approach, which is first published in [36]

- We finally improve our AEC methodology by integrating process variation analysis to our approach.
- We investigate the change in equivalence boundary by filtering for different target equivalence threshold values.
- We utilize evolutionary computation in our process variations-aware AEC analysis.
- We apply Quasi Monte Carlo (QMC) method in process variation analysis, which makes it possible to estimate the yield with fewer samples compared to Monte Carlo (MC).
- We build a multi-objective optimization problem, where in addition to equivalence value, we consider closeness to the equivalence boundary by defining a parameter for each point of concern.

- We demonstrate the validity of our approach on three designs, an inverter, an operational amplifier and a buck converter.

2. RELATED WORK

In this chapter, an overview of recent studies on those areas is presented before proposing our approaches on equivalence checking and process variations-effect analysis. Section 2.1 discusses the recent research on digital equivalence checking with subsections on different languages. Section 2.2 presents various studies on analog model verification with subsections on model checking and equivalence checking. Finally, Section 2.3 discusses researches on process variations-effect analysis.

2.1. Digital Approach

Hardware comparison can be dated back to 1965 in literature with the name of "Boolean comparison programs" [37]. Although there were researches after that study, the concept of comparison between a high level representation of hardware ("flowchart" in [38]) and its low level counterpart was introduced in [38]. Boolean comparison programs have evolved to the concept of Combinational Equivalence Checking (CEC) in the following years and it is widely utilized in IC design projects to check the equivalence of two designs. There are widely used tools like Formality [39] to match the registers in two designs and to compare the combinational logic circuits between the matched registers to decide combinational equivalence. Although the solution to CEC problem is solid now, for sequential circuits, the equivalence problem is a hot topic for research.

In 1992, Pixley [40] provided a comprehensive theoretical framework for hardware equivalence and introduced the concept of alignability for bringing two state machines into corresponding states, which is a good notion of sequential equivalence. Sequential Equivalence Checking (SEC) is the procedure of providing a formal proof of functional equivalence of designs that may have different implementations sequentially. Sequential differences can be exemplified as follows: mismatches in throughputs and latencies (allowing for re-pipelined design checking), retimed pipelines (leading to broken mapping between registers, and ineffective combinational equivalence checkers), and even

resource allocation and scheduling differences. These sequential modifications can be applied to a design in order to meet constraints like area, timing etc. Figure 2.1 presents three examples on those modifications:

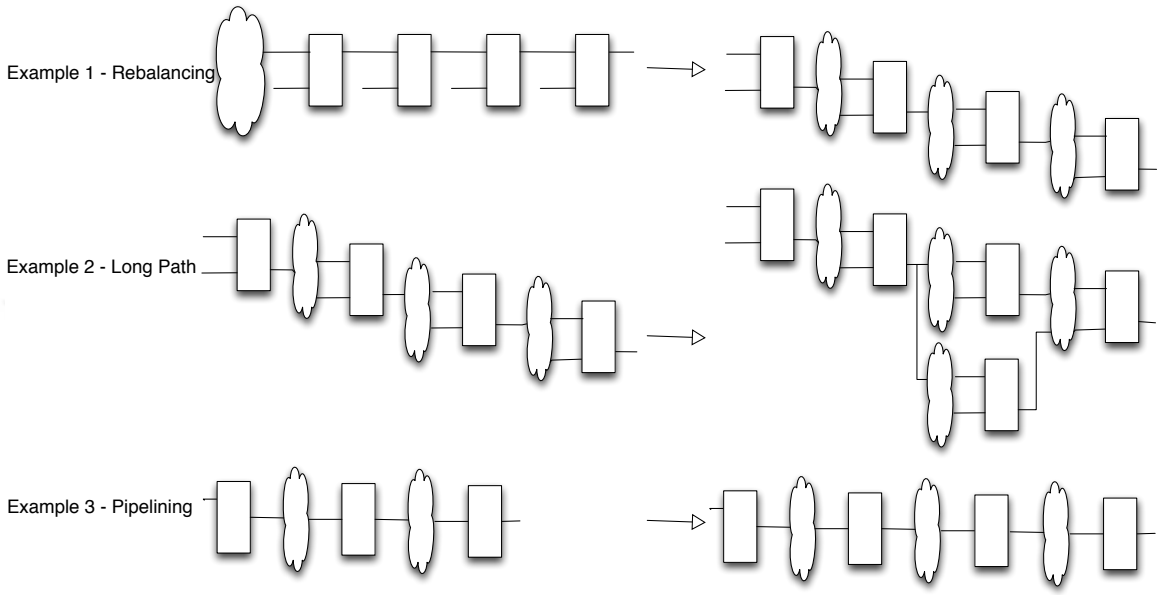

Figure 2.1. Examples on sequential modifications in [1].

Examples 1 and 2 in Figure 2.1 illustrate moving combinational logic across flip-flop boundaries or re-timing to meet set-up time constraints. Example 3 in Figure 2.1 illustrates addition of pipeline stages to a calculation or re-pipelining, which trade off latency to maintain throughput while meeting set-up time constraints. These examples give some insight about how functionally same designs may differ in implementation.

In case of comparing high-level models with their RTL implementation, we come across different implementations that have functionally same behavior. SEC aims at proving the functional equivalence of designs implemented at different abstraction levels. In the course of proof, both implementations are transformed to the same representation. In the literature, some studies refer to this as the verification condition [5]. Then, those two conditions are given to solvers like SAT, BDD etc. to prove/disprove for equivalence [41, 42].

The literature on SEC can be categorized according to the programming language used in System Level Modeling. Those languages have broad spectrum of C,

C++, SystemVerilog, SpecC, SystemC, Matlab, Simulink. C has been the most prevalent modeling language for years. However, the trend is moving to object-oriented languages, as it is for software technology.

Another parameter for categorization of SEC literature is the cycle-accuracy of the SLM. Most of the current research is done on cycle-accurate models. However, there is an intent to devise some mechanism to convert non-cycle accurate models to cycle-accurate ones and then apply the existing methodologies to check the transformed cycle-accurate model for equivalency with respect to its RTL implementation. There are also studies, which define new equivalency definitions like event-based equivalence. Furthermore, there are some studies that aim at abstracting the existing IP by some mechanism in a proof by construction mechanism. In the following sections, studies will be presented in an order of complexity of the programming language used. Simulink model equivalence checking will be discussed separately in Section 2.1.3, since it is the subject of thesis.

2.1.1. C and C++

As it was stated previously, C is a prevalent language for modeling hardware. In [22], Kroening and Clarke used a Satisfiability (SAT) based Bounded Model Checking (BMC) to transform the equivalence problem to decision problem of bit-vector logic type. In BMC, boolean formulas are obtained by processing the transition relations of both the system and the specification. Then, satisfiability check is applied to those boolean formulas utilizing an efficient SAT engine. SAT engine generates a counter-example, in case the formula turns out to be satisfiable. If the formula is not satisfiable, analysis of the system and the specification is continued. The analysis is ended when the completeness limit is exceeded by the probable counter-example length. The authors extended the existing methodology to cover the ANSI-C language features set completely. However, function computed by the circuit and the program is compared. Reactive systems are not supported. In [23], this methodology is extended to be able to process reactive programs and circuits. Even intricate properties like nested loops and pointers are supported in this study. Also, non-cycle-accurate models are supported

in addition to cycle-accurate models. By means of this work, behavior of complex, reactive hardware designs can be specified by a widespread sequential programming language like ANSI-C. Details on implementation can be reached from [24].

Another study by Ding *et al.* [25], claims that reasonable cuts reduce time consumption in combinational verification having in the mind that converting large scale circuits to several small ones, which will be verified separately, will ease the problem. They embed the technique into sequential equivalence checking as an improvement of Van Eijk's algorithm [26], main idea of which is to find the nodes which have the same value (or negated values) in all reachable states of the two circuits.

Ghosh *et al.* [27], announced that they had built a framework which is capable of taking real life C++ programs and applying equivalence check on them. In that work, they synthesize finite state machines from the designs to be compared and after that they apply techniques other than formal verification to locate possible intermediate equivalences between them. Finally, formal techniques are used in a controlled manner to prove or disprove the equivalences.

2.1.2. SystemC and TLM

On the SystemC side of the EC problem, we come across two studies [2, 28]. In [28], Tu *et al.* introduce an approach to ease the equivalence checking problem by utilizing compare points for decomposition, whereas in literature mapping of latches are used for compare points. They divide the design considering compare points and describe the operation of design at those points via symbolic expressions. Finally for checking equivalence, they check the equivalence of expressions by utilizing a SAT solver. By making use of compare points, the authors can split the problem of equivalence-checking into less difficult problems that can be solved by lower level engines. Observables are selected visually by means of inspecting the block diagram of the system. In a later version of the same study, which is [2], observables are obtained by moving on the state transition graphs for the two systems. This is better in the sense that visual inspection of the observables may result in missing of some parame-

ters which are valuable to observe. In reaching a compare point, the state machines of the two designs are initialized from the same state, and the state change is followed in every cycle until they come across another sequential compare point. In Figure 2.2, B is the first compare point obtained.

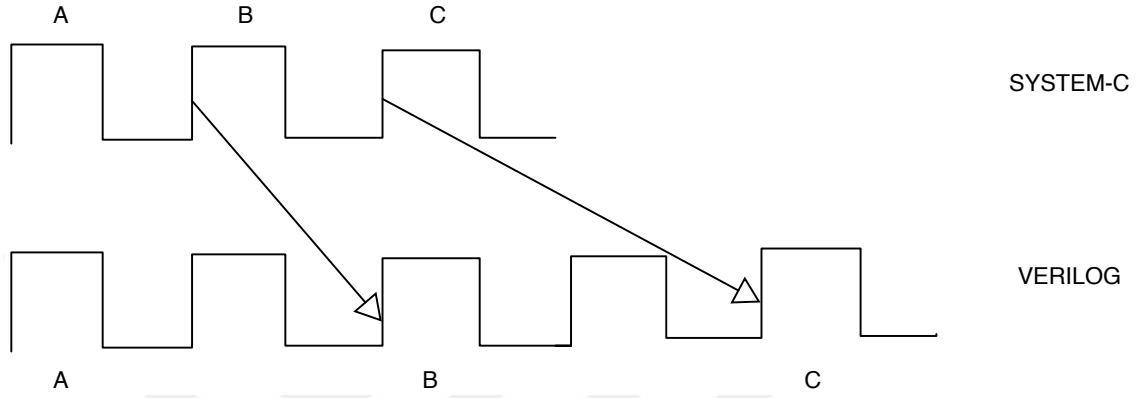

Figure 2.2. Example for illustrating a compare point extraction in [2].

Another high level language, where equivalence checking is studied is TLM. In [43], a formal definition of equivalence based on events is presented. Study shows how such a definition can be used for proving the equivalence in the RTL vs. TLM context, without requiring timing or structural similarities between the modules to be compared. The models are considered as black-boxes and only the I/O behaviors are matched. Thus, no similarities on the internal structures of the models (e.g., common registers, or common finite state machine templates) are required to check the equivalence. The functional equivalence can be checked even if there is no timing correlation between the models, since the definition proposed in this paper relies on the concepts of sequence of events disregarding the real instant when such events occur. Also paper proves that modules abstracted according to RTL-to-TLM abstraction methodology are correct by construction with respect to event-based equivalence. By this way, IPs which are acquired from third parties can be integrated easily to the system level models in house.

2.1.3. Simulink

The Simulink language is a graphical modeling language comprising block diagrams and an extensive set of block libraries [44]. A Simulink model example is shown

in Figure 2.3.

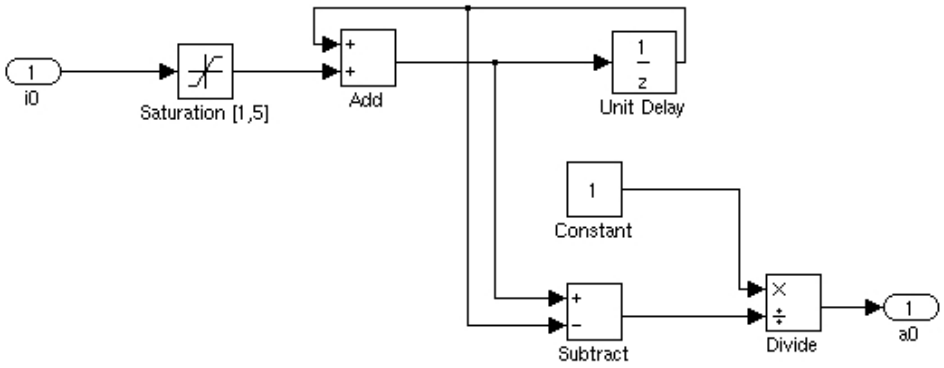

Figure 2.3. A Simulink model.

Matlab/Simulink is an environment, where a single design description can be used to built a unified algorithm-architecture-circuit co-design environment for dedicated signal processing hardware [44]. All phases of a chip design flow, which are FPGA emulation, ASIC design, verification, and chip testing can be integrated around this environment, as it is done in study [3]. This unified description enables system designer with a visibility through several layers of design hierarchy down to circuit level to select the optimal architecture. The tool flow propagates circuit-level performance and power estimations up to rapidly evaluate architecture-level tradeoffs. That unified construction in [3] is illustrated in Figure 2.4.

Simulink may also be used to build an integrated Electronic System Level (ESL) environment for model-based SoC design. An example of that usage is illustrated in Figure 2.5 [4]. In that environment, algorithm specification modeling is performed in Simulink, and software and hardware synthesis of SoC is done by the combination of several commercialized ESL tools. [45] also proposes a cosimulation methodology that results in an efficient SoC design as well as fast verification by integrating HDL, System-C, and algorithm-level abstraction using the design tools Active-HDL and Matlab Simulink.

Studies referred in this section previously present usage of Simulink as a powerful modeling language. In addition to that, Simulink has RTW Embedded Coder tool,

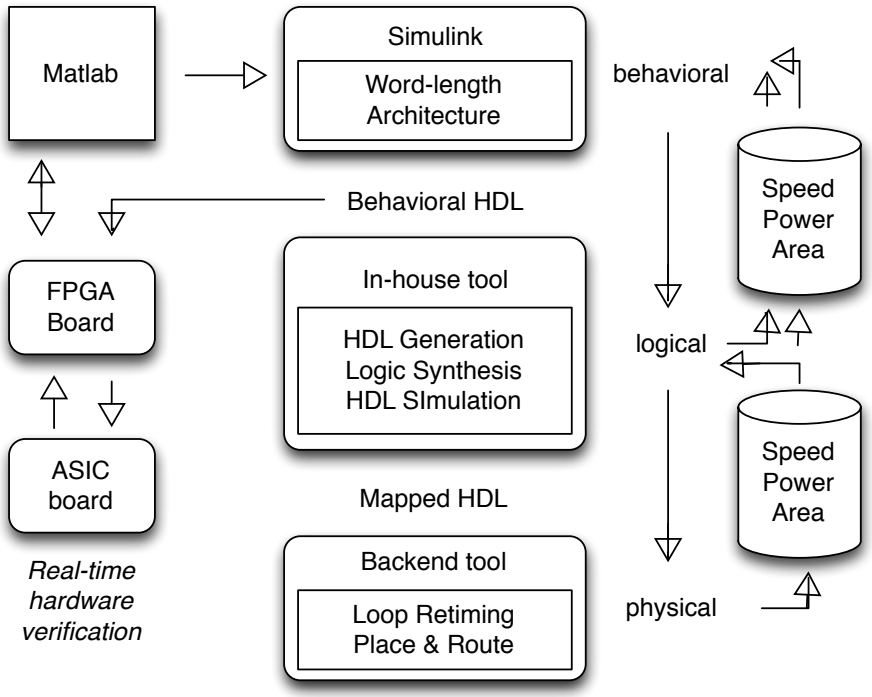

Figure 2.4. Simulink based ASIC design and verification environment in [3].

which has the capability of C code synthesis from Simulink code. As it may be remembered from the discussions on sequential equivalence checking of C models, there are procedures, which effectively does checking even in the presence of intricate constructs like pointers, nested loops etc. Considering that, the hardest part to be solved in checking the equivalence of Simulink model and RTL implementation is checking the equivalency of Simulink model code and C code generated from that code.

In [5], a tool for validating the translations of Simulink models to optimized C codes via Real-Time Workshop generator is proposed. For validation, the program for the source and the target, input/output mapping for those programs and state variables are provided to tool. Transition relations are generated for both the source and the target. Considering the syntax of the languages for the target and the source, verification condition is generated by the tool. Verification condition is the representation of the correspondence between the target and the source, in addition to the target and the source codes themselves. Then, a decision procedure named Yices is applied. The procedure used in [5] is illustrated in Figure 2.6.

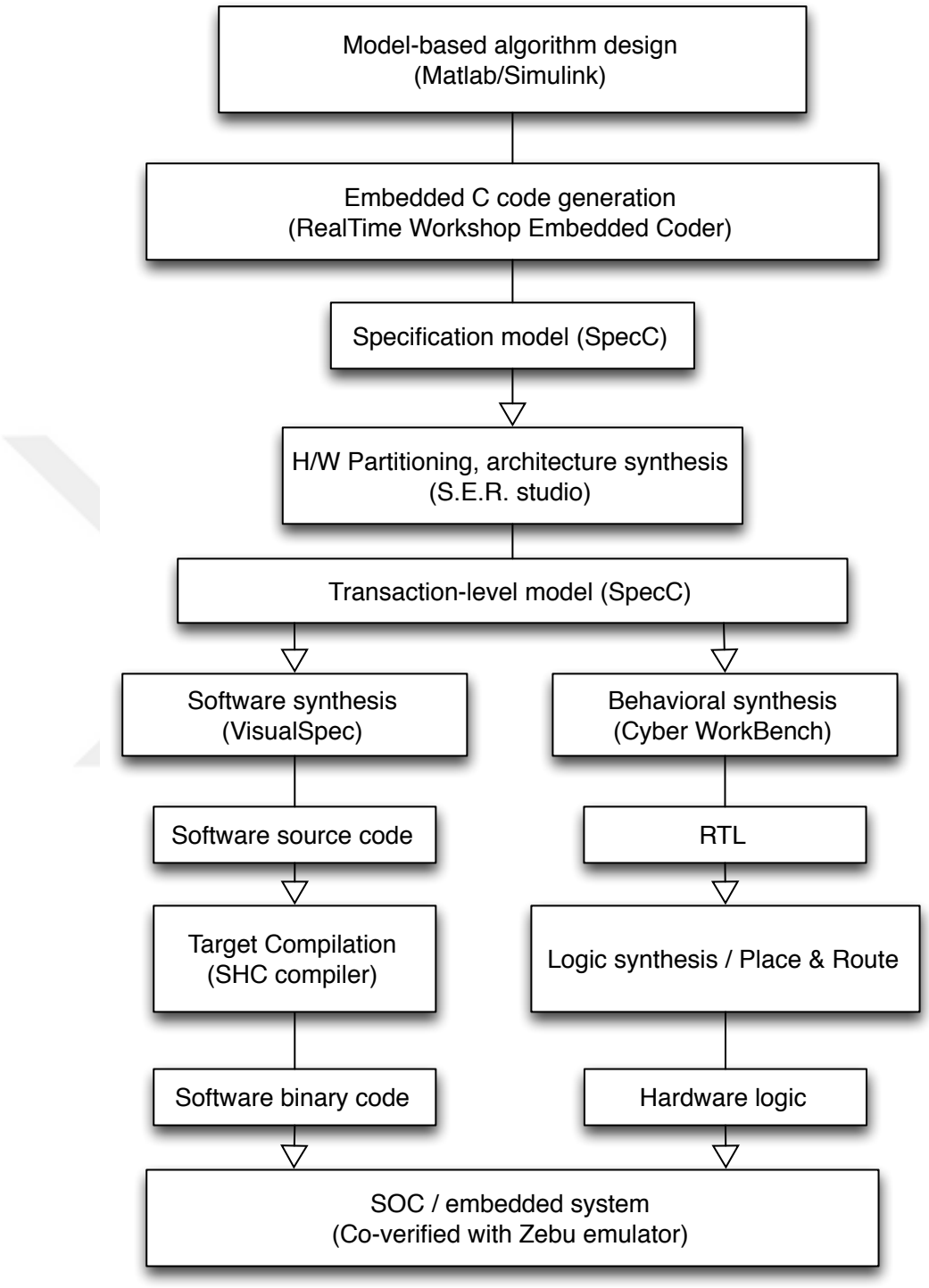

Figure 2.5. Integrated ESL environment and SoC flow in [4].

As it can be observed from the architecture of the tool, creation of the verification condition for target and source programs is at the heart of the translation validation problem. Creation of the verification condition is mainly dependent upon the categorization of the block types in Simulink [5]. Simulink blocks are classified as virtual or

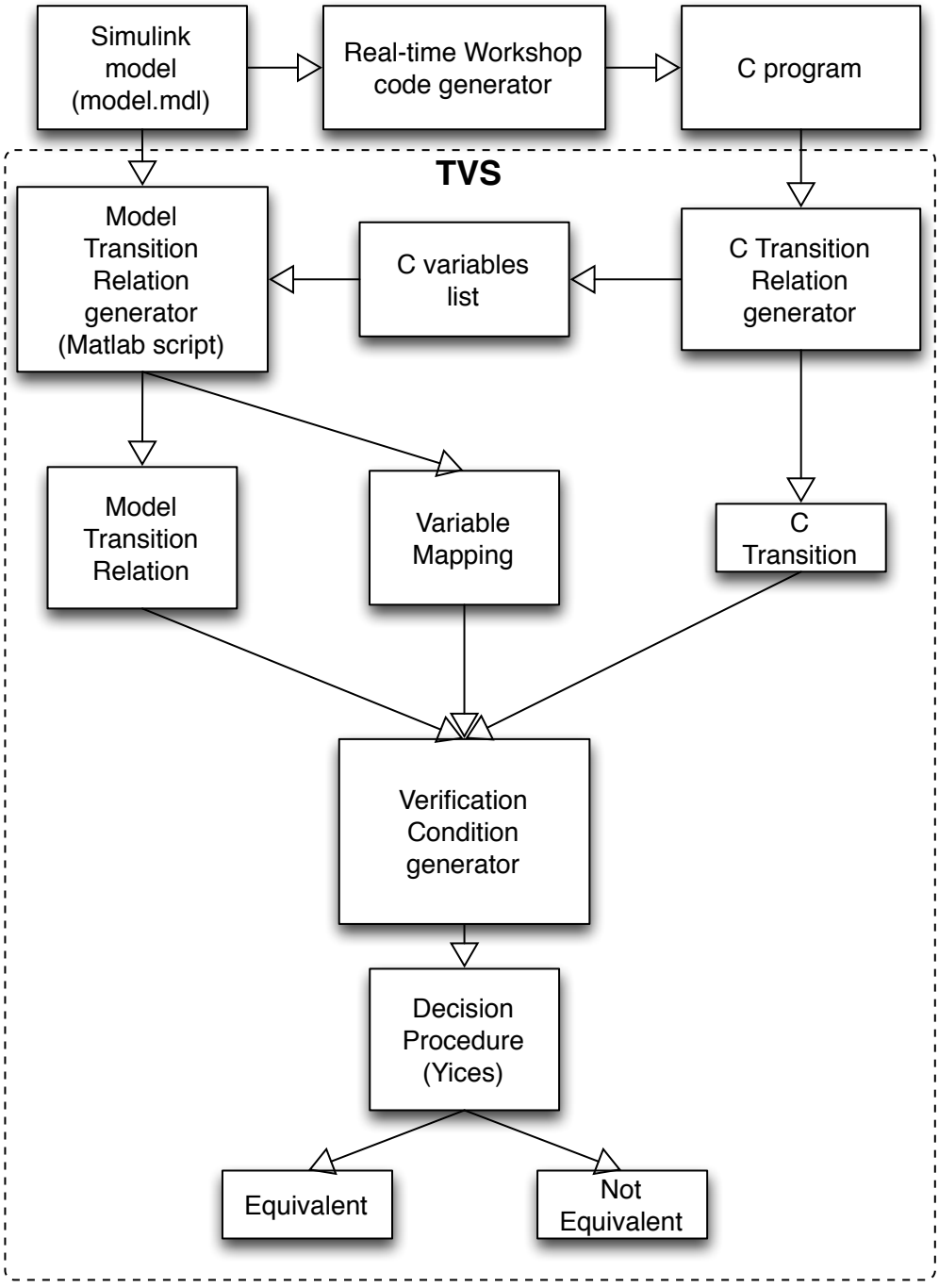

Figure 2.6. The architecture of the translation validation tool in [5].

non-virtual [5]. Virtual blocks do not have memory and they are only definitions of the signal interconnections (Outport, Mux etc.). Virtual blocks do not have any representation in the code generator output. On the other hand, mathematical operations are performed by the non-virtual blocks (Sum, Gain blocks etc.). Those blocks have representations in the code generator output either as a propagation of its operation or as a variable.

As it is explained for the work in [5], determination of the type of a block is an important part in the translation validation problem. Following the correct determination of the type, formal representation of the blocks in Simulink model is achieved. In [6], a similar methodology is adopted for transforming the Simulink model. In [6], the model is parsed, the hierarchy is flattened and a valid execution order of blocks for the resulting ANSI-C model is derived by the front-end. Then, each block instance is processed in order to identify its type and its block specification. Following identification of each block instance, they are mapped to a block-type in the library. In the last step, C block-type definitions required for inclusion are determined and values are assigned to block properties. In Figure 2.7, the basic structure of the generated C code for the Simulink Model example in Figure 2.3 is shown.

```
/*1.Links to blocktypes definitions */
#include <Sum.h>    // Corresponds to the Add block
...

/*2.Declaration and initialization of block instances */
Sum_in_t b3_in;
Sum_out_t b3_out;
const Sum_prop_t b3_props={.Inputs="++"}; ...

/*3.Simulation loop: define data dependencies w.r.t.
   block connections extracted from the model */
int main() {
  for(sim_time=START; sim_time<END; sim_time+=sim_step) {
    b3_in.port1 = b7_out.port1;
    b3_in.port2 = b2_out.port2;
    b3_out = *Sum_semfun(&b3_props, &b3_in); ...
  }
}
```

Figure 2.7. Overview of the structure of the C code generated for a Simulink model [6].

A well-defined formal representation of the Simulink programs does not exist considering the presented literature on SEC of Simulink. [46] works on developing a modeling approach for Simulink diagrams. They propose a recursive approach for modeling a class of Simulink diagrams as input/output-extended finite automata (I/O-EFA). I/O-EFA is a tuple and is a function of symbolic-states, data, numeric-inputs, numeric-outputs, symbolic-inputs, symbolic-outputs, initial location, initial-data values, final

location, and the set of edges. The input-output behavior of an I/O-EFA model preserves the input-output behavior of the corresponding Simulink diagram at each sample time. In [46], the authors treat the blocks in the Simulink library to be "atomic" and formulate two rules (Connecting-rule, Conditioning-rule) used in Simulink for building complex blocks by combining the simpler ones. They provide a recursive and formal definition for the class of Simulink diagrams formed using these rules.

There are some other studies aiming at finding formal representation of Simulink programs. In [47], an automated translation of MATLAB Simulink / Stateflow Models to an intermediate format based on Hybrid Automata in HyVisual is presented. [48] also discusses the translation of Simulink and Stateflow (State Machine Modeling Toolbox of Simulink) models to Hybrid Automata. In addition to formal verification, there are some studies that rely on simulation and testing. In [49], validation of Real Time Workshop generated C code by means of testing is studied.

2.2. Analog Approach

On the verification of analog models, there are two different approaches. One is checking the model against a written property, namely, analog model checking. The other is checking whether the implementation corresponding to the model behaves in the same way with the model, or not, namely, analog equivalence checking. Researches on these two different approaches will be covered in two different sections.

2.2.1. Analog Model Checking

Checking whether a discrete state-space model of an analog circuit satisfies a specified property or not is defined as model checking and it is illustrated in Figure 2.8. Because of the importance of property specifications in model checking, the basics of them will be discussed, then analog model checking survey will be provided. Our motivation in surveying model checking was to make a feasibility analysis to decide whether we could use model checking in our analog equivalence checking approach or not. In our feasibility analysis, we realized that although it is good to use analog

model checking as a formal approach in EC, its applicability is limited by state space explosion problem. We will present the recent researches in the application of analog model checking in this section, although its applicability is limited.

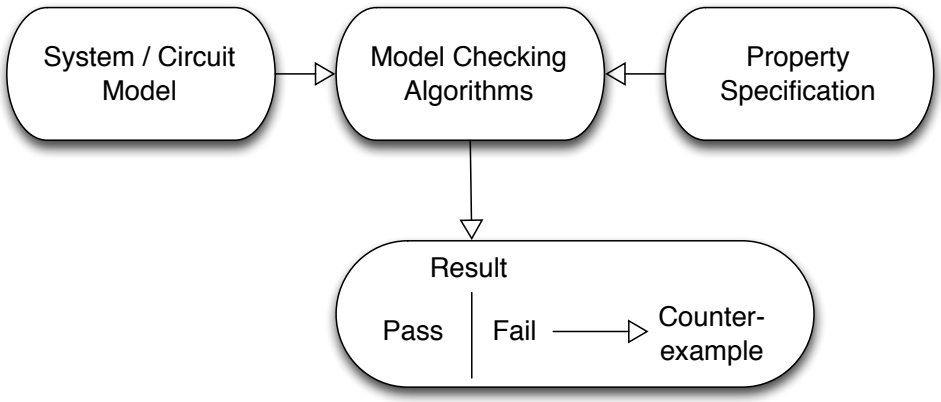

Figure 2.8. Model checking flow [7].

Development of formal properties for systems has its roots in the application of temporal logic to verification of programs [7]. Linear Temporal Logic (LTL) [50] is the first approach introduced. Its time model is non-branching and linear, and operators work on atomic propositions. Computation Tree Logic (CTL) [51] is different from LTL such that its time logic is branching, and in accordance with this fact formula validity is specified on branching computation tree. There is no change for the syntax of the LTL temporal operators, however, a path quantifier is introduced for each temporal operator. The path quantifier either declares holding of formula for all possible paths (defined as A) or defines existence of a path for computation tree (defined as E). Model checking approaches using CTL were applied at the end of 1990s in the area of digital circuit design.

"Sugar", which is an in-house language built in IBM, was developed in order to meet the expectation of having a more user-friendly language. Sugar was chosen as the base for a new standardized property specification language (PSL) by industry committee and PSL was standardized as IEEE 1850 [52]. PSL can communicate with SystemC, SystemVerilog, Verilog, and VHDL by means of the common layer it builds with verification and specification.

General approach in analog model checking area is to extend the capabilities of the aforementioned specification languages in order to be able to express the analog properties. One of those extensions in syntax is Signal Temporal Logic (STL), which was done for PSL. In STL, proof of properties are done on waveforms for transient simulations [53–55].

In [56], another extension of CTL, which is named as CTL-A is proposed. In this language, the operators $<$ and $>$ are introduced. With that extension, various properties of nonlinear circuits like oscillation, overshooting and switching have been proved [57]. Another extended CTL derivative, AnaCTL, is applied to transient response of analog circuits in [58, 59]. Another extension, CTL-AT was developed for CTL-A to define the timed behavior explicitly [60].

In [7], a new property language named Analog Specification Language (ASL) is proposed. The reasons for the necessity of that language are as follows:

- Analog operators in CTL-AT enable specifying some basic properties like reachability of states from DC-operating points when they are used in combination with time constraints. However, there is not a systematic way to specify more advanced analog properties.
- For complex analog behaviors, the temporal methodology will not be sufficient for specification. Additional operations will be required to determine the state space sets.
- Since CTL originates from temporal logics, it does not offer a specification methodology, which is designer-oriented.

In addition to the usage of extended specification languages, Labeled Hybrid Petri Nets (LHPN), which are used in high-level modeling, are utilized in verification of AMS systems [61, 62].

2.2.2. Analog Equivalence Checking

Since analog systems cannot be enumerated, equivalence checking of them should be handled different from its digital counterpart. Some approaches define a tolerance margin for the equivalence, since exact equivalence is impossible [63], whereas some utilize abstractions following the definition of assumptions.

An approach bypasses enumeration problem of analog systems by assumption of linearity of analog circuits. In [64], the authors propose a method for applying equivalence checking to two analog designs defined by their linear transfer functions. The method of verification utilizes bilinear transformation to discretize the transfer functions to express them in the Z-domain. Hence, the designs are defined by using discrete-time elements and transformed to binary decision diagrams (BDDs), which are similar to Finite State Machines. The ideas in [64] have been extended in [65] such that the matching between the implementation and specification is verified in a given frequency range considering the magnitude and phase responses of both designs if the transfer functions of both designs are provided. In [66], the authors introduce an equivalence checking approach to verify that a circuit realizes a specification over a defined frequency range considering all the possible values of parameters. As stated in [29, 30], similar to boolean representations for digital circuits, transfer functions should be the abstractions of analog circuits. In those studies, transfer functions are obtained by means of fitting a linear model using linear regression. In case circuit is highly non-linear, converters are put at the input and output to remove the non-linearity and to make the remaining block in the middle linear. This is visualized in Figure 2.9 for PLL.

It should be noted that the methodologies using linearization are not practical since transfer function generation is very difficult in general. For [29, 30], linearization of the circuit, which is to be checked for the equivalence, needs manual intervention to the checking.

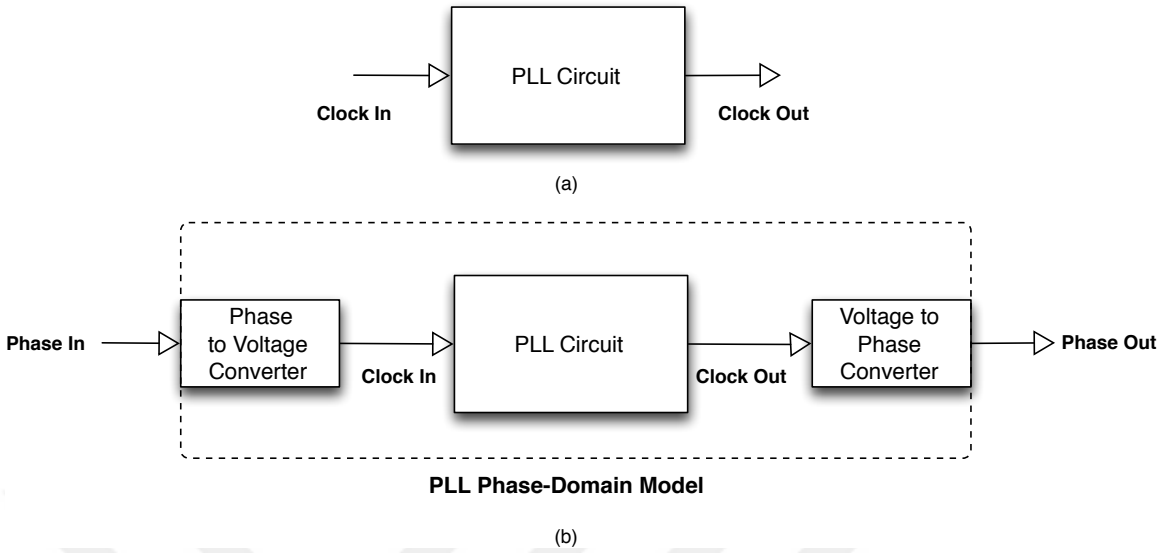

Figure 2.9. Linearization of PLL in the study [2].

A state space sampling based approach for equivalence checking of nonlinear analog circuits is introduced in [10] with application to different CMOS inverters. The authors propose application of non-linear transformations to the sample state spaces to make the comparison possible. Study is extended in [8] for making it possible to apply the approach to a Schmitt-trigger and bandpass transistor circuit. In [9], the approach is extended to enhance handling of nonlinearity in the circuit. Those extensions are structural recognition, reachability analysis and eigenvalue mapping to circuit elements. This enabled application of equivalence checking to the analog behavior of a NAND gate and a mixer circuit. The structure of the methodology is shown in Figure 2.10. However, determination of the exact transformation is not easy and automating it is not feasible. These lead to the definition of some heuristics to qualify the behaviors of the circuits for analysis, which affects the applicability of the approach.

In [11], another equivalence checking approach is proposed for verifying VHDL-AMS designs. In this approach simulation, equivalence checking and rewriting systems are combined into an environment. As part of the verification approach, codes for implementation and specification are decomposed into parts of data converter, analog and digital. Classical equivalence checking approaches are applied to the digital parts. For analog parts, both designs are simplified utilizing pattern matching and rewriting rules. Finally, the outputs are sent to comparators in order to check them using

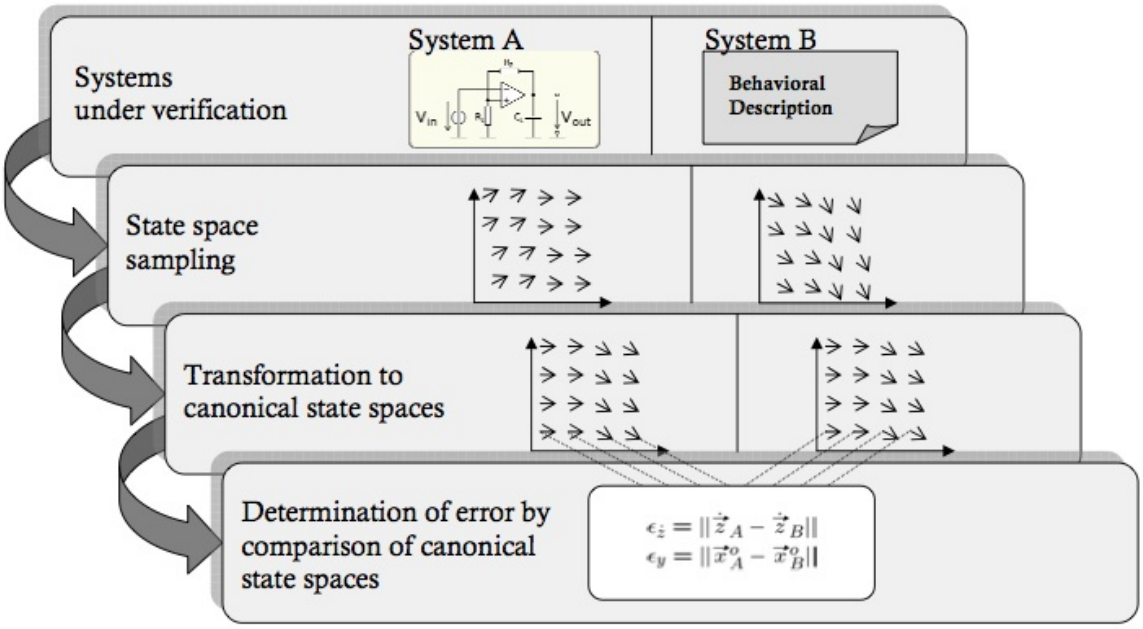

Figure 2.10. The structure of equivalence checking in [8–10].

simulation. Methodology is illustrated in Figure 2.11.

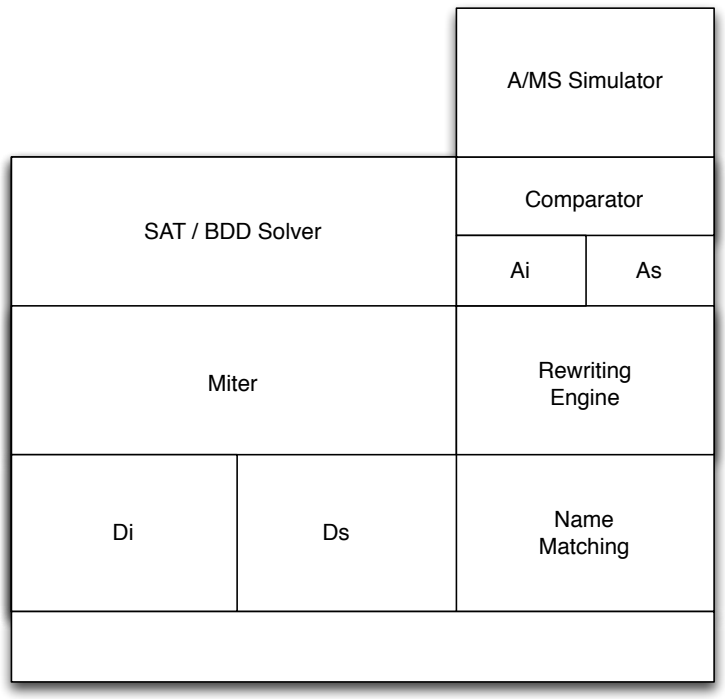

Figure 2.11. The methodology in [11].

The syntactic method presented in [11] can only be applied to simple designs, for which it is easy to utilize the rewriting rules in the application. Although the introduced approach is practical, it does not consider the coupling between the digital

and analog parts. Verification approaches for analog circuits, which are syntax based, are applicable only when the design is performed at high levels. Those approaches become inapplicable in case of circuit having non-linear behavior.

In [7], where ASL is proposed, an equivalence checking methodology based on complete state space-covering input stimuli application is also proposed. Stimuli covering the complete state-space are generated and utilized in verification by means of the flow in Figure 2.12. As stimuli are generated for both implementations to be compared, two implementations are simulated with both its own input stimuli and the other's input stimuli. Then, two simulation results are compared for equivalence with a tolerance margin. This process is shown in Figure 2.13.

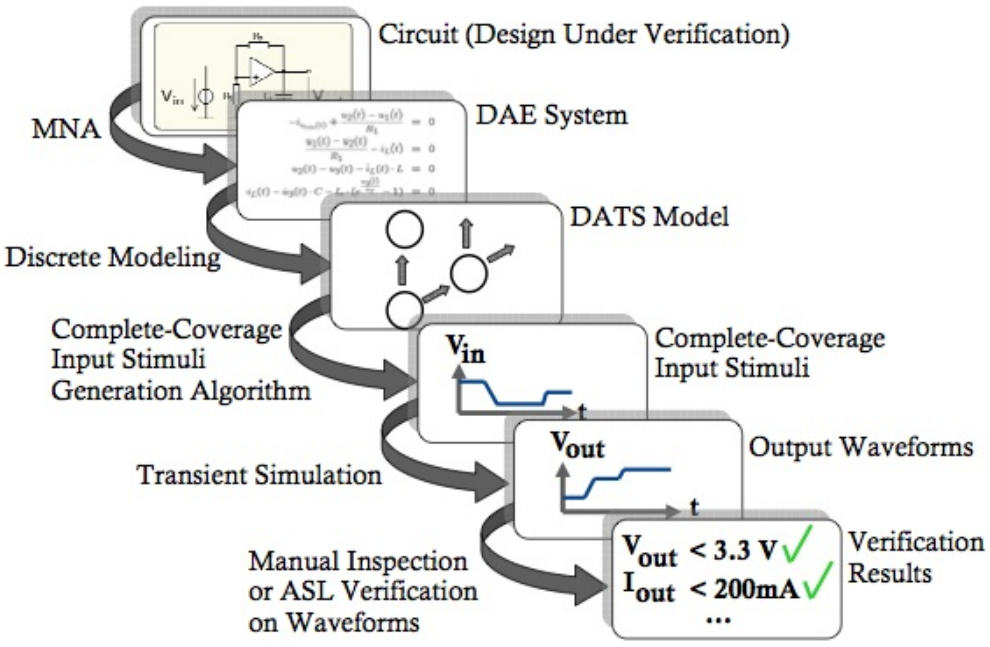

Figure 2.12. Complete state space-covering input stimuli generation and verification flow in [7].

Although the approach in [7] sounds well, its applicability is limited, since the growth in state space variables leads to an exponential increase in the run time.

As another approach, [12] presents a systematic, hierarchical, optimization based semi-formal equivalence checking methodology. Study checks the equivalence between a behavioral model and its electrical implementation over a limited, but highly likely

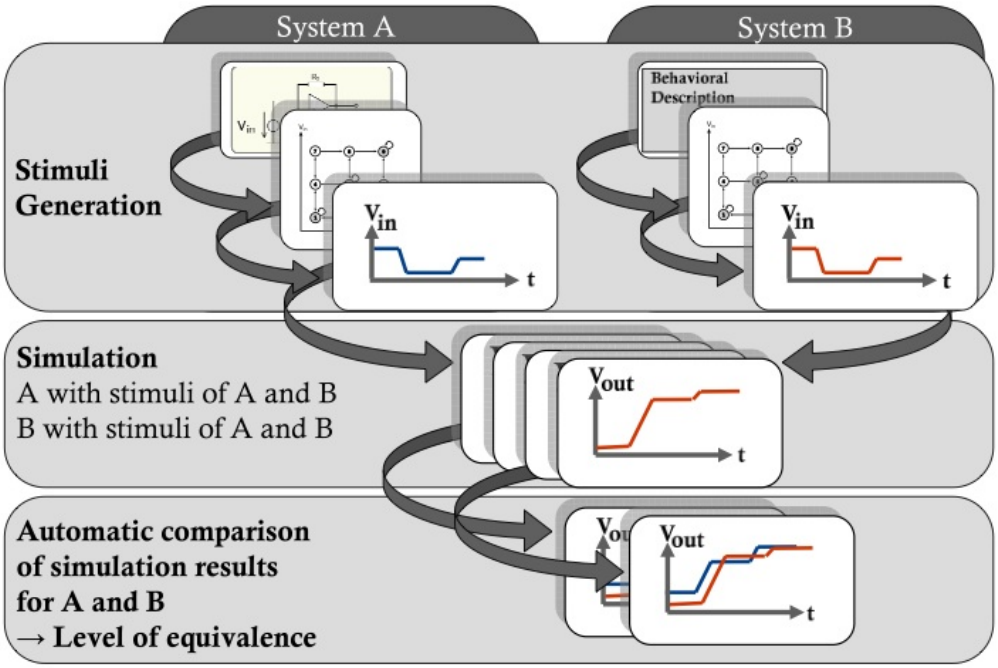

Figure 2.13. The equivalence checking procedure by means of complete state space-covering stimuli application [7].

input space named as the Constrained Behavioral Input Space. This input space is obtained from system level simulations. Reason behind defining a constrained input space is given as the state space explosion, which limits the usage of other equivalence check methodologies like in [7]. Further, study distinguishes between the behavioral and electrical domains and defines mappings between the two domains to allow for calculation of deviation between the behavioral and electrical implementation. This explanation can be visualized by Figure 2.14.

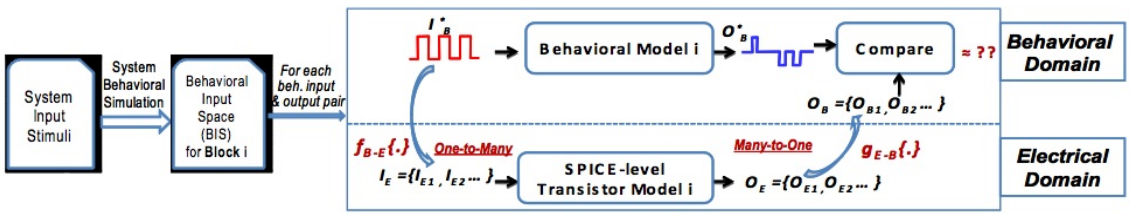

Figure 2.14. Signal mapping between the behavioral and electrical domains [12].

Then, the verification problem is formulated as an optimization problem, which is solved by interfacing a Sequential Quadratic Programming (SQP) based optimizer (DONLP2) with commercial circuit simulation tools (Cadence Spectre for the study). Actually in optimization, maximum error between the real behavioral output and the

mapped output from electrical domain is searched under some constraints, which may be rise time, fall time etc. The optimization problem is formulated as in Equation 2.1.

$$\begin{aligned} max_{p_E} \quad & \varepsilon = h_{err}(O_B(p_E), O_B^*) \\ & subject\ to: \\ & \underline{p_E} \leq p_E \leq \overline{p_E} \\ & I_E = f_{B-E}(I_B^*, p_E) \\ & O_E(p_E) = Q_{SP}(I_E(p_E)) \\ & O_B(p_E) = g_{E-B}(O_E(p_E)) \end{aligned} \tag{2.1}$$

Parameters and functions in Equation 2.1 are detailed in Table 2.1.

Table 2.1. Declarations of parameters and functions in the optimization problem definition in [12].

h_{err} - error function	p_E - electrical input parameter
$\underline{p_E}$ - lower bound for p_E	$\overline{p_E}$ - upper bound for p_E
O_B - mapped behavioral output	O_B^* - reference behavioral output
I_B^* - behavioral model input	I_E - electrical input mapped from behavioral
f_{B-E} - function mapping behavioral input to electrical domain	g_{E-B} - function mapping electrical output to behavioral domain
Q_E - electrical domain output	Q_{SP} - SPICE simulation to generate electrical output from input

The equivalence checking methodology in [12] is illustrated in Figure 2.15. Then, the proposed methodology is applied for equivalence checking of a PLL as a test case, by dividing the design into blocks.

The above analog equivalence checking studies except [12] have scalability issues due to the state-space explosion problem. In that sense, we consider [12] as the starting

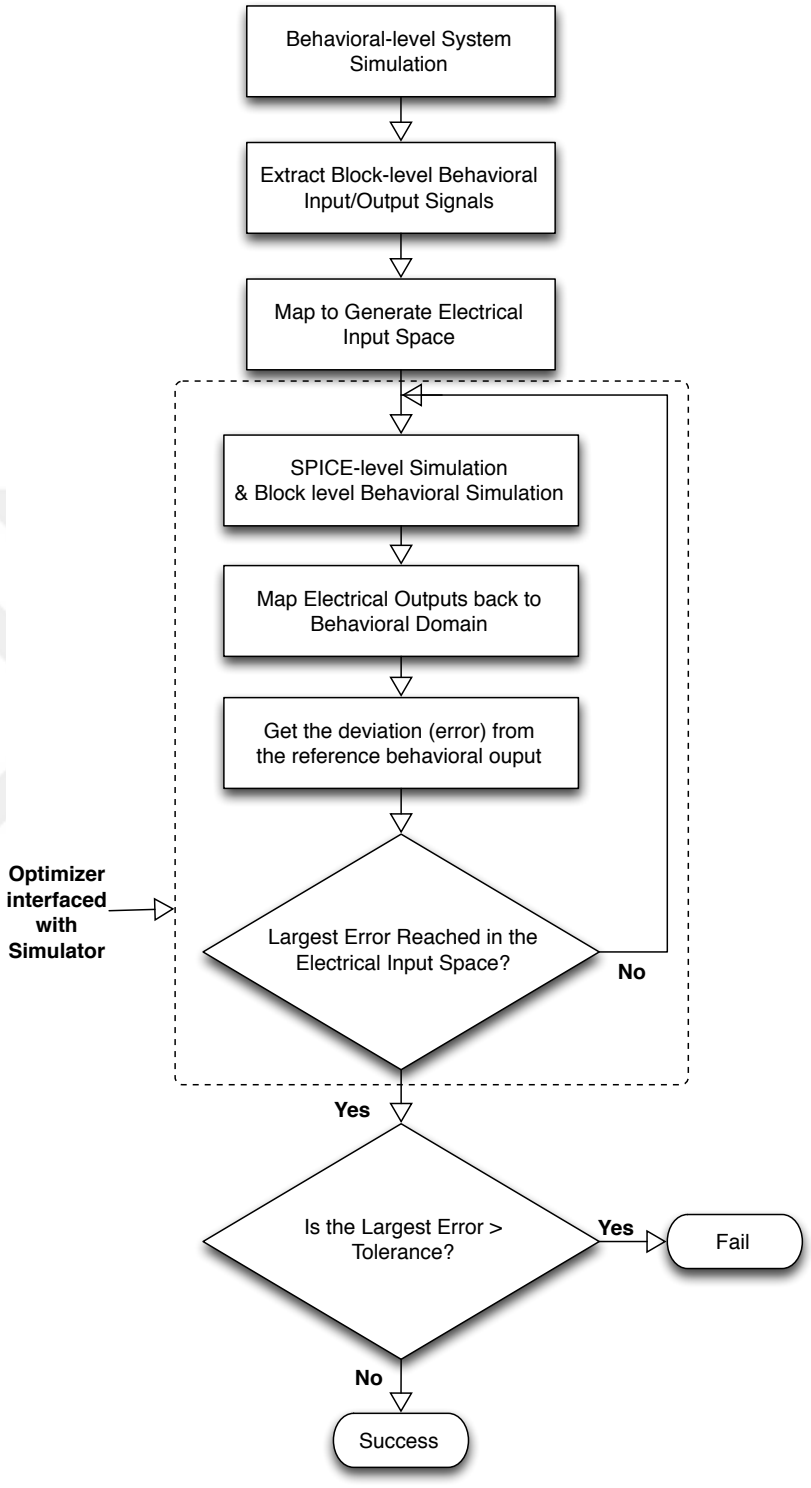

Figure 2.15. Optimization based equivalence checking flow in [12].

point for our research on extending our methodology to AMS systems. In [12], Verilog AMS is the behavioral level of AMS circuit, whereas in our case we use Simulink models.

2.3. Process Variations

Random process variations affect the yield and the quality of circuits significantly in modern technologies [67]. Because of this, analyzing the effect of those variations in an analog equivalence checking methodology is important, in the sense that the analysis may give some clues on the degree of deviation from normal performance of the circuit and on the effect of variations to equivalency. So addition of the variation analysis to our methodology is a contribution to the literature. Estimation based on Monte-Carlo (MC) simulation is currently advantageous as it is highly accurate and general [68]. Estimations of yield with a reasonable accuracy do not require more than several minutes even with primitive Monte Carlo simulations. On the other hand, either for analog sizing considering process variations or optimization of yield, the bottleneck is the estimation of yield via Monte Carlo [67, 69–71]. Because of that reason, in the literature there are various studies targeting that problem and those studies attack the problem from two different sides with regards to Koksma-Hlawka Theorem, which puts an upper bound on the error between the estimation and the real value. Theorem is formulated as in Equation 2.2.

$$|\widehat{I} - I| \leq D_n^*(x_1, x_2, ..., x_n) V_{HK}(f) \tag{2.2}$$

In Equation 2.2, the uniformity of the generated samples are represented by the star discrepancy, D_n, and its value becomes lower, when points are distributed more uniformly. Overall variation of f is represented by $V_{HK}(f)$ and the problem determines its value.

For diminishing the error, as the first approach decreasing of $V_{HK}(f)$ is attempted and those methods are named as variance-reduction methods. One of those methods is Importance Sampling, which aims at reducing variance by taking more samples from the tails of the Gaussian distribution (Parameter of interest is assumed to have Gaussian distribution). The method can be visualized better in Figure 2.16.

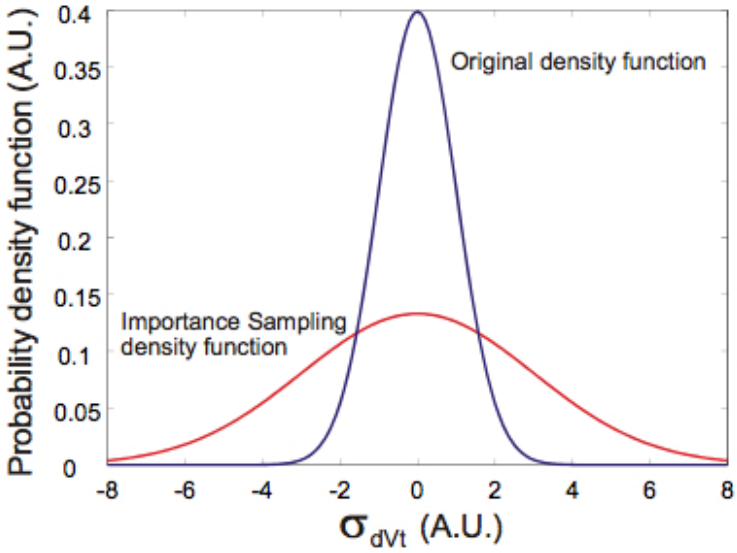

Figure 2.16. Effect of importance sampling on density function [13].

This approach is successfully applied in [13] for accurate estimation of SRAM yield. Although importance sampling method is successfully applied, it is criticized to be circuit-specific, which leads to a problem of generality.

For reducing the estimation error, decreasing the star discrepancy is attempted, as the second approach. One of the methods, which utilizes that approach is Quasi Monte Carlo (QMC) using low discrepancy sequences (LDS) to generate more uniformly distributed samples. QMC overcomes the generality problem, which importance sampling method has run into. However, in order to realize the advantage of the QMC, the required number of samples to be added to set becomes so large for high-dimensional problems [72]. If the number of samples is less than the required value, the uniformity of points degrades after the first few dimensions, thereby, worsening the performance at higher dimensions. In order to alleviate this problem, non-uniformity is sacrificed in higher dimensions to decrease the loss to a lesser value, as proposed in study [68], which is named as ranking-integrated QMC. In that approach [68], process variation variables are ranked according to their sensitivities in the first step. Then, considering the rating in the initial phase, more significant variables, which mainly determine the variance are chosen.

In [67], it is claimed that even utilizing the methodology in [68], for LDS, the high threshold on dimension is typically 10-12, in order to preserve the uniformity. In [67], a new method named AYLeSS is proposed, which eliminates the limitation problem imposed on the effective dimension number. The main idea of AYLeSS is the splitting of the variables for process variations into sub-groups and then applying ranking-integrated QMC to each sub-group. Experimental results in [67] demonstrate the achievement of 1.5 to 2.5 times enhancement in speed in comparison to ranking-integrated QMC for AYLeSS. Furthermore, those results prove that AYLeSS is able to handle designs with many process variation parameters.

Another study [73], attempts to enhance Quasi Monte Carlo techniques to be able to apply them to Statistical Static Timing Analysis (SSTA). In order to be truly competitive with existing SSTA tools, new method should require fewer random STA runs, such that typical sample sizes of 10000 STA runs should be decreased to a few 100s of sample points. The methodology, which makes that reduction possible synthesizes two recently introduced ideas. The first is reduced sampling via deterministic QMC methods. The second one aims at decreasing dimensionality by a Karhunen - Loeve expansion (KLE) of a random field model of intra-die variation [74, 75]. In order to utilize QMC effectively in high dimensions, it is required to map "important" statistical variables to the individual dimensions of the sampling process. To that end, utilization of either rank correlation coefficients or designer expertise is suggested in [76]. For SSTA, there are two obstacles that pose problem on application of QMC. Firstly, the inherent dimensionality is huge (up to millions) and secondly, it is not possible to trace neither designer expertise nor rank correlation in measurement of statistical importance of each variable. In order to bypass those obstacles, an enhancement of KLE-based spatial correlation modeling proposed in [74], which was made applicable by [75], is required. A very large model with lots of random variables (RVs) to symbolize gate-level statistics can be fed to the KLE of a random field model of intra-die statistical variation to reduce them to few tens of uncorrelated RVs. By the combination of QMC and KLE, study achieves at error percentages smaller than 5% at a confidence level of 95.45% with a few hundred samples [73]. Errors are actually measured as 2.09% on average, and for the ISCAS89 circuits, the method runs between a few seconds and

90 seconds [73]. It is claimed in [73] that QMC-based SSTA can become applicable to complex circuits in industry, if multi-core processors are utilized or the implementation of the method is optimized.

There is a formal approach to verify analog circuits in the presence of noise and process variation in [77], however, it is limited to linear time-invariant designs, since it is dependent on the closed form solution for the statistical differential equation (SDE). In [78], again SDE is utilized in verification of analog circuits in presence of noise and process variations using pattern matching. The same limitation of SDE applies here as well and the methodology is limited to fixed simulation step-size.

3. DIGITAL APPROACH

3.1. Introduction

Model Based Design is a natural response to today's IC Design industry trends. In model based design, an executable model description is the starting point to define the system at the highest level of abstraction. Languages like C/C++, SystemC are widely used in model based design [79]. Matlab/Simulink is also commonly used as a platform for modeling algorithm specifications and simulation, so that it is adopted for merging design, optimization, and ASIC hardware verification steps as in [3], and defined to be the starting point of model based System-on-Chip (SoC) design at the system level [4].

Automatic generation of low-level codes is another motivating factor of model based design, since manual implementation of the system level model is error prone and time consuming. Automatically generated codes can be in the form of RTL (Verilog, VHDL) or C/C++. There are various High Level Synthesis (HLS) tools generating RTL codes from C/C++ and SystemC. Simulink has tools, namely Hardware Design Language (HDL) Coder and Real Time Workshop (RTW), for generating RTL and C codes, respectively. In [80], Simulink and HDL Coder are used for High Level Modeling and code generation of Low Density Parity Check Decoders. In [4], RTW is utilized for generating C code from the Simulink model as a part of their SoC design flow in system level.

As it is the case for other HLS processes, correctness of the automatic code generation needs to be checked. In [5], formal validation of the translation from Simulink to C by means of RTW is studied for a subset of built-in Simulink blocks. However, to the best of our knowledge, formal equivalency of the HDL code generated by Simulink HDL Coder is not investigated in a study.

In the IC design industry, it is a common practice that the Simulink model is designed by system designers and the RTL counterpart is written by the design engineers. This process may be error prone, even if the same designer develops both codes. Thus, the equivalence between the Simulink model and its RTL counterpart should be guaranteed in order not to have misalignments. It is possible to use simulations at system level and RTL designs for checking equivalence. However, this suffers from coverage, whereas formal equivalence checking gives 100% coverage. In that respect, a methodology for checking the equivalence between Simulink models and their RTL implementation is required. In this chapter, a formal equivalence checking methodology for Simulink and RTL is proposed.

A typical Simulink model comprises blocks from three different domains: Built-in Simulink blocks, Stateflow charts, and User-defined blocks. For a Simulink model comprising blocks of all those three types, our methodology utilizes Matlab's HDL Coder and RTW tools, Mentor Graphics' Catapult C, and Synopsys' Formality in the flow. To summarize the proposed methodology, HDL files for built-in Simulink blocks and Stateflow charts are generated by HDL Coder. Then, for user defined blocks, which are S-functions in our case, C files are generated by RTW. Since HDL Coder is not able produce HDL files for those blocks, the generated C files are further arranged in a form to be fed to Catapult-C which generates the RTL files. Finally, the integration of the RTL files generated by both Simulink HDL Coder and Catapult-C forms the complete RTL implementation of the design. At this point, it may be questioned why automatically generated codes are not used as the RTL counterpart. Metrics like performance, area and power are still better for manual implementation, so it is the preferred implementation style. In the last stage, both the manual code and the automatically generated RTL designs are loaded into Formality for formal equivalence checking. Figure 3.1 illustrates our methodology, details of which will be explained in the following sections.

3.2. Methodology

3.2.1. Simulink Background

Simulink is a software for modeling, simulating, and analyzing real time systems [44]. Simulink models consist of a set of blocks that are connected by signals specifying the flow of data. Models can be structured hierarchically with the help of subsystems. Simulink provides predefined blocks, which are organized under several block libraries, like math operations and lookup tables.

In addition to predefined blocks, Simulink models can include state machines designed by Stateflow. Stateflow extends Simulink with a design environment for developing state charts and flow graphs. Furthermore, Simulink user defined blocks can be added to Simulink by means of Matlab functions or System-functions (S-functions). S-functions are written in C in a pre-defined format just like function calls used for built-in Simulink blocks. After compilation of those files to Matlab executable *mex* files, S-functions can be integrated to Simulink. This predefined integration process is automated by means of S-function builder functionality of Simulink, which generates files in the predefined format as C files.

Simulink provides two tools for automatic code generation: (i) HDL Coder generates portable, synthesizable VHDL or Verilog code from user defined functions, Simulink blocks, and Stateflow charts, (ii) RTW generates C and C++ code from Simulink diagrams, Stateflow charts, and user defined functions.

3.2.2. Building of the Methodology

In a typical digital IC Design flow, in which Simulink is utilized for system level design, it is a common practice that Simulink model and its RTL design counterpart are verified separately in their own domains. This simulation-based practice suffers from incompleteness of equivalence checking. The lack of a methodology for formal equivalence checking of Simulink models is mostly due to the fact that Simulink does

not have formal semantics. That has motivated several studies working on translating Simulink models to a semantically equivalent formal model. In [47], formal model is HyVisual, whereas in [48], it is Hybrid Automata, and in [46], it is Input/Output Extended Finite Automata.

For our specific case of digital IC design, counterparts of the system level models are mostly implemented in Verilog/VHDL. Both of these languages are both standardized by IEEE, and they can constitute a suitable domain with formal semantics into which the Simulink level models can be translated. In this thesis, we opted for Verilog. As the formal equivalence checking part is considered, commercial formal equivalence checking tools like Formality and Conformal already exist. Since Verilog is selected to be the common formal representation domain for both the system level and HDL level designs, a flow for translating a Simulink model into a Verilog implementation is needed first.

In order for a flow to be qualified for conversion from Simulink models into HDL, it should be able to handle all three types of blocks used in Simulink modeling. For a large number of built-in Simulink blocks referred in the distribution document and Stateflow charts, HDL Coder can generate HDL codes. For user-defined blocks, which are defined as S-function blocks, HDL Coder is not able to generate HDL. In order to convert those blocks into HDL, a different approach should be applied. RTW can be employed to generate C files from S-functions. Those C files are in the form of functions and the operation realized by an S-function is described by the function called "*outputs_wrapper*". For the final phase of HDL generation from C files, Catapult C Synthesis tool is utilized [81]. As C files from RTW are input to Catapult C and *outputs_wrapper* function is set as top level, HDL files for S-functions are generated. By integration of these HDL codes for the S-functions to the code generated by the HDL Coder, the complete Verilog representation for the Simulink model is obtained. The final phase of our methodology is to check for formal equivalence of automatically generated HDL code and its manually implemented counterpart. Although this phase could be completed by any available formal checker, in our case we deployed Formality [39]. A visual representation of our methodology is depicted in Figure 3.1.

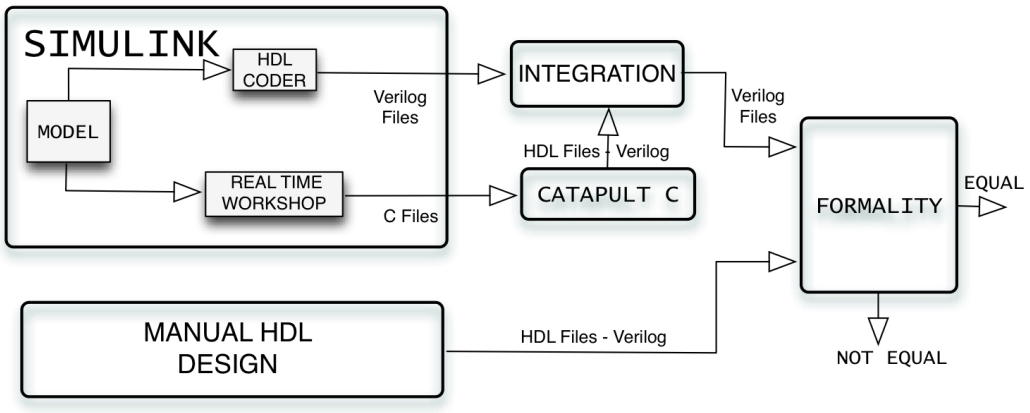

Figure 3.1. Formal equivalence checking methodology.

3.2.3. A Simple Simulink Example

In order to demonstrate the operation of the methodology, we use a simple design in Figure 3.2 including built-in Simulink blocks, a Stateflow Chart and a user-defined S-function block as a case study. Add1 and UnitDelay1 blocks form the built-in Simulink

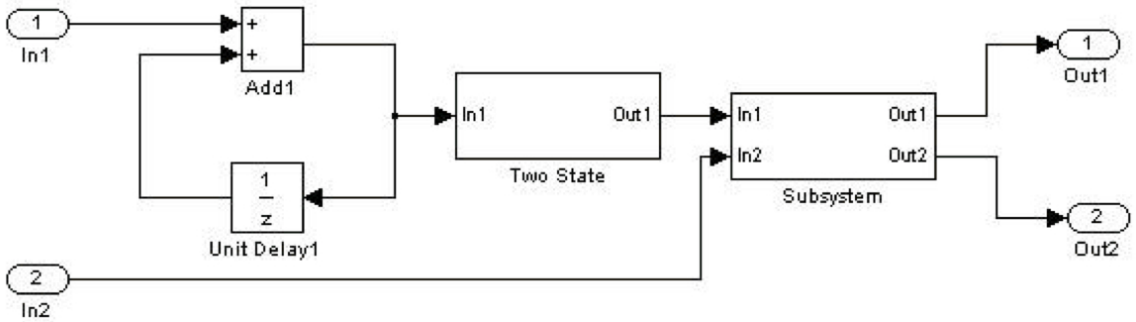

Figure 3.2. Case study with built-in Simulink, Stateflow and S-function blocks.

block part of the design, realizing a counter, which counts in steps input by In1. Following the built-in blocks, Two State Subsystem is a Stateflow Chart shown in Figure 3.3. It has one input and one output. It contains a Stateflow Chart, which comprises two states. The state machine starts with the initial state of State_0, which means output y is 10. According to the value of input x, the state machine can change its state as it can be seen from Figure 3.3. Stateflow Charts are put under subsystems as a requirement of HDL Coder.

The last block named Subsystem is a user-defined S-function, which outputs the difference of its inputs and the addition of inputs from $Out1$ and $Out2$, respectively.

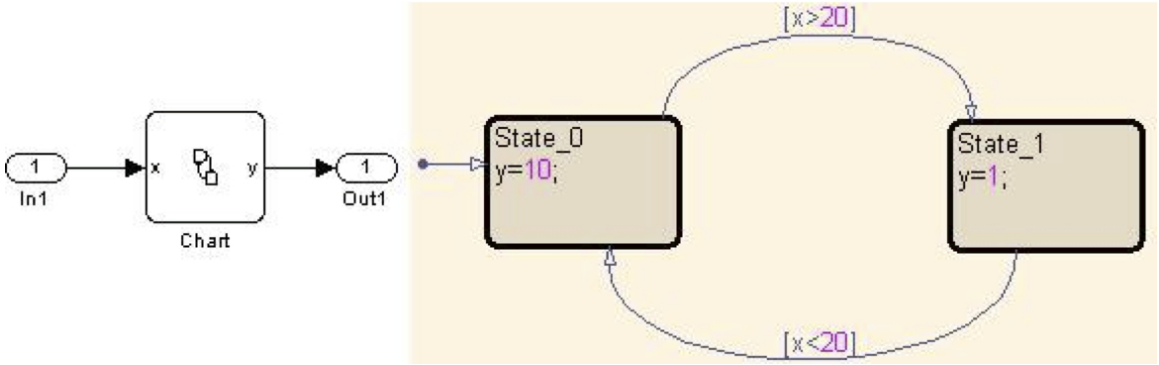

Figure 3.3. Two state subsystem and its state diagram.

I/O's, their datatypes and functionality realized by the S-function are all entered by the S-function builder block of Simulink via GUI as shown in Figures 3.4 and 3.5.

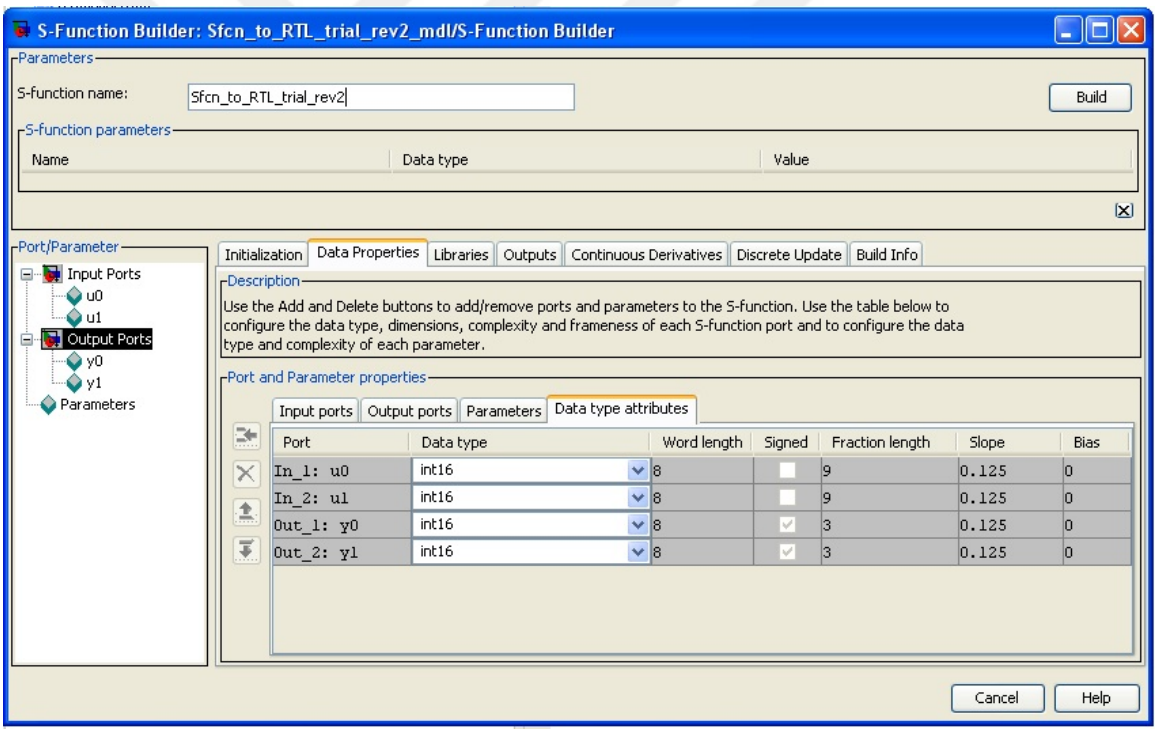

Figure 3.4. S-function builder I/O setting window.

After giving the datatypes of I/O's as Int16, build functionality of S-function builder is run and *.c*, *.tlc* (Target Language Compiler), *output_wrapper.c*, *rtwmake cfg.m* and *.mexw32* files are produced. The *.tlc* file is needed when Real Time Workshop is to be used. In running Real Time Workshop on the developed S-function builder block, the solver should be set to *Fixed-Step* and the target *tlc* file should be set as *rtwsfcn.tlc*. Real Time Workshop run produces an S-function named "*RTWS-function*", which is placed in the subsystem and shown in Figure 3.6.

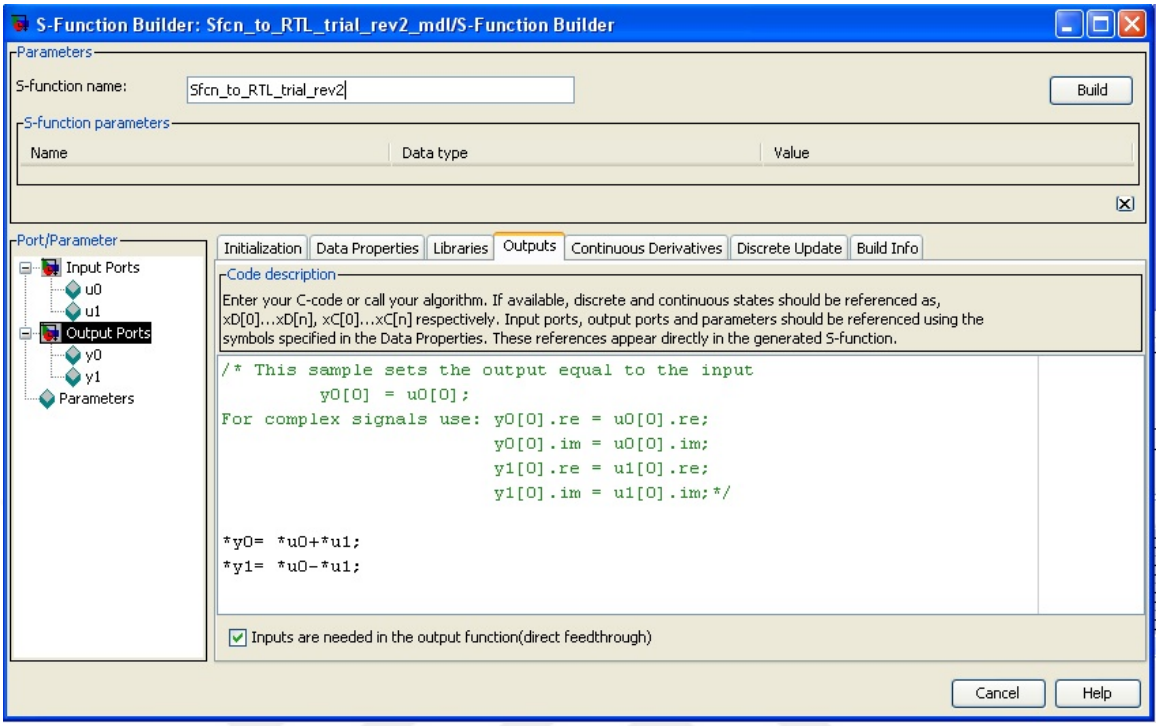

Figure 3.5. S-function builder functionality setting window.

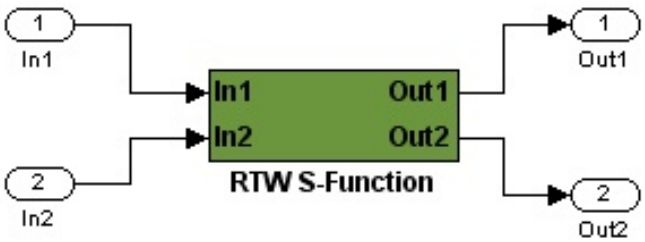

Figure 3.6. Simulink block for S-function.

An HDL Coder run on the design generates HDL for built-in Simulink blocks and Stateflow Chart, but reports failure for the S-function block. Hence, as mentioned earlier, C files are generated for the S-function block by means of RTW and those files are input to Catapult C. For a Catapult C run, header files such as *twtypes.h*, *simstruc.h* etc. are taken from MATLAB include directories. Catapult synthesizes C functions and it expects a function to be set as top-level. C files belonging to S-functions include the functions on Table 3.1.

Functions except *mdlOutputs*, are only for integrating functionality in Simulink flow. So for generating RTL file, *mdlOutputs* file is sufficient. In addition to this,

Table 3.1. C functions belonging to S-functions.

mdlStart	mdlSetInputPortDataType
mdlInitializeSizes	mdlSetOutputPortDataType
mdlOutputs	mdlSetDefaultPortDataTypes
mdlTerminate	mdlSetInputPortFrameData

mdlOutputs function calls a function *...Outputs_wrapper*() which is implemented in the *...wrapper.c* file. So core functionality to be synthesized is defined in the *...Outputs_wrapper*() function which is found under *...wrapper.c* file. Addition of *...wrapper.c* file as input file to Catapult and marking the only function *...Outputs_wrapper*() as top level is sufficient for RTL synthesis. Clock (*Clk*) and reset (*rst*) settings are also done on Catapult C. If the design does not include *clk* and *rst*, selecting Create Combinational Block functionality results in generating the needed RTL as core RTL and leaving *clk* and *rst* unconnected, which is the case in our situation. Consequently *rtl.v* file including the following hierarchy is generated.

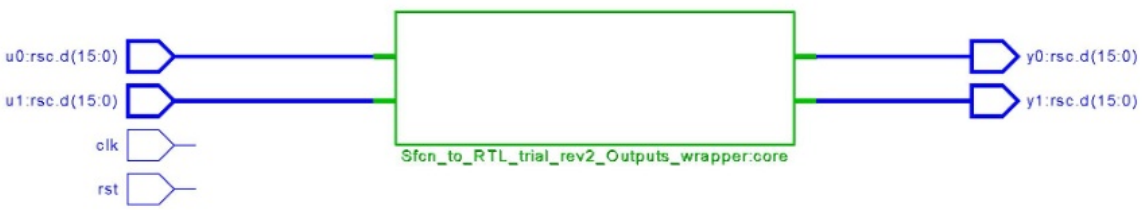

Figure 3.7. Hierarchy for rtl.v file generated by Catapult.

Logic in the ...Outputs_wrapper:core is as in Figure 3.8 below:

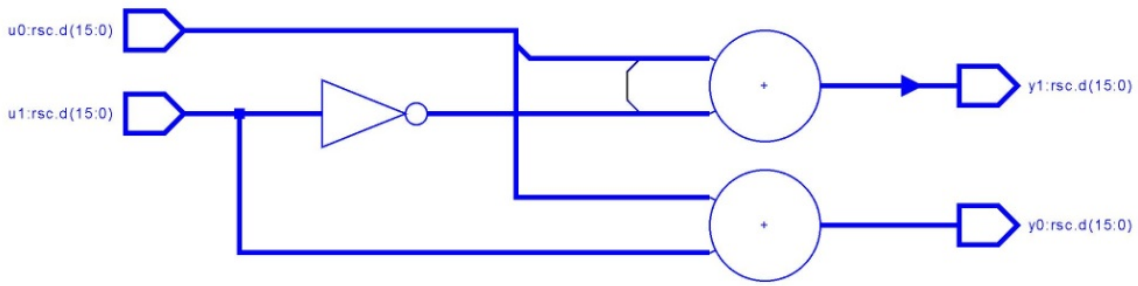

Figure 3.8. Logic implemented by wrapper:core from Catapult.

After the integration of the code for S-function, automatically generated HDL and manually written code are input to Formality, where formal equivalence is checked

and successfully verified. The reference design, implementation design, and finally formality result windows can be seen in Figures 3.9, 3.10 and 3.11, respectively.

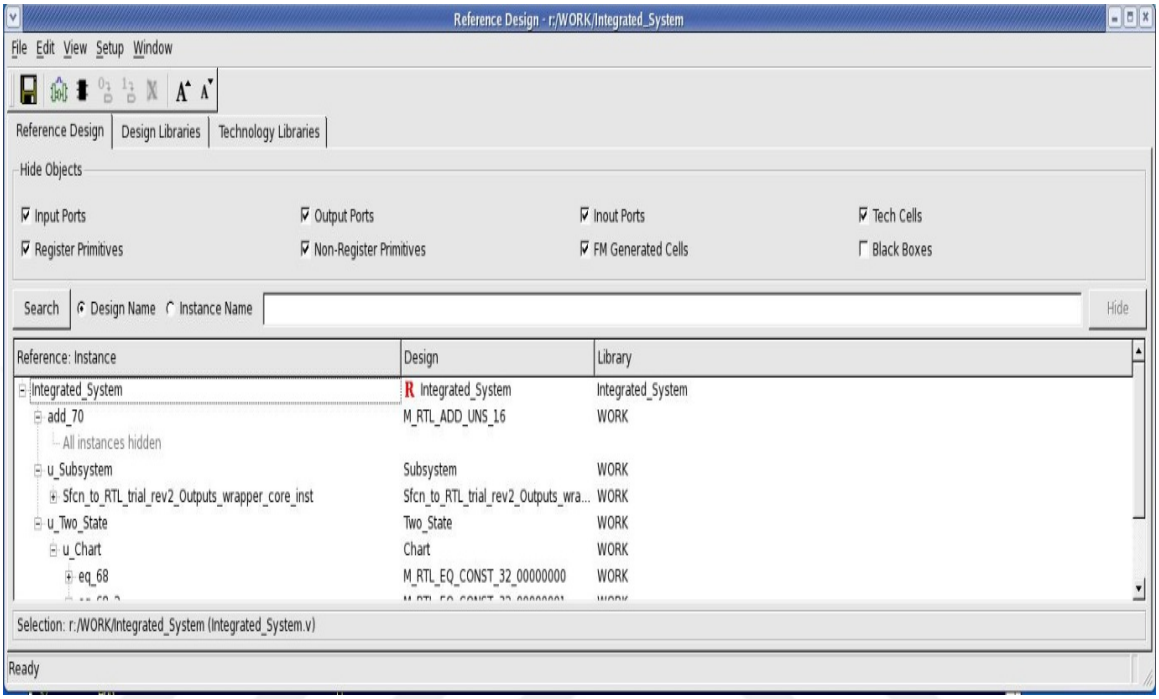

Figure 3.9. Reference design window on Formality.

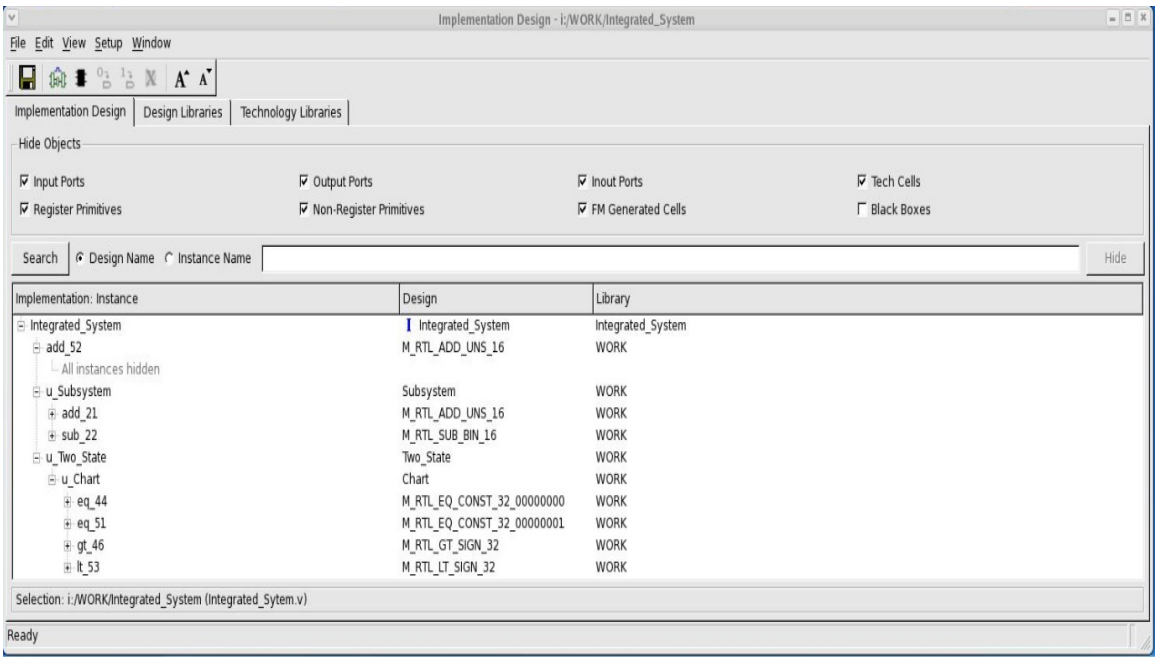

Figure 3.10. Implementation design window on Formality.

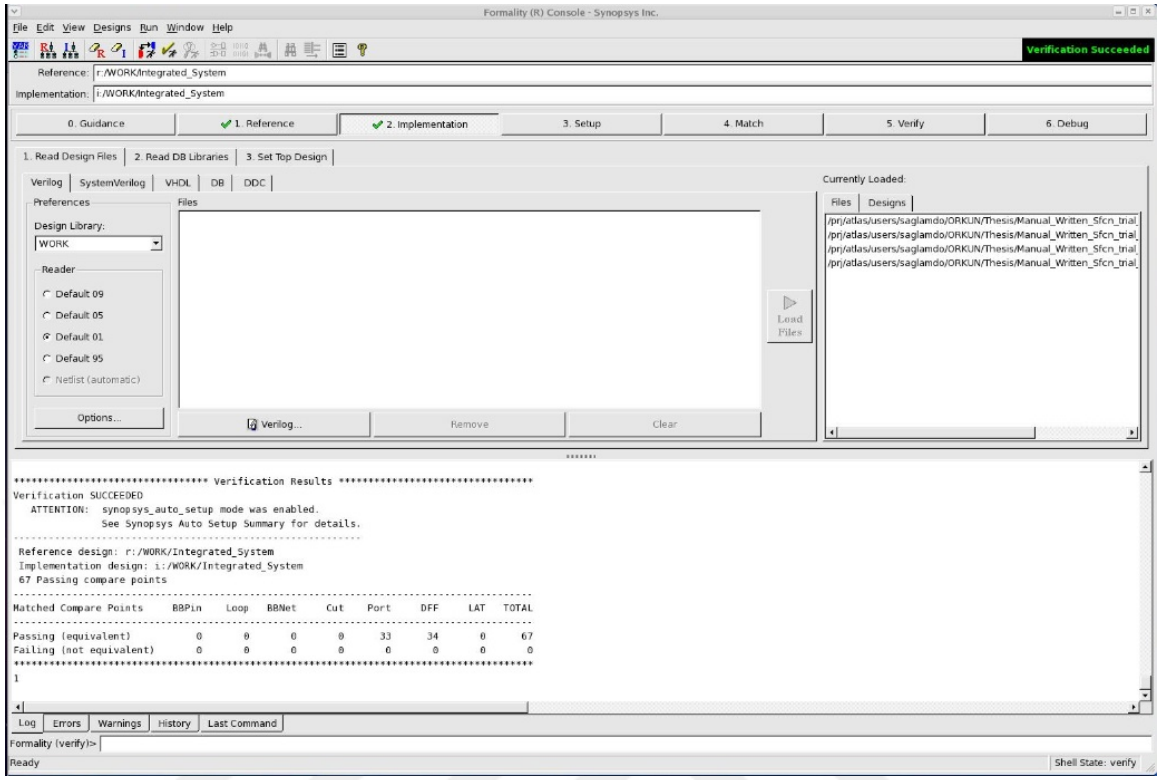

Figure 3.11. Formality result window.

3.3. Experiments

Now, we demonstrate the applicability of our methodology on complex designs. These designs are Configurable Finite Impulse Response (FIR) and Least Mean Squares (LMS) filters, which are distributed in the Matlab package and AES, which is a design of a well-known and published encryption standard.

3.3.1. Configurable FIR Filter

The first design is a configurable pipelined symmetric 32 tap FIR filter. It is designed in a four stage pipelined structure. It has four adders/multipliers/accumulators in each stage and there are 16 coefficient addressable registers in the design. The top level of the Simulink model is as shown in Figure 3.12.

The model consists of two main parts: a finite state machine, which is modeled using Stateflow, and a datapath, which is implemented using generic Simulink blocks.

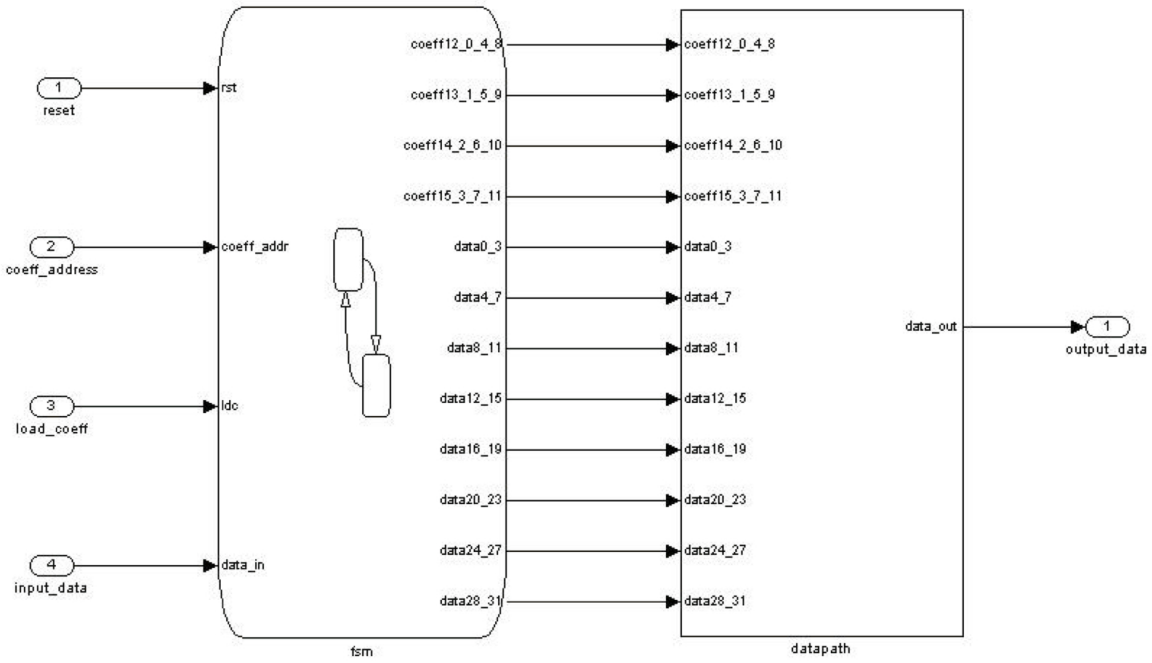

Figure 3.12. The top level of configurable FIR filter Simulink model.

The Stateflow part comprises five states and is implemented using functions and truth-table structures. The state machine is shown in Figure 3.13 and it starts with idle state. Following the disappearance of the reset signal, state machine works by means of functions and truth-table structures. Each coefficient variable is configured to store four different filter coefficients respectively in the 4 different states of the FSM. Similarly, the data variables are configured to store different data values in different states.

The datapath is designed in pipeline form including three subsystems: pre_add (4 adders), mult (4 multipliers) and acc (4 adders/delay elements) in addition to 8 delay elements with three adders as shown in Figure 3.14.

The proposed methodology was followed and the formal equivalence of the manually written HDL code and the Simulink model was verified as shown in Figure 3.15.

3.3.2. Least Mean Squares (LMS) Filter

The LMS filter design takes the input signal, the desired signal, *Step_Size* and *Reset_Weights* as inputs and outputs the error. Input signal is filtered and the differ-

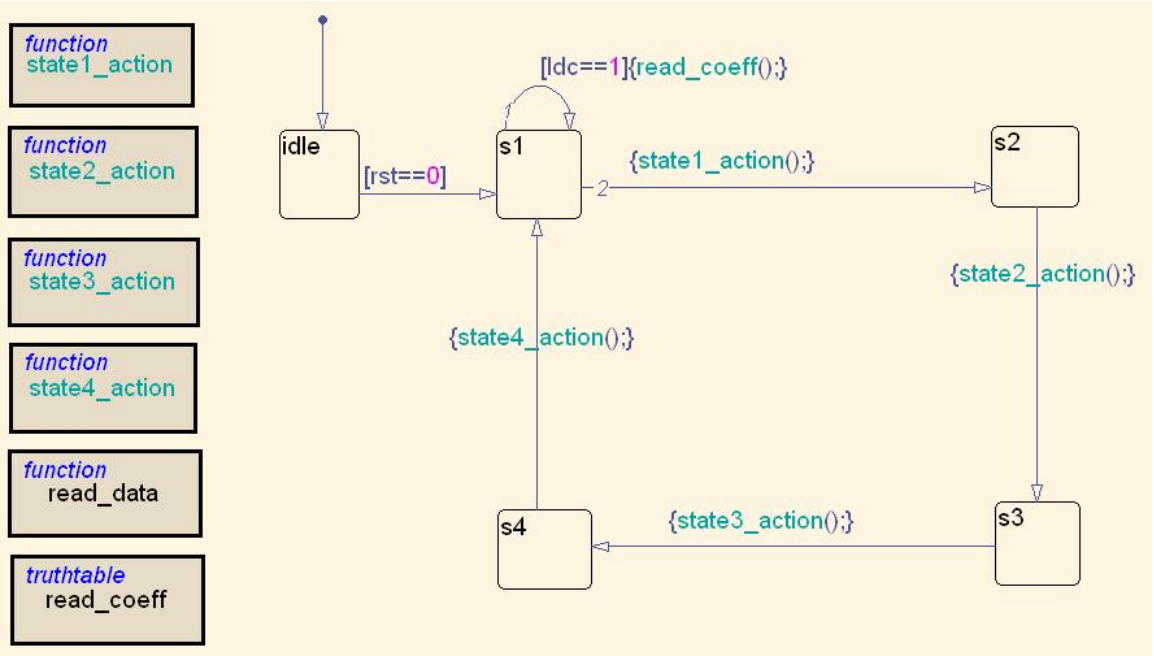

Figure 3.13. Stateflow part of the Simulink model for configurable FIR.

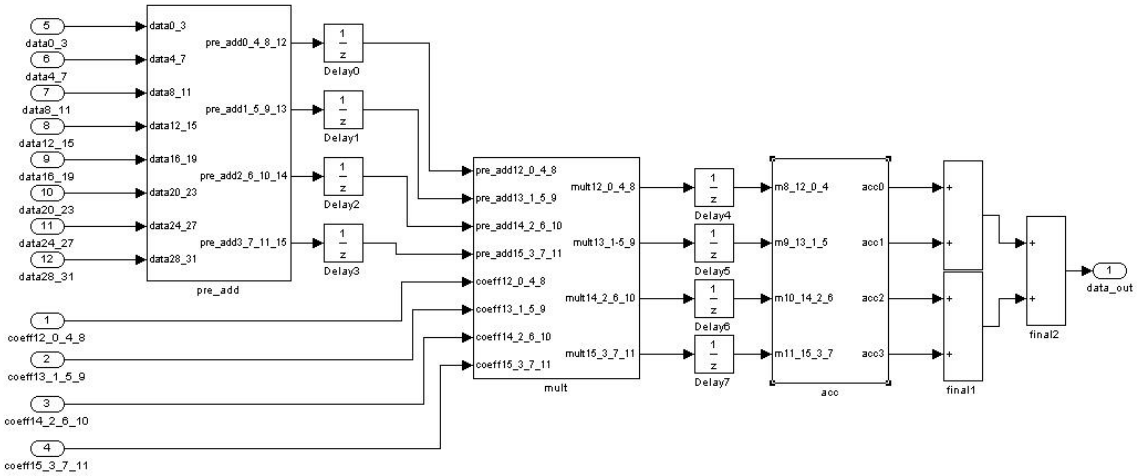

Figure 3.14. Datapath part of the Simulink model for configurable FIR.

ence between filtered signal and desired signal is fed back, as can be seen in Figure 3.16. Each of the four blocks in the figure comprises 10 LMS Taps and 10 adders as shown in Figure 3.17. Each tap is designed using two multipliers, two delay elements, one switch and one adder as shown in Figure 3.18.

Following the methodology, the HDL code was generated. In the generated RTL, different named modules were defined for the same blocks. On the contrary, for each Tap and LMSX10 design, same module name is used in the manual RTL. The generated

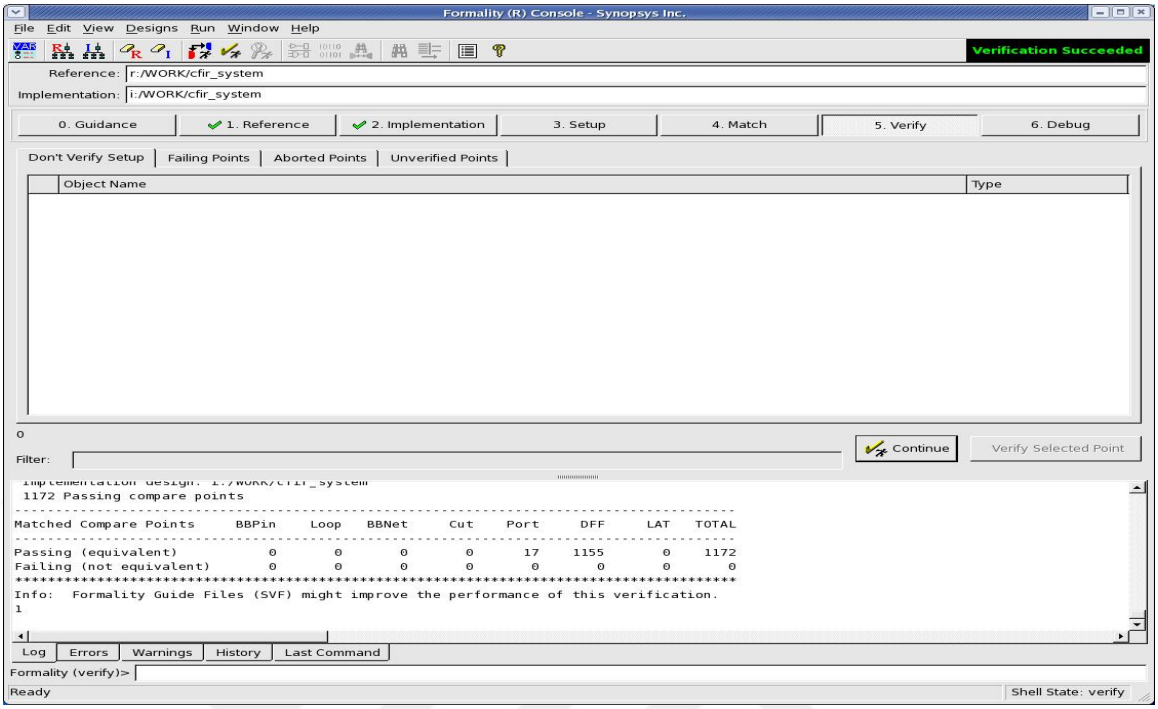

Figure 3.15. Formality window for configurable FIR.

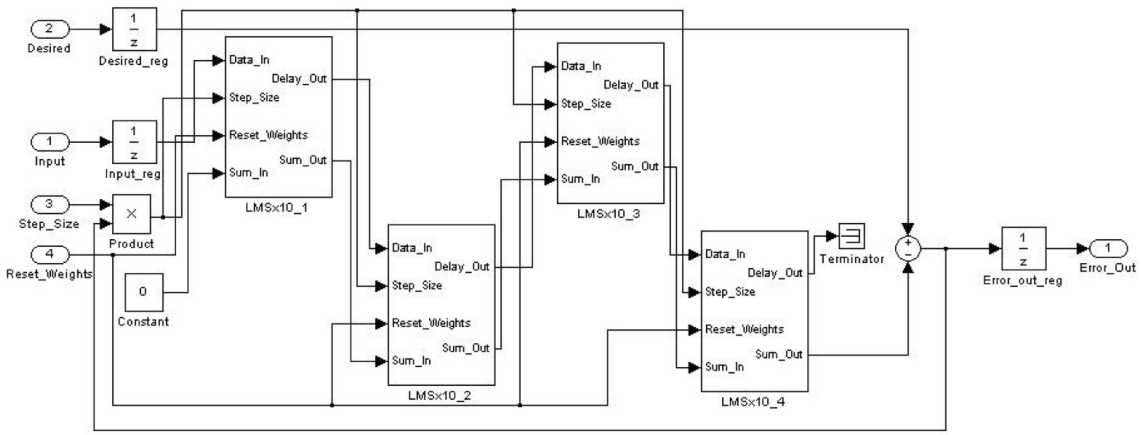

Figure 3.16. The system model of LMS filter.

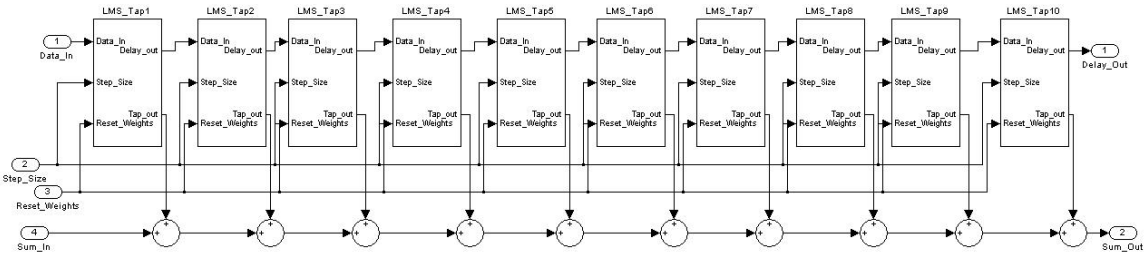

Figure 3.17. LMSx10 blocks of LMS filter.

RTL code was verified to be formally equivalent to the manually written one as shown in Figure 3.19.

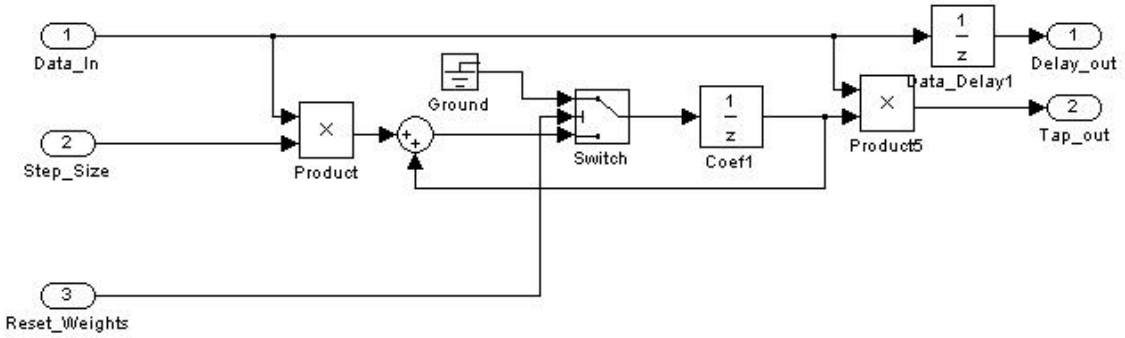

Figure 3.18. LMS tap blocks of LMS filter.

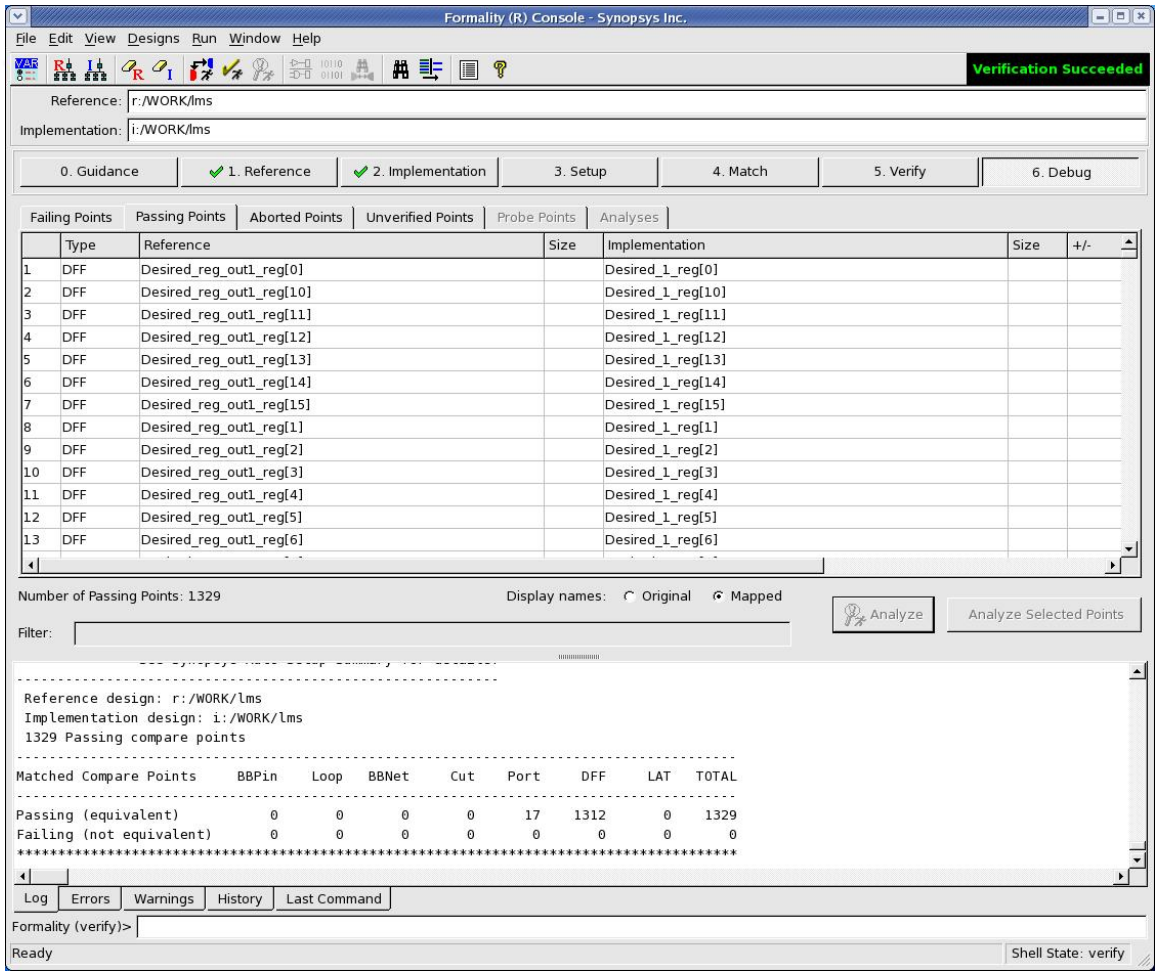

Figure 3.19. Formality window for LMS filter.

3.3.3. AES

Encryption is defined as the transformation of raw data to unintelligible data via an algorithm, which is named as cipher. AES is selected by the United States government for usage by the agencies and the Federal departments to secure sensitive

data. AES is an encryption standard, specifications of which have been published by The National Institute of Standards and Technology (NIST) [82].

The AES algorithm utilizes the same secret key for both encryption and decryption, which makes it a symmetric cipher. Furthermore, the AES algorithm processes groups of bits (blocks) of fixed-length, and the same transformations are applied to each block, which make AES a block cipher. The AES cipher operates on 128 bits and uses cipher keys with 128, 192 or 256 bits in length. All the AES cipher transformations are performed on a two dimensional 4x4 array of bytes, namely state. AES cipher performs a set of four transformations ($SubBytes$, $ShiftRows$, $MixColumns$, and $AddRoundKey$) on the array for certain number of rounds depending on the cipher key length. For our case, we chose cipher key of 128 bits, which requires 10 rounds of transformation. First, $AddRoundKey$ transformation is applied, then in the first 9 rounds, $SubBytes$, $ShiftRows$, $MixColumns$, and $AddRoundKey$ transformations are applied, respectively. In the last round, $SubBytes$, $ShiftRows$, and $AddRoundKey$ transformations are applied [14]. Ciphertext is the result of the above-mentioned cipher operations performed on the state.

Each encryption phase of the AES algorithm necessitates round keys of four 32-bit words. Considering the starting set of keys necessary for the first AddRoundKey transformation, $4 * (Nr + 1)$ (Nr:Number of Rounds) round keys are required in total. The cipher keys are utilized to generate all the round keys. In our case, we have chosen AES128, where the number of rounds is 10, the key length is four (meaning 128 bits), and the block size is four words (meaning 128 bits). The pseudocode for the AES cipher is shown in Figure 3.20.

To briefly explain cipher transformations:

SubBytes Transformation: This transformation is a byte replacement operation performed on single bytes of state, using a substitution table called S-box. This table is constructed by taking the multiplicative inverse in the finite Galois Field(GF) (2^8)

```
Cipher(byte PlainText[4*Nb], byte CipherText[4*Nb], word w[Nb*(Nr+1)])
begin
   byte state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])

   for round = 1 step 1 to Nr-1
      SubBytes(state)
      ShiftRows(state)
      MixColumns(state)
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

SubBytes(state)
ShiftRows(state)
AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

out = state

end
```

Figure 3.20. Pseudocode for AES.

of the byte and then applying the following transformation in Equation 3.1.

$$b_i' = b_i \oplus b_{(i+4)mod8} \oplus b_{(i+5)mod8} \oplus b_{(i+6)mod8} \oplus b_{(i+7)mod8} \oplus c_i \tag{3.1}$$

That transformation can be visualized as in Figure 3.21.

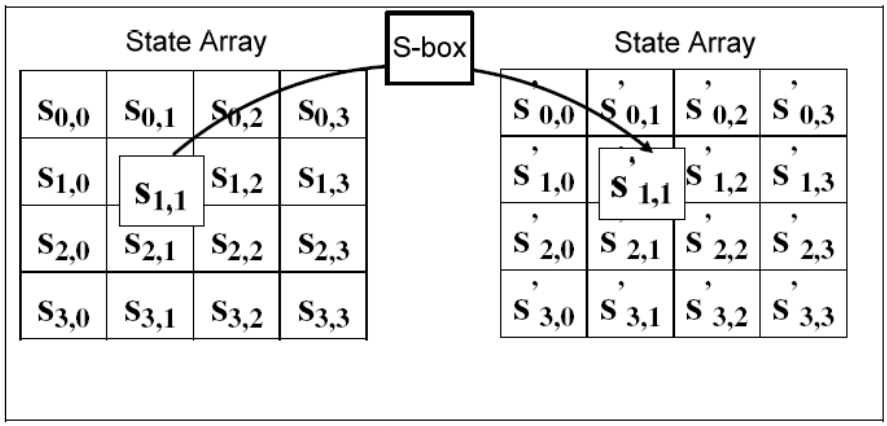

Figure 3.21. Illustration for subbytes transformation.

Shift Rows Transformation: This transformation cyclically shifts the last three rows of the state by different offsets. The first row is left unchanged in this transformation. Each byte of the second row is shifted one position to the left. The third and fourth rows are shifted left by two and three positions, respectively. The transformation can be visualized as in Figure 3.22.

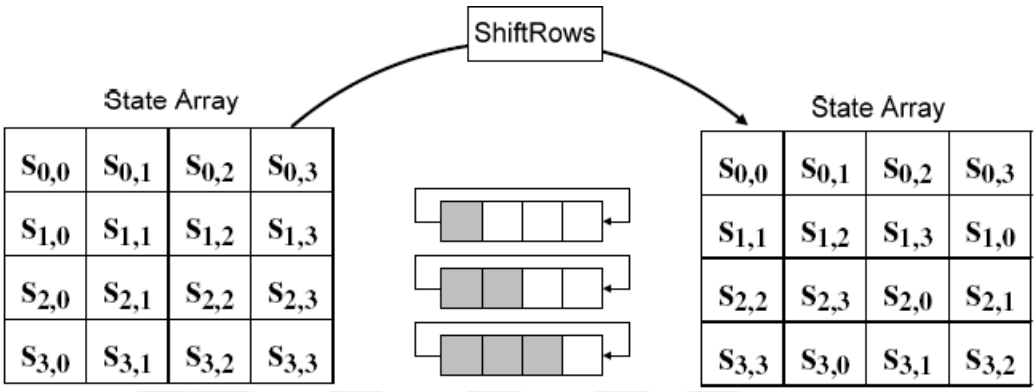

Figure 3.22. Illustration for shift rows transformation.

Mix Columns Transformation: This transformation performs multiplication operation on each columns of the state. In this operation, each column is considered as a polynomial of four terms in the finite field $GF(2^8)$. Each column is multiplied modulo x^4+1 over the $GF(2^8)$ with a predetermined polynomial function $a(x) = 03x^3$ + $01x^2$ + 01x + 02. The transformation can be represented by a matrix multiplication as shown in Equation 3.2.

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{0,c} \\ s_{0,c} \\ s_{0,c} \end{bmatrix} \tag{3.2}$$

Add Round Key Transformation: In this transformation, the round key values are added to the state by means of a simple Exclusive Or (XOR) operation. Each round key consists of $KeyExpansion$ routine generated Nb words (Block size, which is four in our case). The key expansion algorithm is shown in Figure 3.23.

```
KeyExpansion(byte CipherKey[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
word temp

i = 0
while (i < Nk)
w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
i = i+1
end while

i = Nk
while (i < Nb * (Nr+1)]
temp = w[i-1]

if (i mod Nk = 0)
temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
else if (Nk > 6 and i mod Nk = 4)
temp = SubWord(temp)
end if

w[i] = w[i-Nk] xor temp
i = i + 1
end while
end
```

Figure 3.23. Pseudocode for key expansion.

The manual HDL implementation of AES was obtained from [14] by rewriting SystemVerilog implementation as Verilog. The hierarchy of the implementation is shown in Figure 3.24. In addition to the AES required I/O's, *ld* and *done* are added to design to indicate the availability of desired input and output respectively.

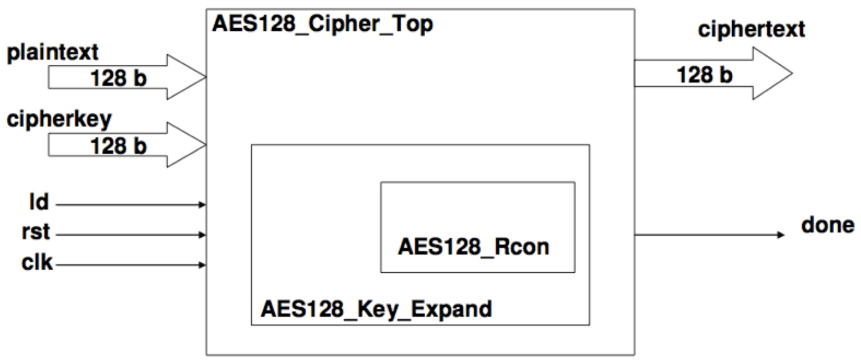

Figure 3.24. The hierarchy of the manual implementation for AES 128 in [14].

In the design, $AES128_Key_Expand$ module, which generates the keys, is utilized with the $AES128_cipher_top$ module. While the $AES128_cipher_top$ module performs an iteration of the encryption transformations on the array using the previously gener-

ated keys, the $AES128_Key_Expand$ produces the next round's keys. $AES128_Rcon$ calculates a parameter for $AES128_Key_Expand$ for key generation process.

In the adapted design, key and text inputs were separated to four words, since in Simulink at most 32 bits can be represented. Interfaces of $AES128_Key_Expand$ and $AES128_RCon$ are shown in Figures 3.25 and 3.26, respectively.

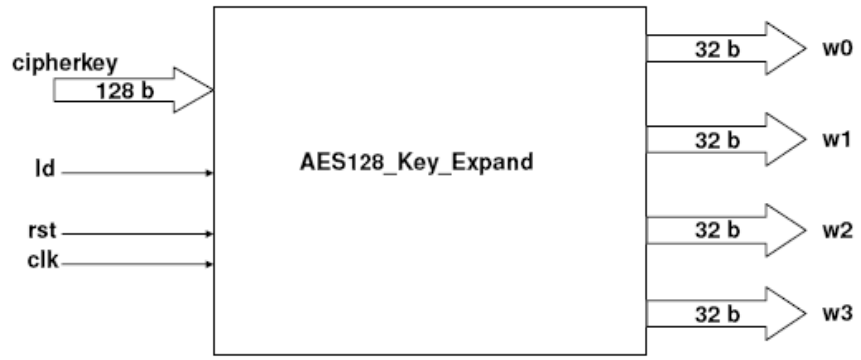

Figure 3.25. AES128_Key_Expand block diagram.

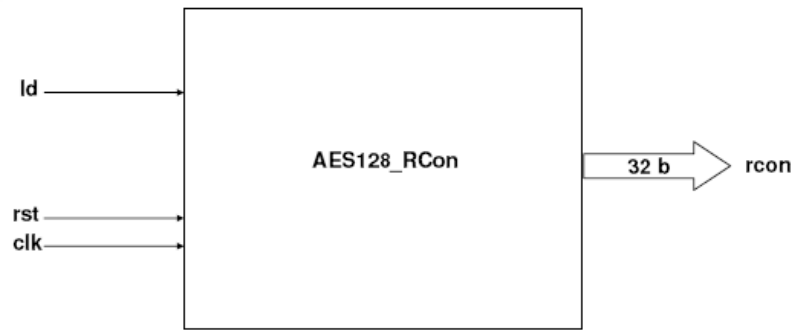

Figure 3.26. AES128_RCon block diagram.

For the Simulink model design, built-in Simulink models and S-functions were used. $Mix\ Columns$ transformation, which is a matrix multiplication operation, was implemented utilizing S-function builder, since it can be expressed more easily in software language. The remaining parts of the model were realized using built-in Simulink blocks.

Since there are four columns for our case of AES128, there are four $Mix\ Columns$ blocks in the RTL and also in Simulink. S-function builder block for single $Mix\ Columns$ transformation is shown in Figure 3.27.

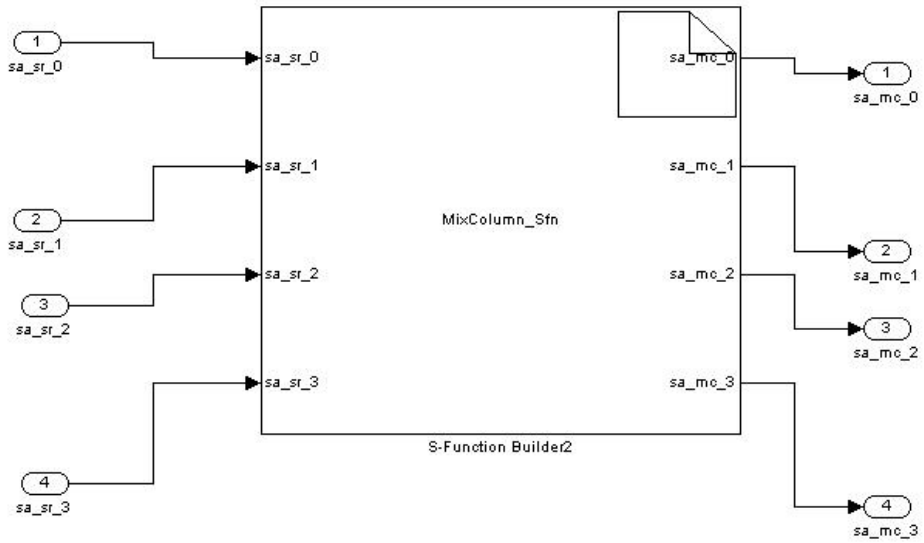

Figure 3.27. S-function builder for mix column transformation.

As it is seen from Figure 3.27, there are four inputs for four bytes (which are outputs of *ShiftRows* transformation) and there are four outputs for four bytes. In the S-function Builder, I/O's were defined as uint8 datatypes. And in the output pane of the builder, the functionality in Figure 3.28 was entered. Then, using Real Time Workshop, RTW S-function block in Figure 3.29 was generated.

```
uint8_T xtime_0,xtime_1,xtime_2,xtime_3;
uint8_T sa_sr_0_left_shift, sa_sr_1_left_shift, sa_sr_2_left_shift, sa_sr_3_left_shift;
uint8_T sa_sr_0_right_shift, sa_sr_1_right_shift, sa_sr_2_right_shift, sa_sr_3_right_shift;
uint8_T and_constant = 27;
uint8_T and_value_0,and_value_1,and_value_2,and_value_3;
sa_sr_0_left_shift = (*sa_sr_0)<<1;
sa_sr_1_left_shift = (*sa_sr_1)<<1;
sa_sr_2_left_shift = (*sa_sr_2)<<1;
sa_sr_3_left_shift = (*sa_sr_3)<<1;
sa_sr_0_right_shift = (*sa_sr_0)>>7;
sa_sr_1_right_shift = (*sa_sr_1)>>7;
sa_sr_2_right_shift = (*sa_sr_2)>>7;
sa_sr_3_right_shift = (*sa_sr_3)>>7;
if(sa_sr_0_right_shift == 0)
    and_value_0 = 0;
else
    and_value_0 = 255;
if(sa_sr_1_right_shift == 0)
    and_value_1 = 0;
else
    and_value_1 = 255;
if(sa_sr_2_right_shift == 0)
    and_value_2 = 0;
else
    and_value_2 = 255;
if(sa_sr_3_right_shift == 0)
    and_value_3 = 0;
else
    and_value_3 = 255;
xtime_0 = sa_sr_0_left_shift ^ (and_constant & and_value_0);
xtime_1 = sa_sr_1_left_shift ^ (and_constant & and_value_1);
xtime_2 = sa_sr_2_left_shift ^ (and_constant & and_value_2);
xtime_3 = sa_sr_3_left_shift ^ (and_constant & and_value_3);
*sa_mc_0 = xtime_0 ^ xtime_1 ^ (*sa_sr_1) ^ (*sa_sr_2) ^ (*sa_sr_3);
*sa_mc_1 = xtime_1 ^ xtime_2 ^ (*sa_sr_0) ^ (*sa_sr_2) ^ (*sa_sr_3);
*sa_mc_2 = xtime_2 ^ xtime_3 ^ (*sa_sr_0) ^ (*sa_sr_1) ^ (*sa_sr_3);
*sa_mc_3 = xtime_0 ^ xtime_3 ^ (*sa_sr_0) ^ (*sa_sr_1) ^ (*sa_sr_2);
```

Figure 3.28. S-function builder code for mix column transformation.

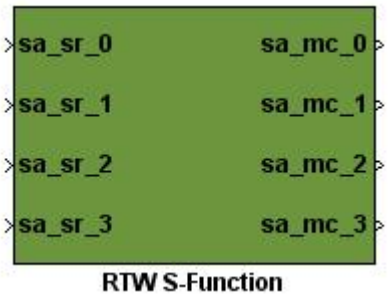

Figure 3.29. RTW S-function for mix columns.

As we have expected, HDL Coder was not able to produce RTL code for the S-function. Because of this, RTW-produced C files were to be used for RTL generation utilizing Catapult. As explained in the methodology section, the *Outputs_Wrapper* function was used to generate RTL file, the schematic of which is shown in Figure 3.30.

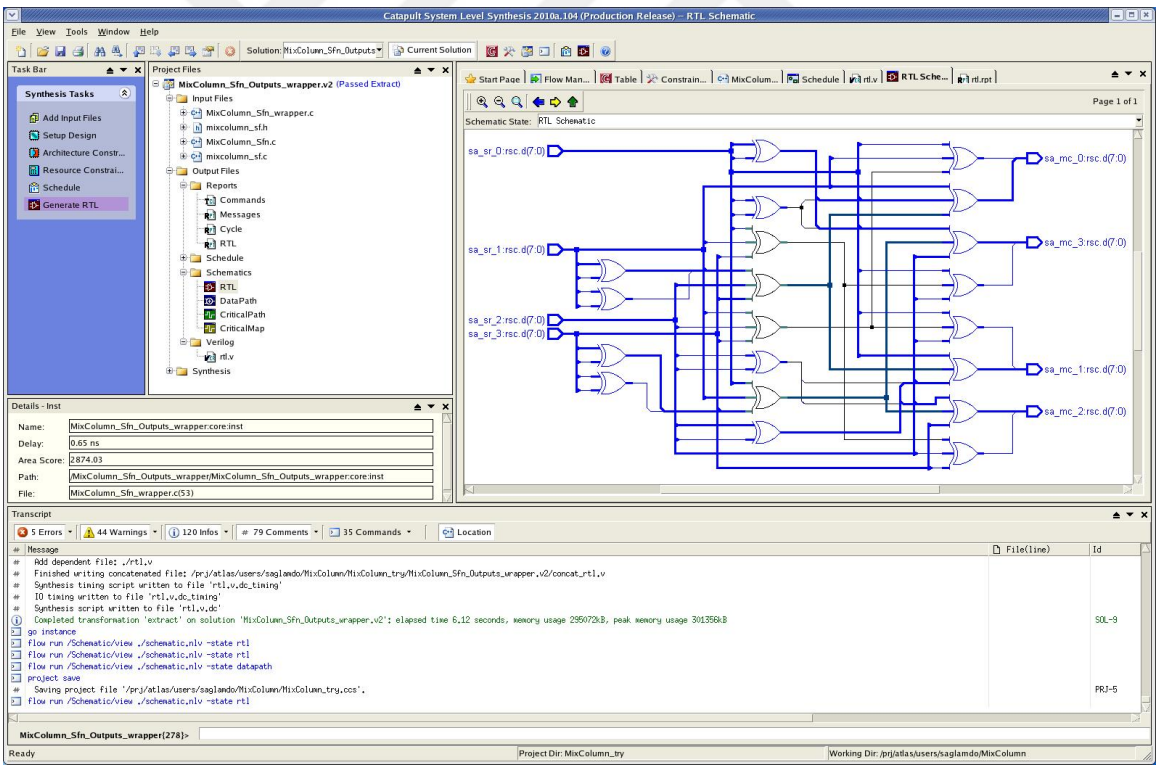

Figure 3.30. Schematic for Catapult output for mix column transformation.

At the last step of our flow, Formality was used to check the equivalence of the Catapult-generated RTL and the manual RTL for *Mix Columns* transformation. Formality verified that two implementations were equivalent. The Formality window for equivalency check is shown in Figure 3.31.

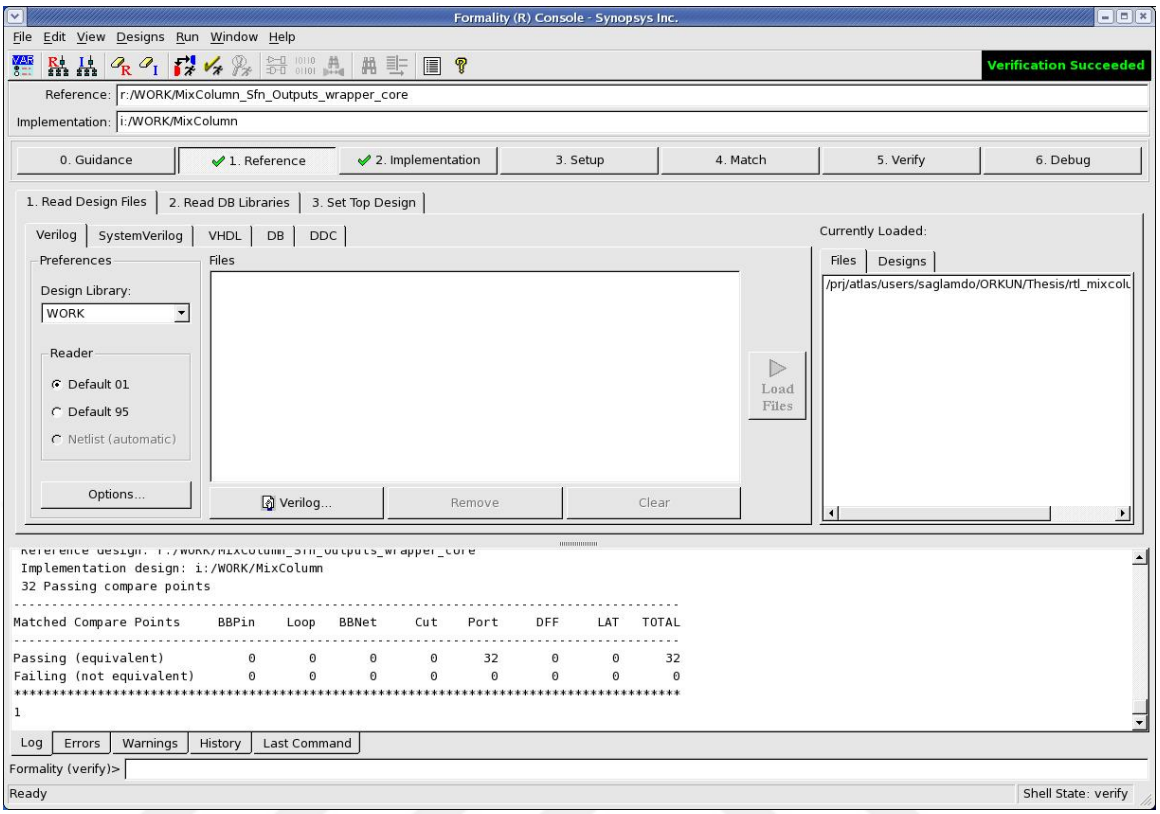

Figure 3.31. Formality window for mix column transformation.

If the verification of the full AES implementation is considered, the hand-written RTL description was ready. Also, the Catapult-generated RTL for user-defined implementation of *Mix Columns* transformation was verified to be equivalent with the manual implementation. However, the Simulink model of AES was missing. There was no implementation of Simulink model for AES using generic Simulink blocks and S-function blocks in the literature. We had to develop a Simulink model for AES as a novelty to literature. Adapting the same hierarchy with the manual RTL, a Simulink model was designed. The top level block depicting the interface is shown in Figure 3.32, whereas Figure 3.33 shows the block diagram of the top level AES Simulink model.

As it can be remembered from previous explanations $AES128_key_expand$ and $AES128_rcon$ were also implemented for Simulink model. They are as shown in Figures 3.34 and 3.35, respectively.

The next step for our flow application to AES was to generate RTL description for AES Simulink model. In the RTL file generation in HDL Coder, we were making

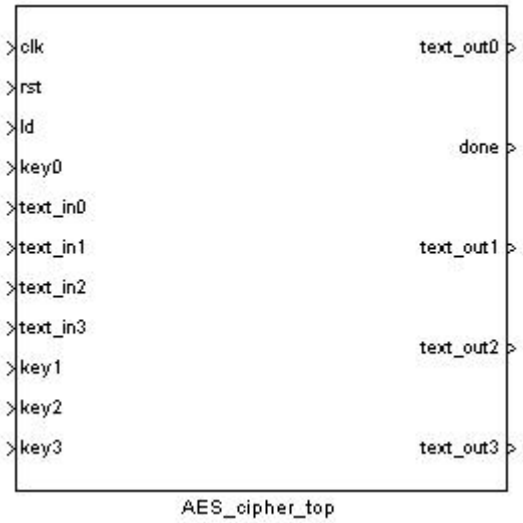

Figure 3.32. AES cipher top showing interface.

use of HDL Workflow Advisor. HDL Workflow Advisor has a GUI, which integrates the facilities of HDL Coder such as setting the target device, checking of model's validity for RTL generation and setting parameters of choice for the RTL to be generated (Like reset type, Verilog/VHDL selection) etc. A sample snapshot of HDL Workflow Advisor can be seen in Figure 3.36. We utilized the checking block compatibility facility of HDL Workflow Advisor. HDL Workflow Advisor reported that *BusCreator/Selector*, *SwitchCase* and *Merge* blocks used in the model were not synthesizable by HDL Coder.

Switch Case and *Merge* blocks were being utilized to implement the state machines in the AES design. We replaced them with HDL Coder synthesizable *Multiport Switches* such that for different "Cases", different subsystems provided the output. *Multiport Switches* were used to direct the correct input to output by means of its select input. After those modifications, we were able to generate RTL for AES, which was our first target. It should be noted that previously presented figures for the top level, *AES*128*_key_expand* and *AES*128*_rcon* are their last versions after these updates.

As the AES Simulink model was complex, functional verifications of both Simulink model and Manual RTL were necessary. We verified both of them after some iterations. Snapshots for each of those simulations are in Figures 3.37 and 3.38.

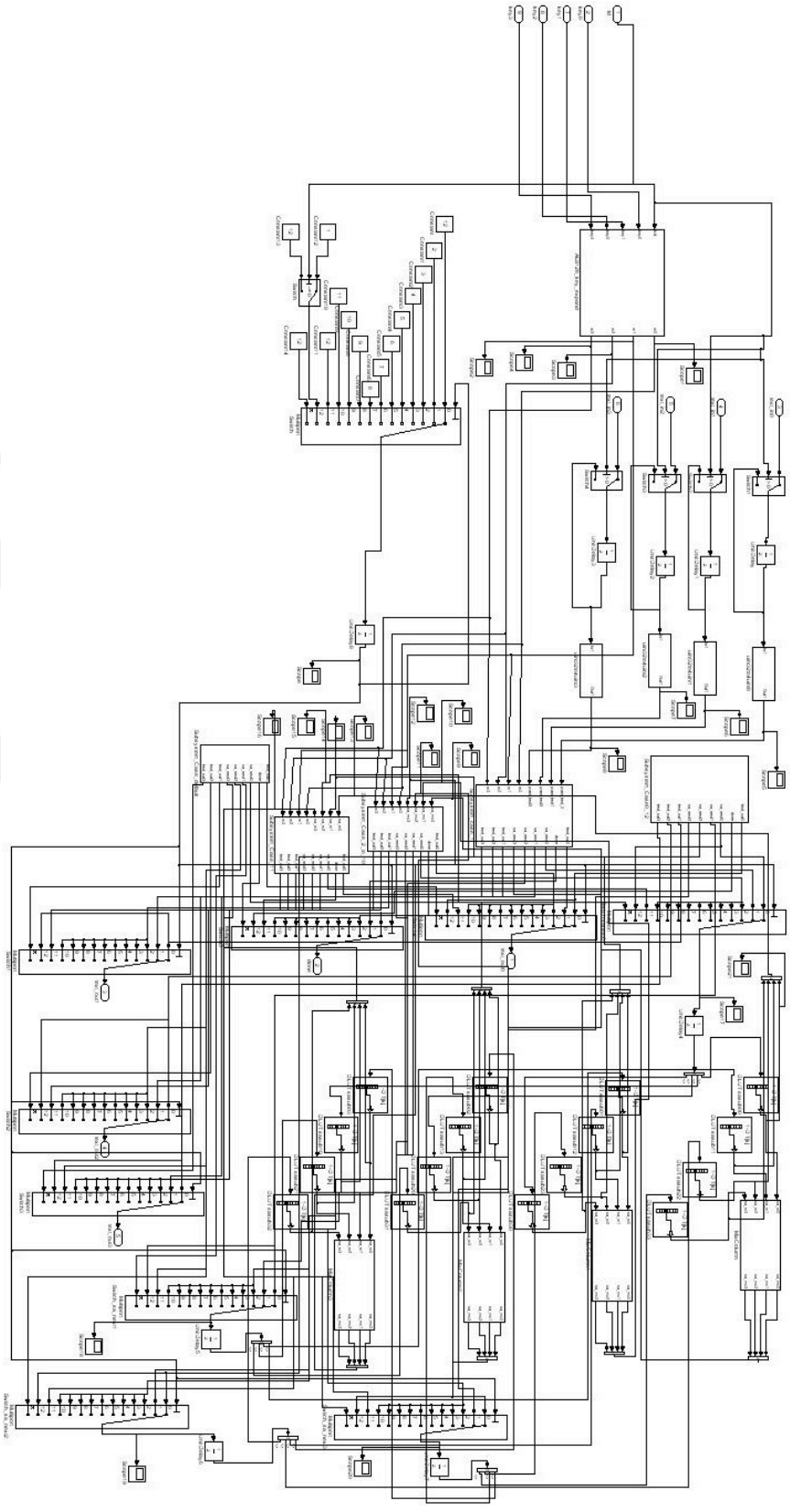

Figure 3.33. The AES cipher top.

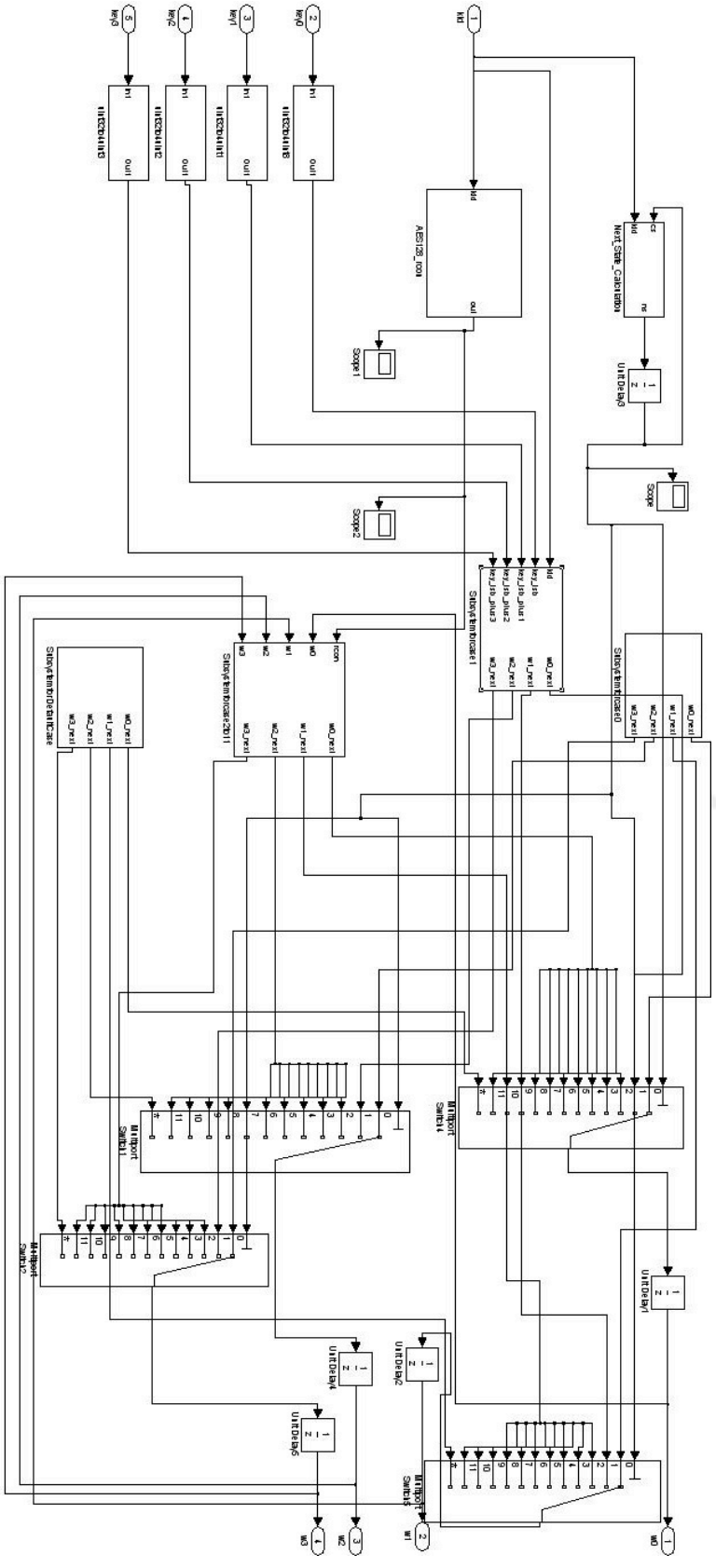

Figure 3.34. The AES key expand module of AES cipher top.

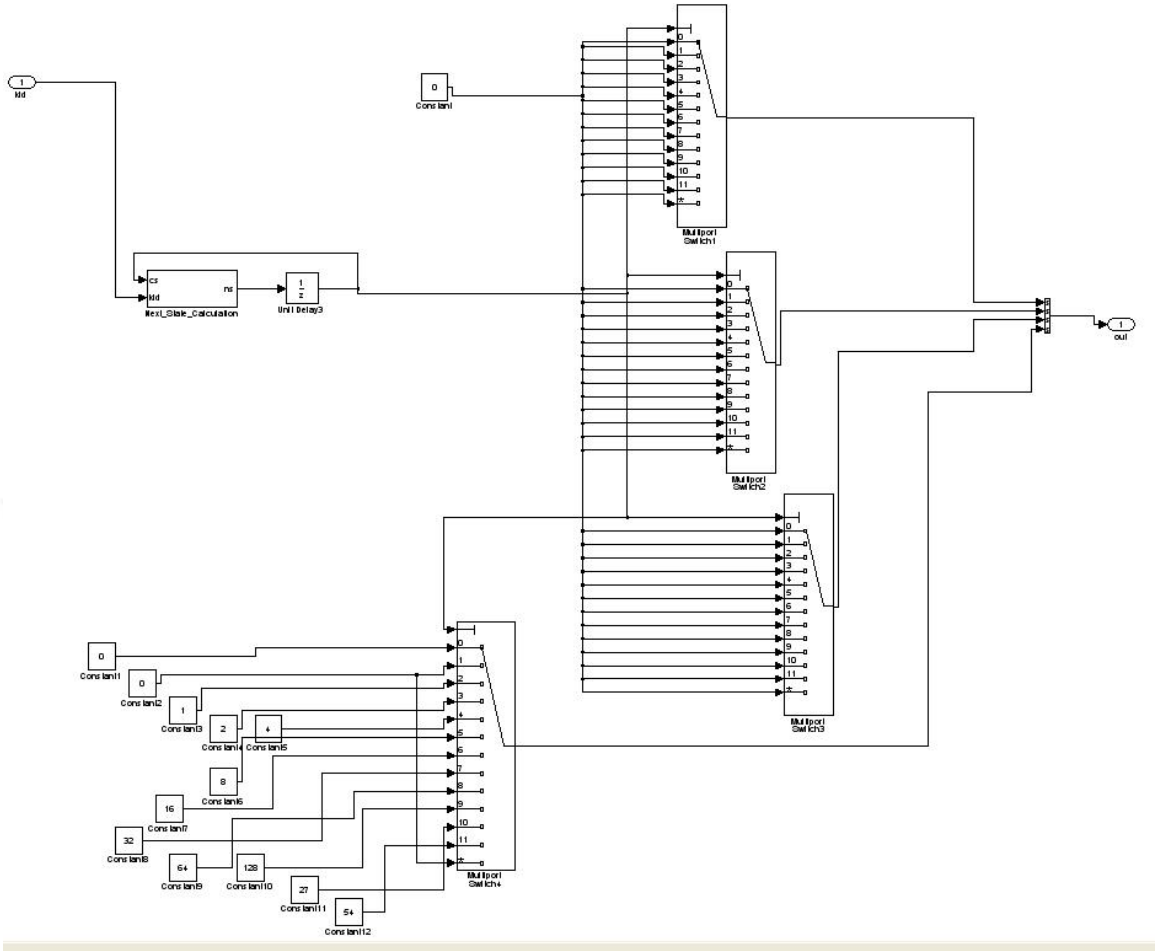

Figure 3.35. The AES 128 rcon module of AES key expand module.

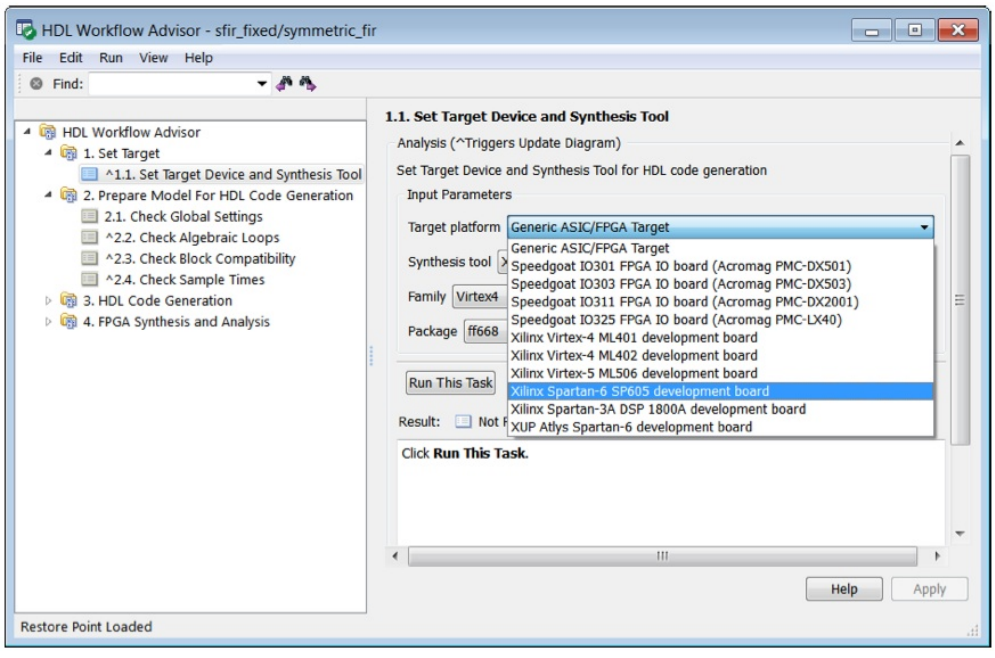

Figure 3.36. A snapshot of HDL workflow advisor.

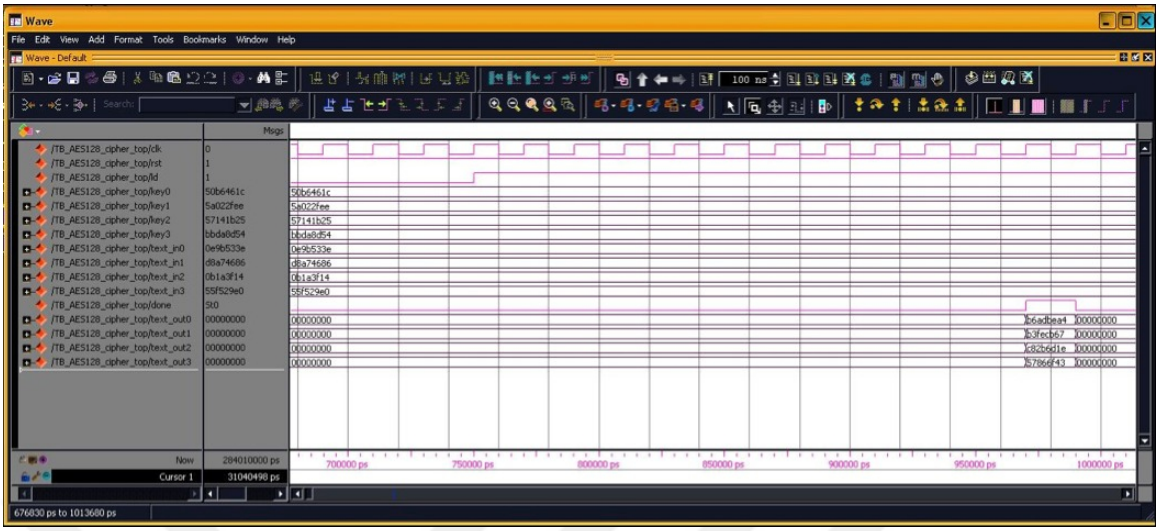

Figure 3.37. Modelsim simulation snapshot of manual RTL.

By following our proposed methodology, the HDL code for *MixColumn* transformation was generated by the application of RTW and Catapult C in succession. For Simulink design except *MixColumn* transformation, HDL codes were obtained by means of HDL Coder. Formality confirmed the formal equivalence of the Simulink model and the manual implementation. It should be noted that by means of our methodology, we found that *AES*128_*Rcon* was not implemented correctly in our manual RTL, while we were running Formality.

3.4. Conclusion

In this chapter, we aim to describe a methodology to check the equivalence between digital Simulink models and their HDL descriptions. In the first step, we validated our approach on a simple example. Then, we used several examples to verify our methodology including Advanced Encryption Standard.

Some metrics for the discussed experiments are given in Table 3.2. For HDL generation, we utilized a dual-core computer with 2.4 GHz clock speed, and 4 GB RAM. For equivalence check, we used an 8-CPU workstation with 16 GB RAM. As it can be seen from the table, our flow executes in a reasonable time. Considering the total durations needed for executing flow, which are the sums of HDL generation times

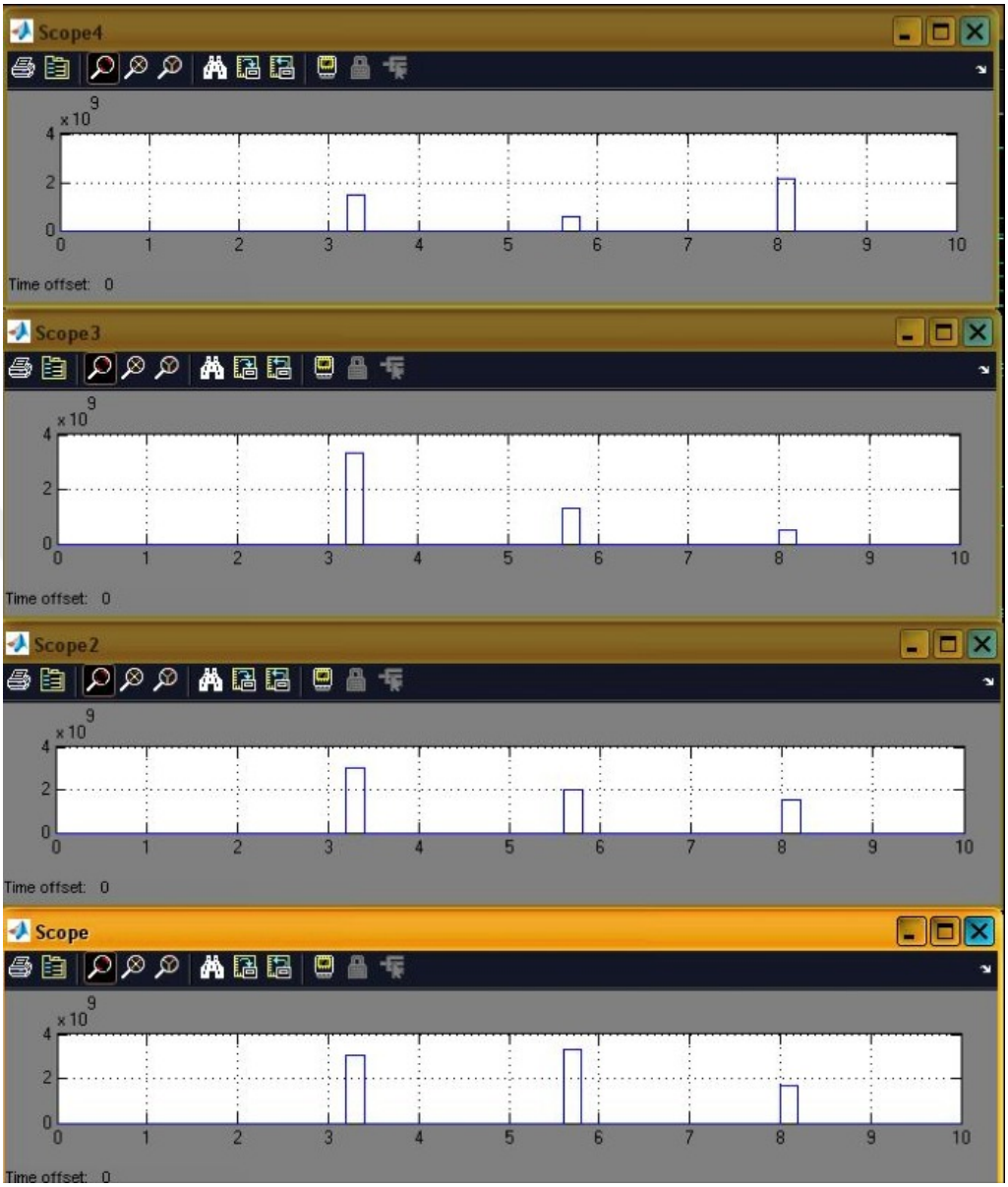

Figure 3.38. Simulink simulation of AES model.

Table 3.2. Some metrics for the experiments.

Parameter	Conf FIR	LMS Filter	AES
# registers	1155	1312	408
HDL gen time (sec)	13.1	36.1	27.5
# RTL lines (Auto)	1424	6611	12138
# RTL lines (Manual)	882	552	1103
Eqv check time (sec)	100.3	180.9	47.4

and equivalence check times for each experiment, it can be said that those durations correlate with the number of registers. Furthermore, for each experiment, it is seen that the number of RTL lines generated automatically is greater than the number of RTL lines written manually, which is an expected result.

4. ANALOG APPROACH UTILIZING OPTIMIZER OF MATLAB

4.1. Introduction

In recent years, Model Based Design (MBD) has become an inevitable part of IC Design industry. Although model based design methodology is very beneficial, the high level model (specification) should not only be checked for its functionality, but also for its equivalence to its low level design counterpart (implementation). In digital IC design, CAD tools and methodologies for verification and equivalence checking are more mature than their analog counterparts. In analog equivalence checking, there are various approaches. One of those approaches assumes linearity of analog circuits and in that approach, equivalence is decided by comparison of linear transfer functions [29,30, 64,65]. Another approach is based on state space sampling. Non-linear transformations on the sample state spaces are done in order to make comparison possible [8–10]. As a formal approach, an equivalence checking methodology based on input stimuli generation covering the complete state space is proposed in [7]. All the above mentioned approaches are enhanced by researches to make them applicable to larger circuits, but they are all limited by the state space explosion problem. Those approaches and their limitations were discussed in Section 2.2.2 in detail.

Due to the state space explosion that limits the usage of formal equivalence checking methodologies as in [7], Singh *et al.* [12] present a hierarchical, optimization based semi-formal (simulation-based) equivalence checking methodology. This approach decides the equivalence between a behavioral model (Verilog AMS) and its electrical implementation over a constrained input space. The verification problem is formulated as an optimization problem, which searches for the maximum error between the behavioral output and the mapped output from electrical domain. The approach followed in [12] is scalable, since it is not affected from the state space explosion. However, Verilog AMS utilized in [12] for high level models does not provide much abstraction and

does not exploit the full advantage of model based design. On the other hand, Simulink is a widespread design environment for both digital and analog models considering the availability of numerous built-in functions and also the ability to integrate user-defined functions. Hence, an analog equivalence methodology for Simulink would be of great help, considering that there is already an equivalence checking methodology for digital designs modeled in Simulink [33].

We present an analog equivalence checking methodology for Simulink models and their circuit level implementations (SPICE) in this chapter. This methodology is also a simulation-based semi-formal approach as in [12]. The equivalence of high and low level designs is determined by comparing a set of predefined performance parameters measured during the simulation of both models. Our methodology investigates around the optimal point of equivalency to obtain a range of input parameters for both models, where the error percentage between the performance parameters of both models is less than a specified threshold. We demonstrate the validity of our approach on three designs, an inverter, an operational amplifier, and a buck converter, where our approach proves to be an efficient tool in equivalence checking of analog circuits.

4.2. Methodology

4.2.1. Building of the Methodology

Our analog equivalence checking methodology is shown in Figure 4.1. To perform equivalence checking we utilize a set of performance and input parameters for the high level (Simulink) and low level (SPICE) designs. *Performance parameters* capture the essential characteristics of designs during their simulation. Examples are propagation delay, slew rate, and output voltage. *Input parameters* characterize the state of the inputs to the design. Examples are rise time of the input voltage, input voltage value, and parasitic resistances. The goal of our methodology is to determine a range of input parameters under which the percentage mismatches between the performance parameters of the models are below a given threshold. We use a threshold because analog signals cannot be exactly the same. Hence, we can determine the region, where

both designs are equivalent to each other up to a tolerance. In order to calculate the range, we use a multi-objective optimizer, since the number of performance parameters is greater than one.

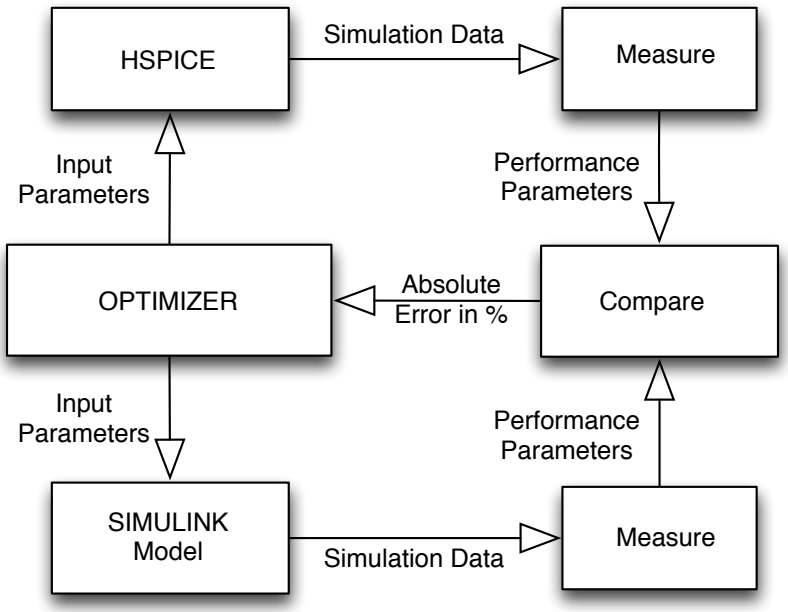

Figure 4.1. Analog equivalence checking methodology.

4.2.2. Implementation of the Methodology

We use Matlab environment in our methodology since it supports Simulink as the modeling platform for analog designs. Also, it has an optimization toolbox providing various multi-objective optimizer functions. Upon using Simulink, we generate analog high level models in two ways: by user-defined Matlab functions or by built-in libraries, specifically the Simscape library, which contains a broad range of electronic component models. We also use HSpice to simulate analog designs.

As the optimizer, *fminimax*, which is a function in the optimization toolbox of Matlab, is utilized. It solves multi-objective optimization problems and minimizes the worst-case (largest) values of a set of multivariable functions, starting at an initial estimate. In addition to the equality and inequality constraints, upper and lower bounds can be given for the output values. In our methodology, objective functions are the percentage errors between the performance parameters while the output vector contains the input parameters. An initial input parameter set is given to the optimizer,

together with the upper and lower bounds. More details on $fminimax$ function will be provided in Section 4.3.1 in the application of the proposed methodology to the inverter example.

In the first step, starting from the initial solution, the best (optimal) input parameter set within the upper and lower bounds, which makes the percentage errors in the performance parameters minimal, is obtained. Then, the second step of the algorithm proceeds depending on the condition of maximum percentage error in performance parameters for optimal input parameter set being lower than 10%. If that condition is satisfied, starting from the optimal input parameter set, which was produced by the first step, a lower and an upper bound is searched for each input parameter. In that search, the stopping criterion is to make the average of the percentage errors in the performance parameters less than 10%. This is done by keeping all of the input parameters - except the one for which bounds are searched - constant. It should be noted that the threshold 10% value has been chosen arbitrarily considering the fact that our experimental models are very accurate.

After completing the two steps, our methodology concludes with an input parameter set and also a lower and an upper threshold for each input parameter. These parameters are where the two implementations behave most similarly and the average of the error percentages of performance parameters is less than 10%, respectively.

Tables 4.1, 4.2 and 4.3 show the algorithms for extraction of performance parameter errors, finding the optimal input parameter set and finding the upper / lower threshold value for each input parameter set, respectively.

4.3. Experiments

We performed experiments on three designs. We used a dual-core computer with 2.4 GHz clock speed, and 4 GB RAM and run times for finding the optimal input parameter set and ranges for each input parameter were as follows: i. inverter: 152 mins, ii. operational amplifier: 133 mins, buck converter: 87 mins. Our methodology

Table 4.1. Algorithm for extraction of performance parameter error percentages for current input parameter values.

Inputs	Input parameter values
Step0	Run HSpice simulations and measure performance parameters for current input parameter values
Step1	Run Simulink simulations and measure performance parameters for current input parameter values
Step2	Measure error %'s of performance parameters
Outputs	Error % of performance parameters

Table 4.2. Algorithm for finding the optimal input parameter set.

Inputs	Initial values, Upper bounds and Lower bounds of input parameters
Step0	fminimax - Evaluate error %'s of performance parameters to be minimum (EXIT condition) and return with new input parameters (at first run - given initial values)
Step1	Assign current input parameter values for HSpice & Simulink runs
Step2	Run algorithm for extraction of performance parameter error %'s for current input parameter values
Step3	Return Step 0 with calculated error %'s of performance parameters
Outputs	Optimal values of input parameters, Final value of error %'s of performance parameters

completes in a reasonable time with good results leaving room for future performance improvements. Now, we describe the applications of our proposed methodology on three designs.

4.3.1. Inverter

As the first application of the methodology, we chose an inverter. For the netlist design, transistor width and length parameters were as follows:

$$W_p = 1.75\text{um} \qquad L_p = 0.18\text{um}$$
$$W_n = 0.75\text{um} \qquad L_n = 0.18\text{um}$$

Table 4.3. Algorithm for finding the upper/lower threshold value for each input parameter.

Inputs	Optimal values of input parameters, Upper or Lower bound for the input parameter of interest
Step0	Check maximum of error %'s of performance parameters being smaller than 10%, if not terminate, else continue with step 1
Step1	fminimax - Evaluate average of error %'s of performance parameters to be as near as possible to 5% (EXIT condition) and return with new input parameter value
Step2	Assign current input parameter value for HSpice and Simulink runs
Step3	Run algorithm for extraction of performance parameter error %'s for current input parameter value
Step4	Return Step 1 with calculated error %'s of performance parameters
Outputs	Upper or Lower threshold value for input parameter

In designing the Simulink model, parameters can be either process related or design oriented. Process parameters like k_p, k_n etc. should be the same for both system design and low level design, so it is not necessary to consider them during high level modeling. Design oriented parameters like transistor aspect ratios, namely $(W/L)_n$ and $(W/L)_p$ affect the behavior of inverter and should be known in high level modeling also. In case we know about the process parameters and the $(W/L)_p$, $(W/L)_n$, general parameters, namely the on-resistances for PMOS (R_p) and NMOS (R_n), and the internal capacitance (C_{int}), which determine the operation of inverter can be calculated. Considering these, we decided to model the inverter utilizing R_n, R_p and C_{int} as in Figure 4.2.

In our inverter model, V_m is the voltage, where $V_{in} = V_{out}$. V_m can be also calculated using R_p, R_n and V_{dd} as $V_{dd}*(R_n/(R_n + R_p))$. If the input voltage becomes greater than V_m, switching of the inverter will occur such that the voltage on the C_{int} and the parallel load capacitor C_{load} will be discharged over R_n. For voltages below V_m, C_{int} and C_{load} will be charged over R_p. Charging and discharging processes will be modeled as exponential, just as it is in circuit theory.

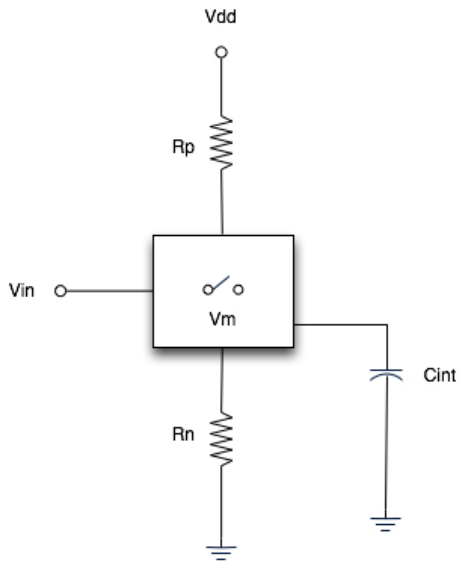

Figure 4.2. Simulink inverter model.

Internal design parameters, R_p, R_n and C_{int}, which are static, were extracted from transient analyses of the circuit. Giving pulses with zero rise and fall times as input voltage, rise and fall times of the output voltage of the inverter were measured without a capacitive load and with a capacitive load. All in all, we had the following equations in hand:

$$t_{rise}(V_{in}) = 0 \& C_{load} = 0 \qquad t_{rise}(V_{out}) = 2.2 R_p C_{int} \tag{4.1}$$

$$t_{rise}(V_{in}) = 0 \& C_{load} = X \qquad t_{rise}(V_{out}) = 2.2 R_p (C_{int} + X) \tag{4.2}$$

$$t_{fall}(V_{in}) = 0 \& C_{load} = 0 \qquad t_{fall}(V_{out}) = 2.2 R_n C_{int} \tag{4.3}$$

$$t_{fall}(V_{in}) = 0 \& C_{load} = X \qquad t_{fall}(V_{out}) = 2.2 R_n (C_{int} + X) \tag{4.4}$$

With subtraction of Equation 4.1 from Equation 4.2, R_p is obtained. Substituting the R_p value, C_{int} is found. Using Equation 4.3 and Equation 4.4 again, we obtain R_n.

Following this approach, we did the simulation on the inverter netlist and obtained the parameters as follows:

R_p= 1618 Ohms R_n=1550 Ohms C_{int}= 1.838 fF

Then, the Simulink model was designed manually using Embedded Matlab function considering also an additional load capacitance (C_{load}) as in Figure 4.3.

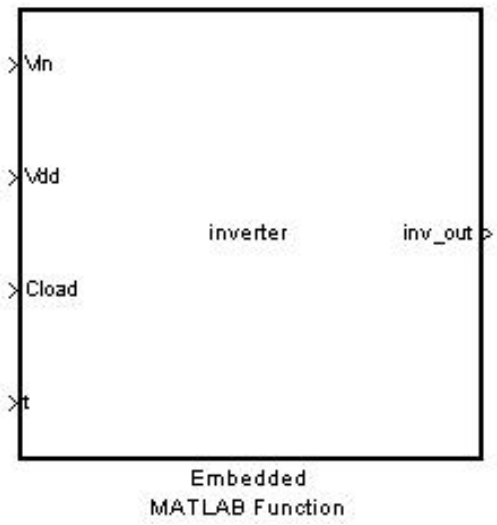

Figure 4.3. Simulink inverter model block.

For testing the methodology, both implementations were put in a configuration. In that configuration, a pulse waveform is applied to the input and there is a load capacitance at the output as shown in Figure 4.4. For application of the pulse waveform, another Embedded Matlab Function generating PULSE waveform was designed as seen in Figure 4.4.

The following parameters were determined to characterize the performance: rise ($Rise_T$) / fall ($Fall_T$) times, maximum (Max_V) / minimum output voltages (Min_V) and propagation delay low to high ($Prop_lh$) / high to low ($Prop_hl$) / mean ($Prop_M$). The input parameters that modify the design state are rise time (t_r) and fall time (t_f) of the pulse waveform and load capacitance (C_{load}). For measurement of the performance parameters, definitions from the literature were used [83]. The only modification made for the implementation purpose was to put a safe guard of 0.5% around the measure-

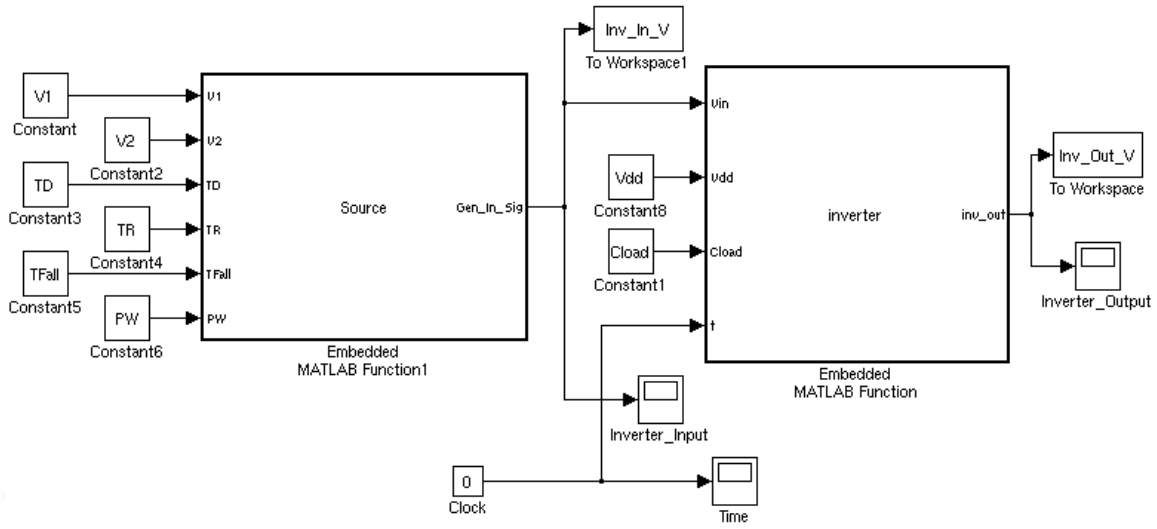

Figure 4.4. Simulink inverter model with input generator.

ment ratio, which makes the 90% measurement point an interval between 90.5% and 89.5%. This was done in order to ensure the existence of a simulation data point near 90%.

After obtaining all the parameters, the last step is the comparison of Simulink and Hspice parameter measurements. For all the parameters except the minimum output voltage, the Simulink parameter was kept as the reference output and the absolute error between the two parameters was scaled accordingly and converted to a percentage. For the measurement of minimum output voltage, since the values are small and can take the value of zero (most probably for Simulink), absolute error is referenced to V_{dd}.

To complete the construction of the methodology, a multi-objective optimization problem, which maximizes the error in the parameters was to be devised, as it was done in [12]. To that end, we utilized $fminimax()$ function of the Optimization toolbox of Matlab in our framework. $fminimax()$ function provides inputs to Simulink and HSpice in our methodology. It performs multi-objective optimization and finds the minimum x value that maximizes the objective functions as in Equation 4.5.

$$min_x\, max_i\ \ F_i(x)\ such\ that \begin{cases} c(x) \leq 0 \\ c_{eq}(x) = 0 \\ A \cdot x \leq b \\ A_{eq} \cdot x = b_{eq} \\ lb \leq x \leq ub \end{cases} \tag{4.5}$$

Matrices and vectors in Equation 4.5 are detailed in Table 4.4.

Table 4.4. Matrices and vectors in the definition of $fminimax()$ function.

x - input parameter vector	F - objective function vector
c - inequality constraint vector	c_{eq} - equality constraint vector
A - inequality matrix	A_{eq} - equality matrix
b - inequality vector	b_{eq} - equality vector
lb - lower bound vector	ub - upper bound vector

By utilizing $fminimax()$ we were able to connect the optimizer to our framework. Following that, we devised an algorithm to find the optimal inverter and ranges for each input parameter. In the first step of the algorithm, we obtain the best matching input parameter set. The best matching input parameter set signals the optimum inverter, where the behaviors of the Simulink model and the transistor netlist are as similar as possible. In the second step of the algorithm, we obtain ranges for being 5% mean equivalent for each input parameter. The pseudocode of the algorithm is summarized below:

- Run optimization to find the best input parameter set to have the least error between implementations
- In case maximum percentage deviation is smaller than 10%
- For each input parameter
- Find lower threshold for a mean of performance parameters' error of 5%
- Find upper threshold for a mean of performance parameters' error of 5%

At the last stage, the optimizer was integrated into the flow and following that, a sample run was done utilizing the algorithm explained in the previous part with following inputs: Start point $t_r = 50$ ps, $t_f = 50$ ps, $c_l = 60$ fF, where the lower bound was [20, 20, 20], and the upper bound was [100, 100, 100] for t_r, t_f, and c_l, respectively. Results are shown in Figure 4.5.

Input Parameters

$t_r = 20.0495$ $t_f = 20$ $c_l = 68.5631$

% Errors in Performance Parameters

$Rise_T = 0.67$ $Fall_T = 0.89$ $Min_V = 1.42$ $Max_V = 2.57$

$Prop_lh = 5.81$ $Prop_M = 0.27$ $Prop_hl = 5.97$

Ranges for Input Parameters

$[t_{r(min)}, t_{r(opt)}, t_{r(max)}] = [8.5001, 20.0495, 100]$

$[t_{f(min)}, t_{f(opt)}, t_{f(max)}] = [1, 20, 92.0225]$

$[c_{l(min)}, c_{l(opt)}, c_{l(max)}] = [44.9708, 68.5631, 100]$

Figure 4.5. Results for the inverter.

Our methodology was able to provide an input parameter set, where most of the error percentages were smaller than 3%, and also ranges for input parameters.

4.3.2. Operational Amplifier (Opamp)

A rather more complex example was the opamp. An optimization based in-house tool in Matlab was utilized for the circuit design. Tool generated a netlist having the following specifications after providing performance targets to it: Bandwidth: 12929.74 kHz, Power: 67.182 uW, Gain: 78.407 dB, Rout: 30.657 kOhm, Area: 7.909 e-9um2. Resulting operational amplifier was a Miller Compensated Two Stage Operational Amplifier with nulling resistor, which is shown in Figure 4.6.

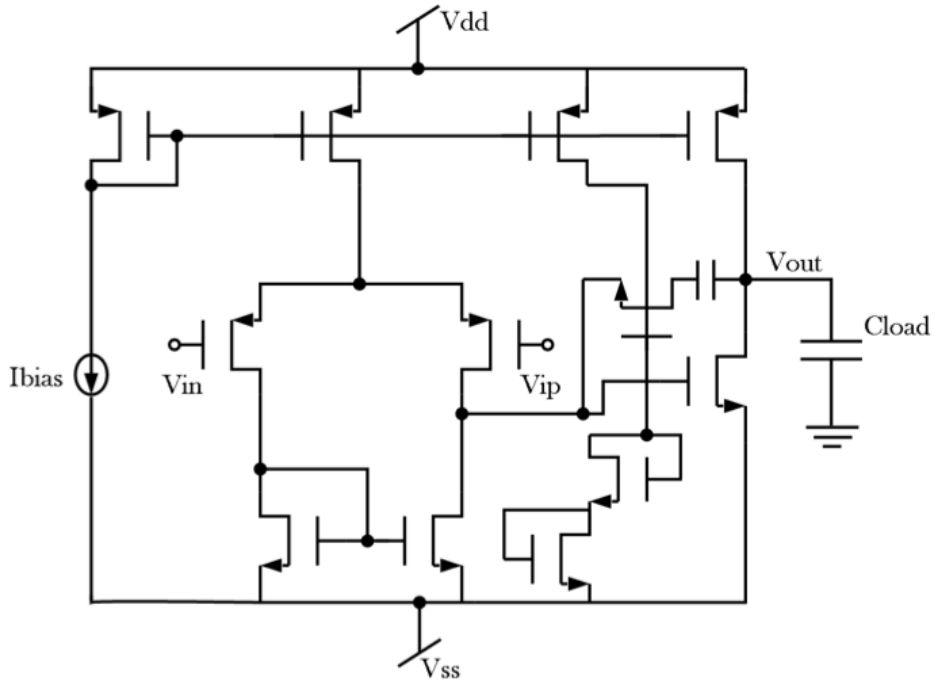

Figure 4.6. Circuit diagram for two stage operational transconductance amplifier [15].

For the Simulink model, the band-limited opamp in the Simscape library having the following parameters was utilized: *Gain*, R_{in}, R_{out}, V_{min}, V_{max}, V_{dot} (maximum slew rate), *Bandwidth* and *Initial Output Voltage* (V_0). The parameters and the functional definition for the band-limited opamp and GUI is shown in Figure 4.7.

Characterization of the opamp necessitates multiple configurations, since the design has both time and frequency domain characteristics. A pulse waveform between input ports with feedback and a load capacitance on the output allows analyzing the time domain characteristics. For the frequency domain, the pulse waveform was replaced with an AC sinusoidal waveform with adjustable frequency. The frequency domain configuration for Simulink is depicted in Figure 4.8. The circuit diagram for the frequency configuration is shown in Figure 4.9.

To be able to run Simulink simulations for models including SimScape components a solver is to be connected to a node in the Simulink model. In models having SimScape components signals are physical signals. In order to be able to see them as waveform or to be able to move the simulation data to workspace for processing, PS-

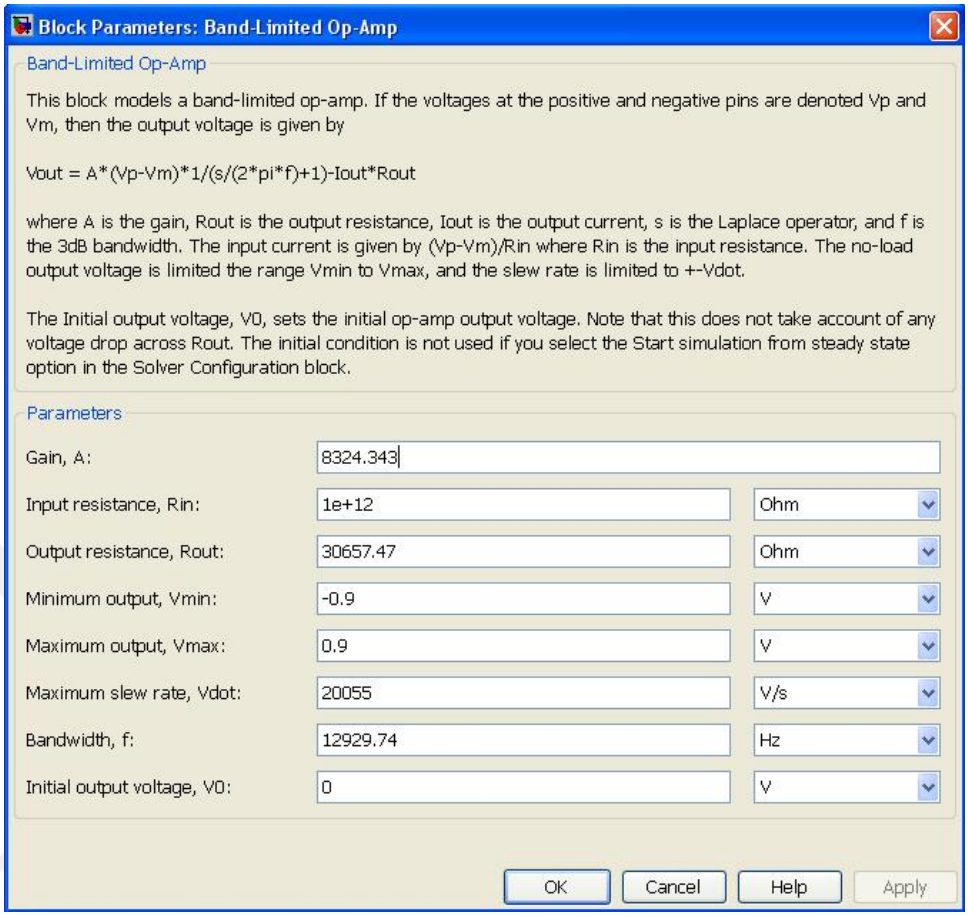

Figure 4.7. Band limited opamp from SimScape library.

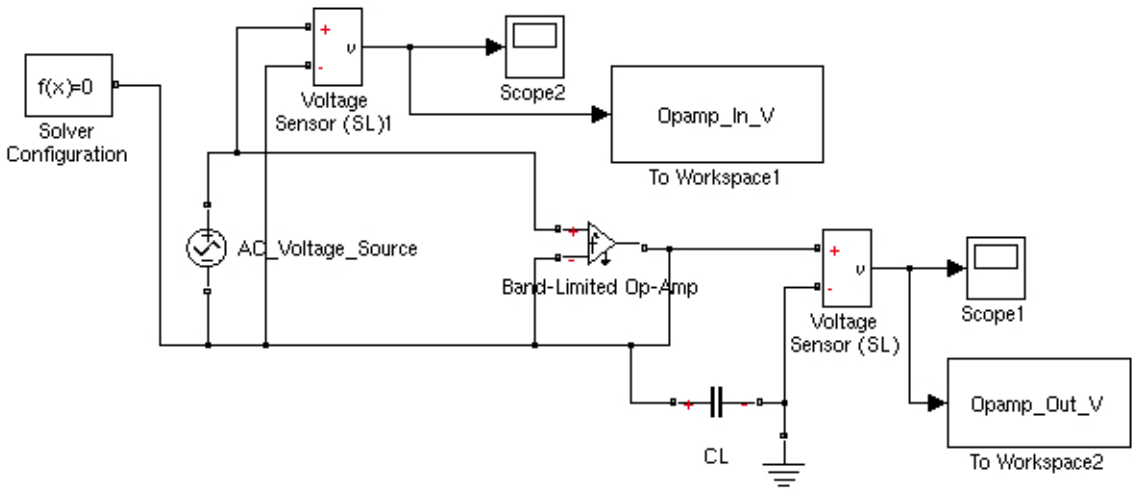

Figure 4.8. Operational amplifier configuration for frequency domain check.

Simulink converter blocks for conversion between physical signals and Simulink signals are to be used. Those blocks are internal in Voltage Sensor blocks, which also includes voltage sensor to measure voltage as shown in Figure 4.10.

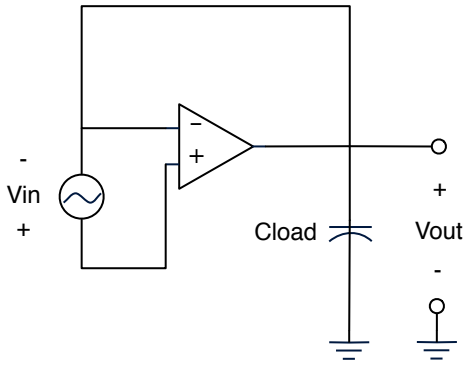

Figure 4.9. Circuit diagram for frequency domain check for operational amplifier.

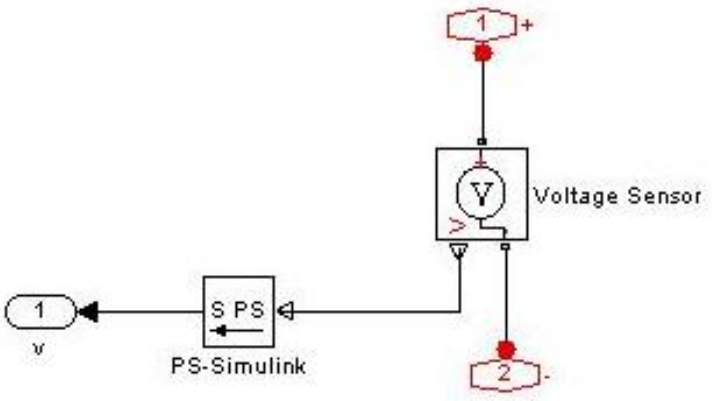

Figure 4.10. Voltage sensor block.

As the performance parameters in the time domain, maximum (Max_V) / minimum (Min_V) output voltages, rise (Rise_T) / fall (Fall_T) times and slew rate low to high (Slew_lh) / high to low (Slew_hl) are used. On the frequency domain, cut-off frequency (Cut_off) is used. For the input parameters, the load capacitance is used in both configurations, whereas rise and fall time of the pulse waveform are utilized in the time domain configuration.

For the opamp, there are two configurations, so the application of the methodology is a little bit different from the inverter case. After execution of each configuration as a single step of the optimization process, the performance parameters related to each configuration were extracted and at the end of each step, the percentage errors between the performance parameters were calculated.

Completing those steps, a sample run was done with following inputs: Start point was $t_r = 0.05$ ms, $t_f = 0.05$ ms, $c_l = 50$ pF, and lower bound was [0.03, 0.03, 10], upper bound was [0.3, 0.3, 200] for t_r, t_f and c_l, respectively. Results of the run are shown in Figure 4.11.

Input Parameters

$t_r = 0.1856$ $t_f = 0.3$ $c_l = 20.1513$

% Errors in Performance Parameters

$Slew_lh = 0.38$ $Slew_hl = 0.38$ $Cut_off = 9.08$ $Max_V = 9.10$

$Min_V = 8.24$ $Fall_T = 0.81$ $Rise_T = 0.81$

Ranges for Input Parameters

$[t_{r(min)}, t_{r(opt)}, t_{r(max)}] = [0.132, 0.1856, 0.2439]$

$[t_{f(min)}, t_{f(opt)}, t_{f(max)}] = [0.2042, 0.3, 0.32]$

$[c_{l(min)}, c_{l(opt)}, c_{l(max)}] = [20.1513, 20.1513, 54.1654]$

Figure 4.11. Results for the opamp.

Our methodology was able to provide an input parameter set, where error percentages were smaller than 10%. Also it was able to provide ranges for input parameters except the lower threshold for the load capacitance.

4.3.3. Buck Converter

For the last experiment, a buck converter, which is commonly used in power management IC's in order to step down the voltage, was chosen. In its basic form, buck converter contains two switches (usually a diode and a transistor) controlling the current through an inductor. The inductor is in series with a parallel connected capacitor and load resistance. The circuit diagram for the buck converter is shown in Figure 4.12.

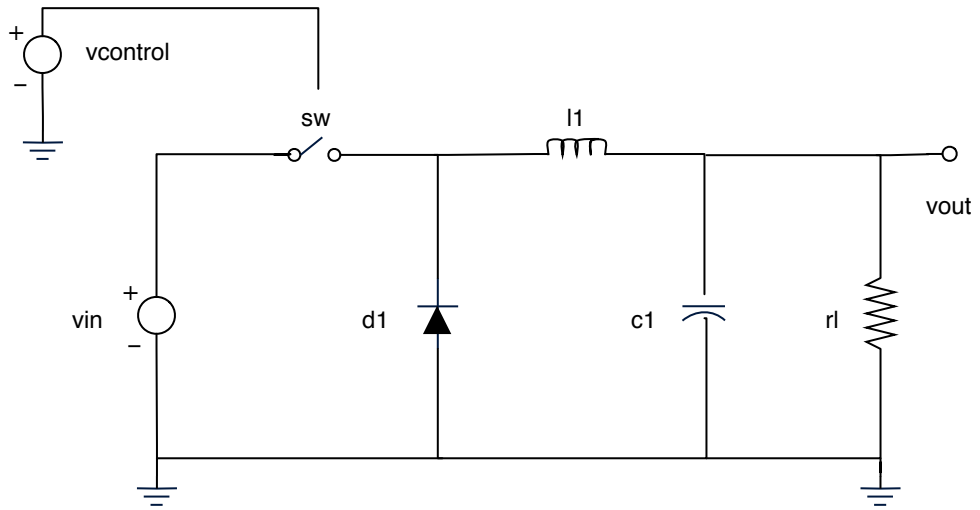

Figure 4.12. Circuit diagram for buck converter.

Regarding the continuity of the output current, there are two modes of operation: continuous or discontinuous. In the experiment, continuous mode will be exercised. In the idealized converter, where all the components are considered to be perfect, the ratio between output and input voltages is equal to the duty cycle of the pulse waveform controlling the switching of the transistor. For the idealized converter, the switch and the diode have zero voltage drop when on. Furthermore, no current flows when off and the inductor and the capacitor have zero series resistances. For the design of the Simulink and the circuit level implementations, the values of the inductor and capacitor were calculated and the switching frequency was chosen to be 3 MHz, as it is the case for the current buck converters in the industrial designs.

For the Simulink model, components in the Simscape library were utilized. Switch was used as the transistor and pulse waveform was utilized for switching. The Simulink model designed can be seen in Figure 4.13.

In the Simscape library, various parasitic characteristics were integrated to the component models like switch on, diode on, inductor series, and capacitor series resistances. This functionality makes it possible to analyze the effects of those parasitic components in the methodology. On the circuit level implementation, switch was built by means of a voltage controlled resistor having a resistance value varying according to the applied control voltage in a piecewise linear relationship. Series inductor and

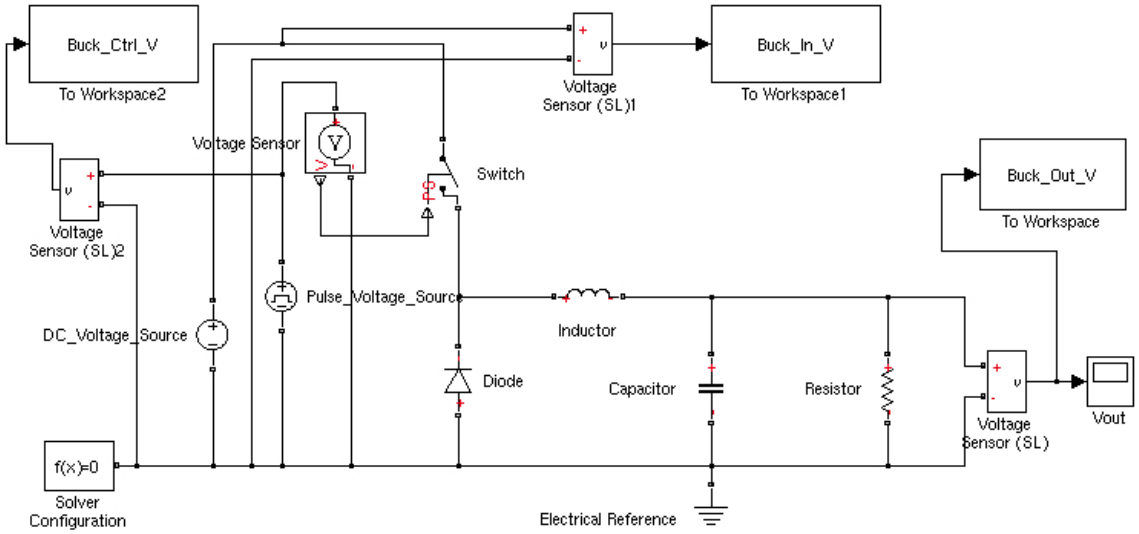

Figure 4.13. Simulink model of the buck converter.

capacitor resistances were added to the netlist for analysis.

As buck converter is a voltage regulating design, performance parameters have to depend on the characteristics of the output voltage. We use three performance parameters. Settling of voltage and spikes can be captured with the maximum output voltage (Max_V). The oscillations even after settling can be captured with the ripple voltage (Ripple_V). We also use the mean output voltage (Out_V). For input parameters, two sets of parameters were introduced. One set contained the major components affecting the functionality: input voltage (v_{in}), duty ratio of the switching pulse waveform (pw), and the load resistance (r_l). The second set included the parasitic resistances: switch on (r_{sw}), diode on, inductor series (r_{ind}), and capacitor series (r_{cap}) resistances. For the valid operation, switch and diode on-resistances were defined to be equal resulting in six input parameters. Diode and switch off resistances, and parallel conductances of the inductor and capacitor could be introduced also, but the aim was to include the most effective parasitics and to keep the number of parameters at a reasonable count they were omitted.

Following the completion of the steps of integrating the implementations to the methodology, a sample run was done with following inputs: Start point was pw = 166 ns, v_{in} = 5 V, r_{sw} = 0.1 Ohms, r_{ind} = 0.05 Ohms, r_{cap} = 0.01 Ohms and r_l =

10 Ohms, and in the same order, lower bound was [50, 3, 0.015, 0.01, 0.002, 1], upper bound was [300, 7, 2, 1, 0.5, 200]. Results are shown in Figure 4.14.

Input Parameters

$pw = 168.1152$ $v_{in} = 3.2674$ $r_{sw} = 0.1941$

$r_{cap} = 0.4215$ $r_l = 11.2811$ $r_{ind} = 1$

% Errors in Performance Parameters

$Out_V = 0.001$ $Ripple_V = 0.007$ $Max_V = 0.0065$

Ranges for Input Parameters

$[pw_{(min)}, pw_{(opt)}, pw_{(max)}] = [52.78, 168.12, 287.99]$

$[v_{in(min)}, v_{in(opt)}, v_{in(max)}] = [3.2674, 3.2674, 5.2874]$

$[r_{sw(min)}, r_{sw(opt)}, r_{sw(max)}] = [0.005, 0.1941, 1.6365]$

$[r_{cap(min)}, r_{cap(opt)}, r_{cap(max)}] = [0.421, 0.421, 0.421]$

$[r_{l(min)}, r_{l(opt)}, r_{l(max)}] = [11.2811, 11.2811, 130.1351]$

$[r_{ind(min)}, r_{ind(opt)}, r_{ind(max)}] = [1, 1, 1.02]$

Figure 4.14. Results for the buck converter.

Our methodology provided an input parameter set with very small error percentages. It also ended up with ranges for most of the parameters except the capacitor series resistance. That exception is due to the effect of capacitor series resistance on performance parameters being very minor.

4.4. Conclusion

In this chapter, a simulation-based analog equivalence checking methodology between high level Simulink models and their low level Spice counterparts was presented. The equivalence of high and low level designs is determined by comparing a set of predefined performance parameters measured during the simulation of both models.

The presented methodology investigates around the optimal point of equivalence to obtain a range of input parameters for both models, where the error percentage between the performance parameters of both models is less than a specified threshold. The validity of our approach was demonstrated on three designs, an inverter, an operational amplifier, and a buck converter, where our approach proves to be an efficient tool in equivalence checking of analog circuits.

5. ANALOG APPROACH WITH EVOLUTIONARY COMPUTATION

5.1. Introduction

Although model-based design is an indispensable part of the IC Design projects, in order to rely on the results obtained by model simulations, a metric is required to specify how reliable the model is at the point where the actual design is made. In that respect, some safe zones in terms of design parameters should be found, where model usage is reliable to a certain extent. So an analog equivalence checking methodology should be able to give the boundaries of the design parameter set. In Chapter 4, we presented an analog equivalence checking methodology, which is capable of finding the upper and lower bounds for each design parameter. Although it is valuable to have the upper and lower bounds, obtaining the whole equivalence boundary is more significant.

Inline with this need, we present an analog equivalence boundary search methodology for Simulink models and their circuit level implementations (HSpice) in this chapter. Our approach is also a simulation-based semi-formal approach as in [12] utilizing evolutionary computation. In literature, Monte Carlo analysis has been used for optimization purposes, because it offers a full coverage of search space by generating random points. On the other hand, a large number of simulation is required in Monte Carlo analysis, which results in excessive computation time, since our methodology requires both HSpice and Simulink simulations. Therefore, rather than using Monte Carlo analysis to find the boundary region, we prefer to utilize evolutionary computation to avoid redundant run time. We demonstrate that our approach reduces the necessary number of simulations when compared with Monte Carlo method. Similarly evolutionary computation is applied to different circuit optimization problems in literature successfully, as in [84] and to the best of our knowledge, there is no study, which applies evolutionary algorithms to equivalence checking other than our study.

The methodology explained in this chapter differs much from the one explained in the previous chapter [34], as Matlab built-in optimizer we utilized in the methodology in the previous chapter was replaced by evolutionary computation. In addition to that, the methodology in this chapter is capable of finding the equivalence boundary in the space defined by the parameters, whereas the methodology in the previous chapter [34] is capable of finding just the upper and the lower bounds for each parameter.

The chapter is organized as follows. In Section 5.2, we present the building of our methodology and the evolutionary computation approach we used in connection with boundary search. In Section 5.3, we present the applications of our methodology on three examples: Inverter, Operational Amplifier and Buck Converter and provide the results of those applications.

5.2. Methodology

5.2.1. Building of the Methodology

Our analog equivalence boundary search methodology is shown in Figure 5.1. To perform equivalence checking we utilize a set of performance and input parameters for the high level (Simulink) and low level (HSpice) designs. *Performance parameters* capture the essential characteristics of designs during their simulation. Some typical performance parameters are propagation delay, slew rate, and output voltage. *Input parameters* characterize the state of the inputs to the design, e.g., rise time of the input voltage, input voltage value, and parasitic resistances.

The goal of our methodology is to determine a boundary of input parameters under which the difference of the performance parameters of the models are below a given threshold defined as our success criteria. We use a threshold, because analog signals cannot exactly be the same. Hence, we can determine the region where both designs are equivalent to each other within a tolerance value.

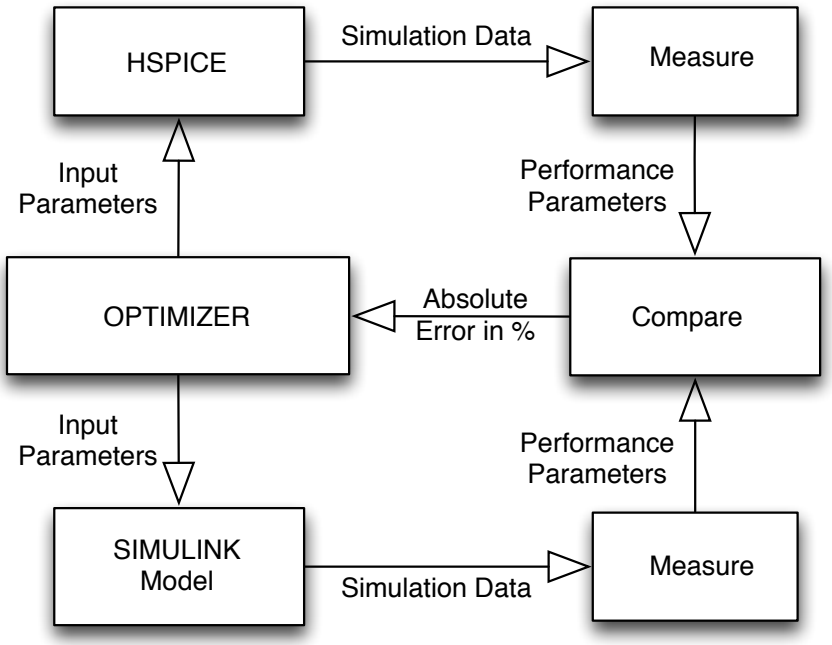

Figure 5.1. Analog equivalence boundary search methodology.

In order to determine the boundary, we use a multi-objective optimizer, which provides the values of input parameters necessary to simulate both HSpice and Simulink models. Then, for each simulation, we measure a set of performance parameters. Finally, the mismatches between the performance parameters for HSpice and Simulink results are measured in percentages and given as input to the optimizer.

As an alternative to using the optimizer to search the space for a boundary, we also investigate the Monte Carlo approach. This approach is a de-facto standard for search problems and it can definitely find the boundary, if a sufficient number of samples are taken, since variance decreases when the number of samples are increased. In our case, the number of samples is proportional to the number of simulations, which deteriorates the run time. Increase in run time makes a methodology inapplicable due to time-to-market concerns, similar to the case of state space explosion which we try to avoid. The optimizer should provide the input parameters in a smart way in order to reduce the number of iterations. To that end, we use an evolutionary computing algorithm that allows us to find the solution in smaller number of steps. Details on the optimizer and the evolutionary computation strategies it utilizes will be explained in the following section.

5.2.2. Proposed Approach

Matlab environment has been used in our methodology since it supports Simulink as the modeling platform for analog designs. In Simulink, we generated analog high level models in two ways: by user-defined Matlab functions or by built-in libraries, specifically the Simscape library, which contains a broad range of electronic component models. We also used HSpice to simulate low level analog designs. It is important to emphasize that these tools have been used only for validating the proposed approach; therefore the overall approach is independent of the design environment. As a result, any high level model and circuit simulator could be utilized in this methodology.

The primary goal of the proposed approach is to find input parameters for the Simulink model and HSpice netlist such that they remain within a boundary region that satisfies the minimum error percentage. In order to obtain the boundary, each input parameter found during optimization should be treated as equally important and independently. Thus, rather than using single-objective optimization, where the main idea is to converge to a certain area that corresponds to a pre-defined cost value, we utilize multi-objective optimization in the search algorithm since it is more suitable for obtaining Pareto fronts. For this purpose, we utilized a modified version of the multi-objective algorithm given in [32].

The flow diagram of the proposed approach is shown in Figure 5.2. Optimization starts with the initialization of an empty and variable-size archive and a fixed-size initial population, where the Sobol sequence is utilized. The initialization of the population by using Low Discrepancy Sequences (LDS) sets is advantageous since more homogeneously distributed initial population results in fewer generation number, as stated in [85]. In the following step, HSpice and Simulink simulations are carried out and resulting performances are compared. Based on the difference between measurements, infeasible sample elimination is performed, in which the individuals satisfying the user defined error percentage are selected. In the next step, a fitness value is assigned to each individual. The fitness assignment method proposed in [32] calculates the fitness value based on Pareto dominance, which is determined by comparing the

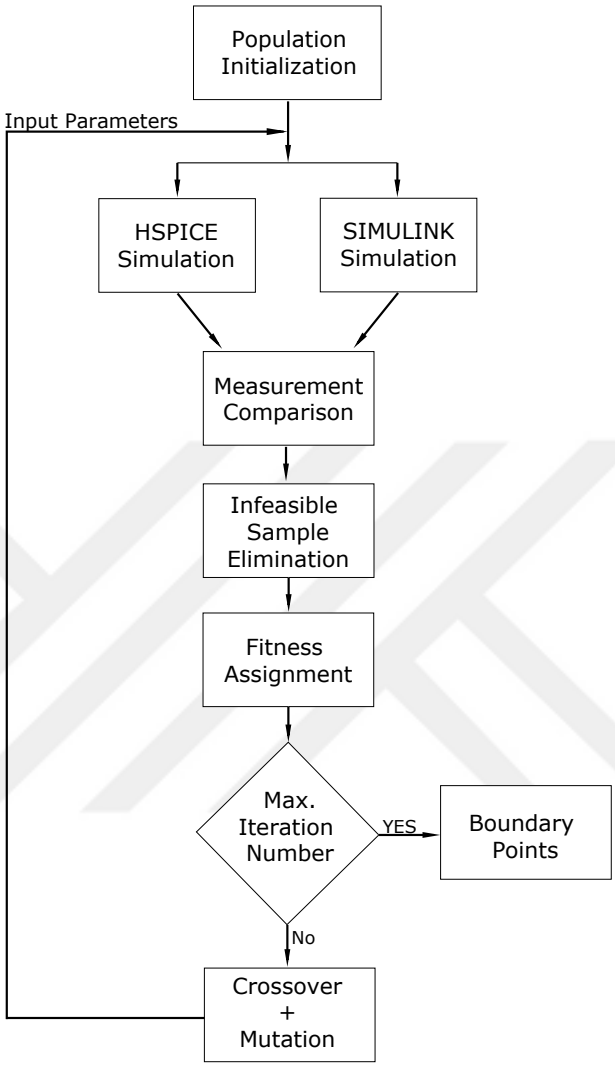

Figure 5.2. Flow diagram of the proposed approach.

design objectives of individuals from population and archive. The domination of an individual can be determined if and only if all design objectives of the individual are not smaller and at least one of them is greater than its competitor. However, in our case, input parameters that satisfy minimum error percentage constraint do not have to comply with Pareto dominance, since we are interested in finding a closed surface that contains both smaller and greater points from input parameters. As a result, we modified the fitness assignment method such that the individual residing on the boundary of its current population is assigned a lower fitness value. We used Matlab's built-in boundary function to obtain the required points, since this function outputs

boundary points for both convex and concave fronts. In this methodology, dominance criteria between two competing individuals is determined based on their location in the current population. If an individual is on the boundary, it dominates its competitor. This analogy is visually illustrated for a two dimensional case in Figure 5.3, where the points residing on the boundary are connected and assumed to dominate the enclosed points.

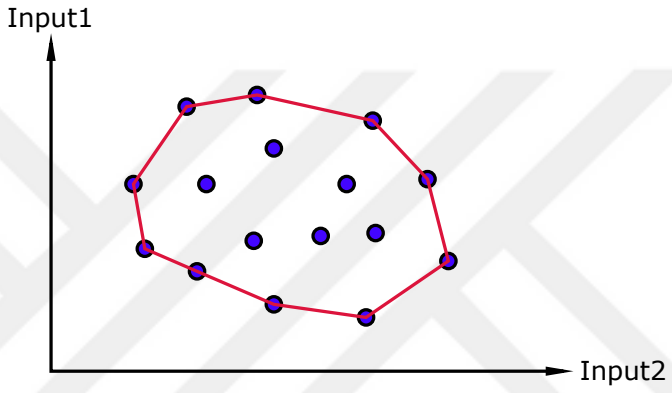

Figure 5.3. Boundary based pareto dominance.

After finding non-dominated points, the archive is filled. A new population is constructed from the members of the archive by using the crossover operation. In this step, randomly chosen archive members are used as parents, which produce two offsprings. In addition, the mutation operation described in [86] is used to increase the diversity of the population. This algorithm is repeated until the maximum iteration number is met. Finally, a boundary can be obtained by using the individuals in the archive at the end of the optimization.

5.3. Experiments

To test the validity of the approach, we performed experiments on three designs utilizing a dual-core computer with 2.4 GHz clock speed, and 4 GB RAM.

5.3.1. Inverter

As the first application of the methodology, an inverter was selected. The Simulink model was built by using two resistances, R_p and R_n, representing on-resistances for PMOS and NMOS, respectively. The internal capacitance (C_{int}) in addition to the load capacitance are charged or discharged through those resistances depending on the input voltage being either greater or smaller than the switching voltage, which is the voltage where the input voltage is equal to the output voltage. Internal design parameters, R_p, R_n and C_{int}, which are static, were extracted from transient analyses of the circuit. Then, the Simulink model was designed manually using the Embedded Matlab function considering also an additional load capacitance.

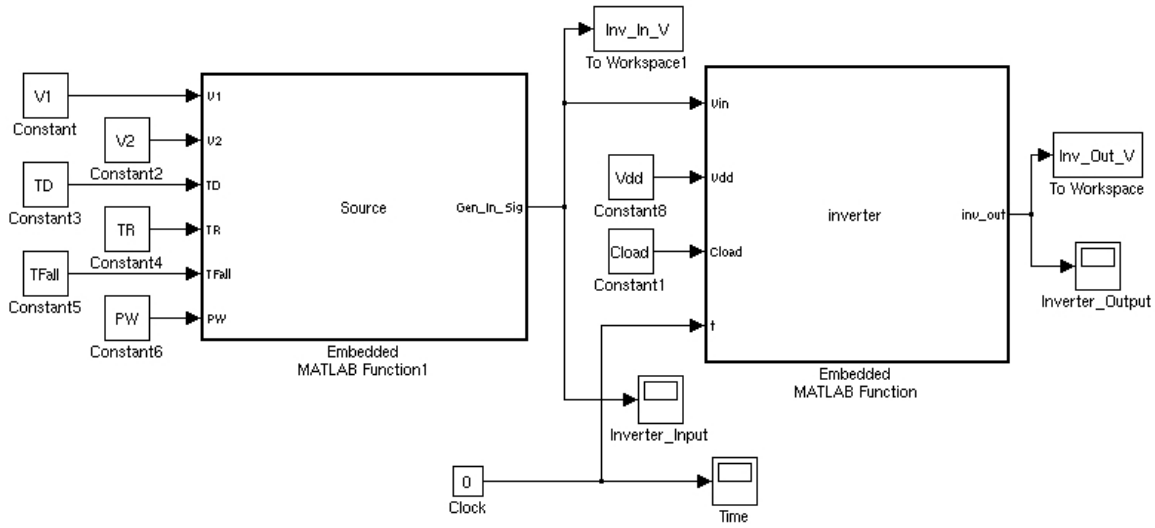

Figure 5.4. Simulink inverter model with input generator.

For testing the methodology, we put both implementations in a configuration, which applies a pulse waveform on the input and has a load capacitance on the output as shown in Figure 5.4. We determined the parameters in Table 5.1, as performance parameters, which characterize the performance and as input parameters, which modify the design state. Design of inverter implementations, parameters for inverter and the configuration for testing methodology are explained in Section 4.3.1 in detail.

Table 5.1. Parameters for inverter example.

Input Parameters	Load Capacitance (c_l), Rise Time (t_r), Fall Time (t_f)
Performance Parameters	Maximum Output Voltage, Minimum Output Voltage, Propagation Delay High to Low, Propagation Delay Mean, Propagation Delay Low to High, Fall Time, Rise Time

At the first phase of the experiment, we decided to generate samples for input parameters via Monte Carlo approach. For that case, we generated 10000 random input parameter samples to be fed to the Simulink model and HSpice implementation. We set the success criteria as 5%. We set lower and upper bounds for input parameters as [1, 1, 20] and [300, 200, 100] for t_r, t_f, and c_l, respectively. t_r and t_f were in ps and c_l was in fF. Then, we ran HSpice and Simulink simulations for those 10000 samples and we determined the successful 1935 samples with error percentage less than 5%, which are shown in Figure 5.5. As it was explained in the methodology section, we

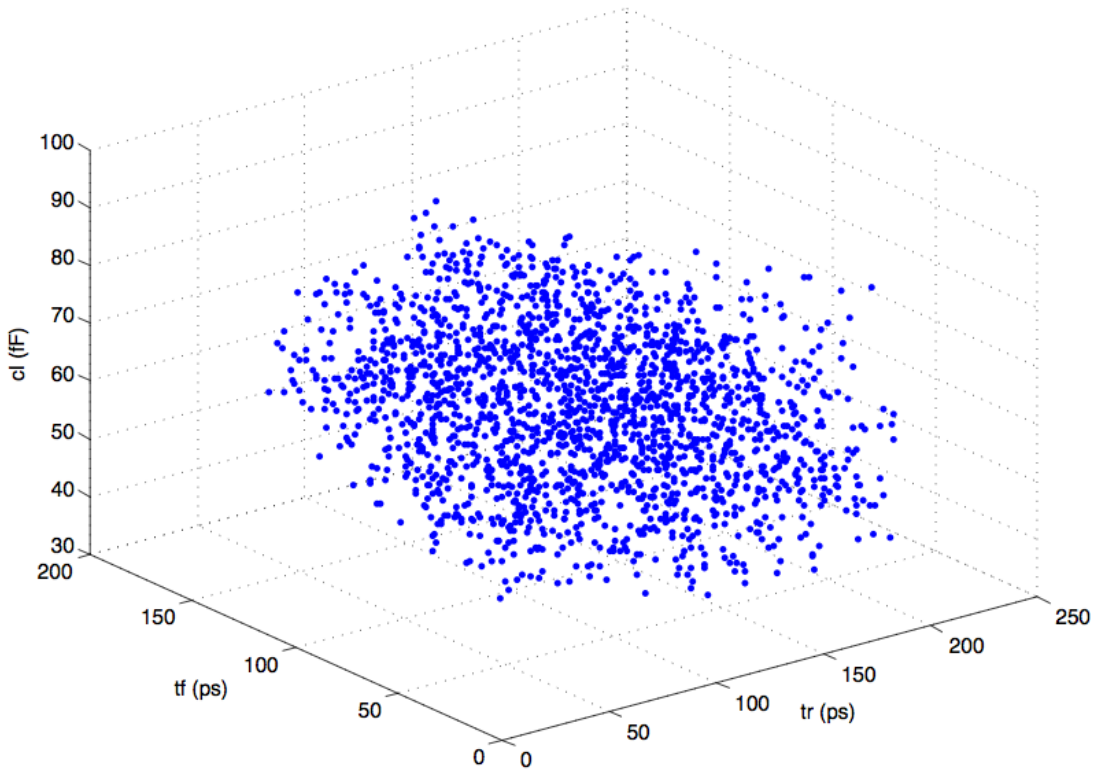

Figure 5.5. Successful points for inverter Monte Carlo run.

can obtain the boundary region by the Monte Carlo analysis. However, the increase in sample size increases the run time dramatically. So we applied our optimization based approach to the inverter case with same success criteria and bounds for Monte Carlo, as well. The number of individuals in the population was 99 and the number of generations was 30. As the result of the run, we obtained Figure 5.6, where the boundary was found by our approach and the points on the same figure are the points of the Monte Carlo run from previous step.

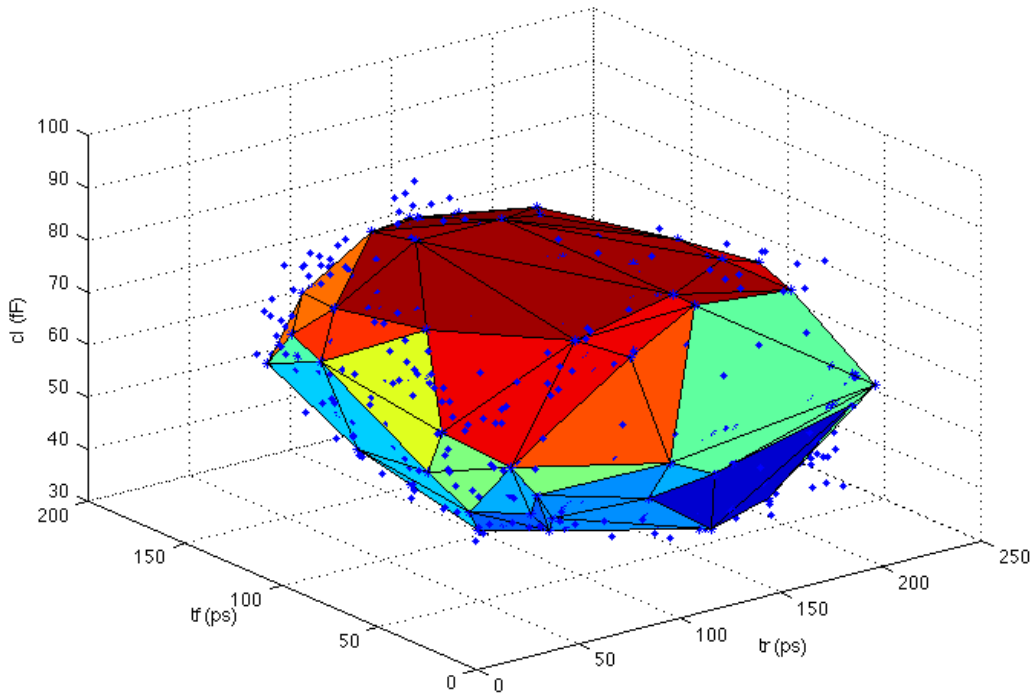

Figure 5.6. Boundary for inverter with population size 99 and generation number 30 and successful Monte Carlo points.

As observed from the figure, there are negligible number of points outside the boundary that our approach found. Our approach succeeds in finding the boundary about 30% of the time spent by the Monte Carlo approach. We have validated our approach by running it with the same population size, but for more generations. The number of points outside the boundary decreases with the increasing number of generations, but the boundary we have found with 30 generations returns favorable results.

In order to visualize the effect of our methodology better, we wanted to provide simulation waveforms for three points in Figure 5.6, one at the center of the boundary, one on the boundary and the last one being on the outside of the boundary. The values for t_r, t_f, and c_l and the average error percentages for the performance parameters are listed in Table 5.2. Simulation waveforms for those points are shown in Figure 5.7.

Table 5.2. Locations of points at the center, on the boundary and on the outside for inverter.

	t_r (ps)	t_f (ps)	c_l (fF)	*average error* (%)
center	60	100	70	3.6
edge	58.67	92.85	91.26	4.1
out	50	180	40	9.9

As it can be observed from Figure 5.7, the simulation waveforms for the center point show the best matching, as expected. When compared to the edge point, the center point's simulations show a better matching for rise time and propagation delay from low to high, whereas other metrics are similar with the edge point simulations. In comparison to the center and edge point, the outside point's simulations match much less. These results are inline with the goal of our methodology and they verify that our methodology finds the equivalence boundary successfully such that the performance parameters show better match for the points on the boundary.

5.3.2. Operational Amplifier (Opamp)

A rather more complex example is the opamp. In order to generate the HSpice netlist for opamp, we utilized an in-house optimization tool in Matlab [15]. We provided the performance targets to the tool, which generated the HSpice netlist for two stage operational transconductance amplifier in Figure 5.8.

For the Simulink model, we utilized the band-limited opamp in the Simscape library having the following parameters: *Gain*, R_{in}, R_{out}, V_{min}, V_{max}, V_{dot} (maximum slew rate), *Bandwidth* and *Initial Output Voltage* (V_0).

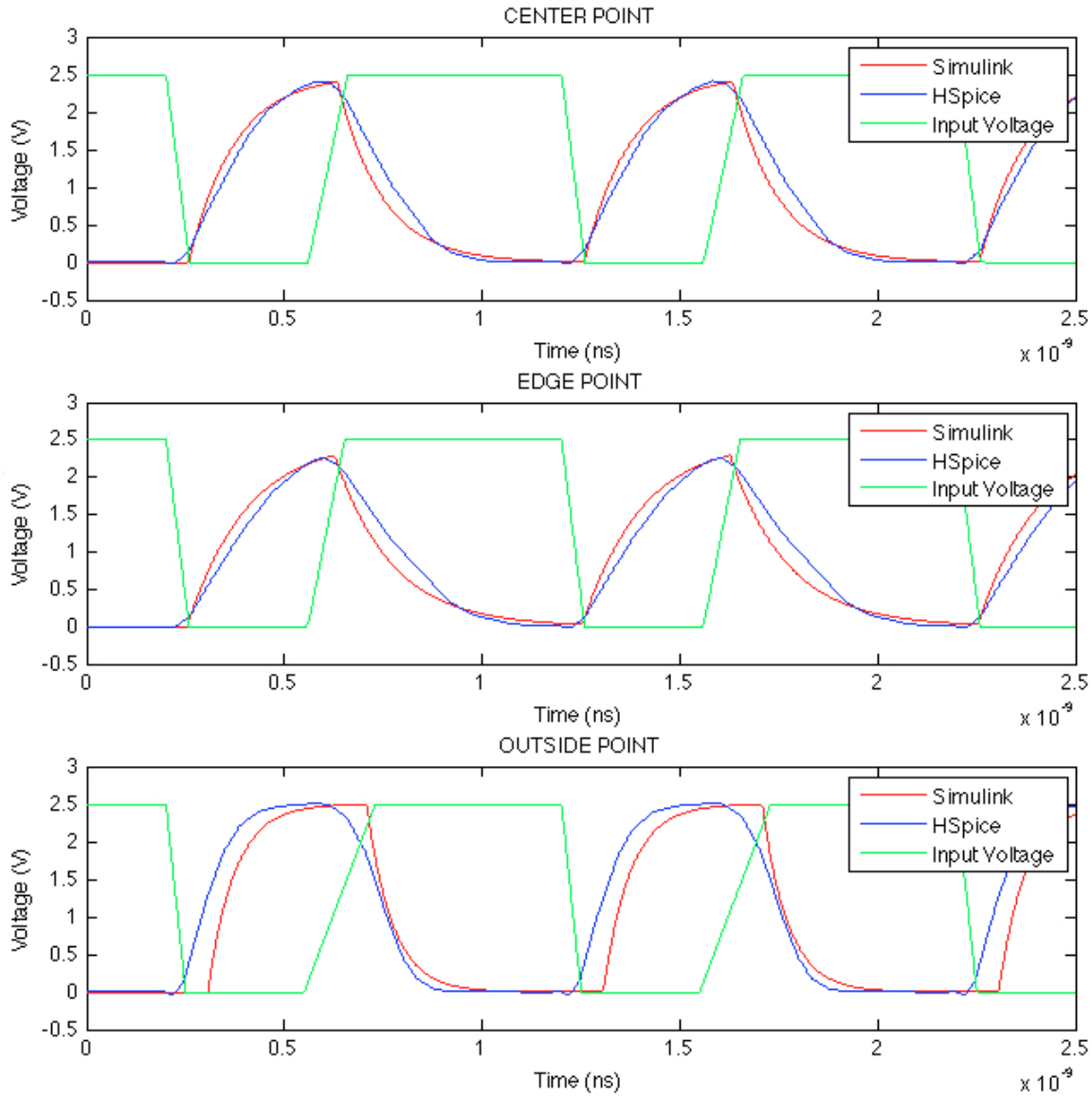

Figure 5.7. Simulink and HSpice simulations for the points at the center, on the edge and on the outside for inverter.

As discussed in Section 4.3.2, characterization of the opamp necessitates multiple configurations. In this experiment, we utilize the same configurations in Section 4.3.2 and the frequency domain configuration was previously shown in Figure 4.8.

We used the parameters in Table 5.3 as performance and input parameters, where the parameters for time and frequency domain configurations are listed separately. As a difference from the inverter case, application of the methodology requires executing multiple configurations. After execution of each configuration as a single step of the optimization process, we extracted the performance parameters related to each config-

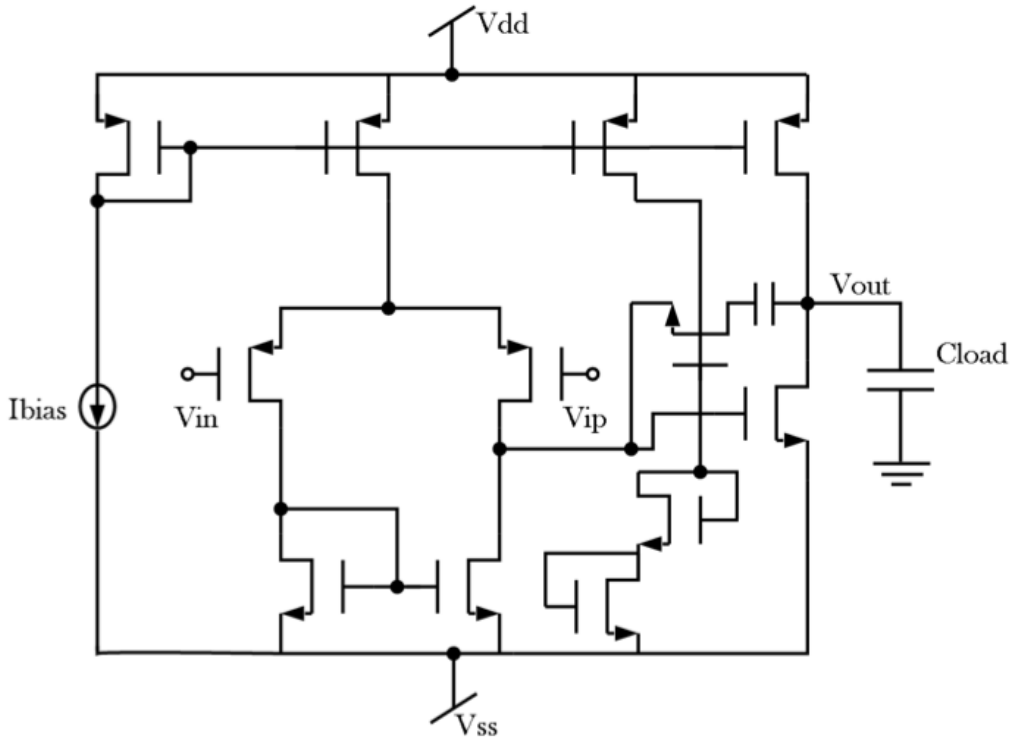

Figure 5.8. Circuit diagram for two stage operational transconductance amplifier [15].

uration and at the end of each step, we calculated the percentage errors between the performance parameters.

We set the success criteria as 20%. We set lower and upper bounds for input parameters as [0.03, 0.03, 10] and [0.3, 0.3, 200] for t_r, t_f, and c_l, respectively. t_r and t_f were in ms and c_l was in pF. The number of individuals in the population was 99 and the number of generations was 30. As the result of the run, we obtained Figure 5.9, where our approach was able to find the equivalence boundary for the opamp.

Like in the previous experiment, we wanted to provide simulation waveforms for three points on Figure 5.9. The values for t_r, t_f, and c_l and the average error percentages for the performance parameters are listed in Table 5.4. Simulation waveforms for those points are shown in Figure 5.10.

As it can be observed from Figure 5.10, the simulations match best for the point at the center in terms of high to low and low to high transitions etc. In comparison to

Table 5.3. Parameters for opamp example.

Input Parameters (time domain)	Load Capacitance (c_l), Rise Time (t_r), Fall Time (t_f)
Input Parameters (frequency domain)	Load Capacitance (c_l)
Performance Parameters (time domain)	Maximum Output Voltage, Minimum Output Voltage, Slew Rate High to Low, Slew Rate Low to High, Fall Time, Rise Time
Performance Parameters (frequency domain)	Cut-off Frequency

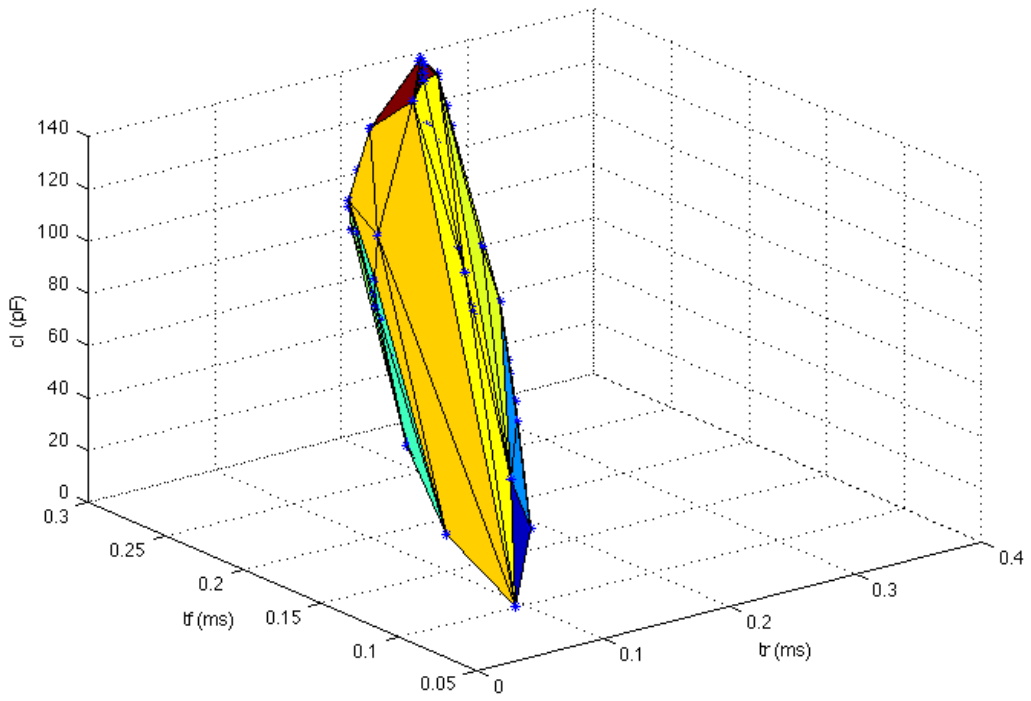

Figure 5.9. Boundary for opamp with population size 99 and generation number 30.

the center point, there is more difference between simulations during high to low transition for the edge point. For the outside point, simulation waveforms do not resemble each other much. These waveforms verify that our methodology finds the equivalence boundary such that for the points inside the boundary, behavioral model and circuit level simulations show better match considering the defined error percentage.

Table 5.4. Locations of points at the center, on the boundary and on the outside for opamp.

	$t_r\ (ms)$	$t_f\ (ms)$	$c_l\ (pF)$	*average error* (%)
center	0.15	0.15	40	17.95
edge	0.161	0.245	96.8	18.64
out	0.3	0.3	20	37.87

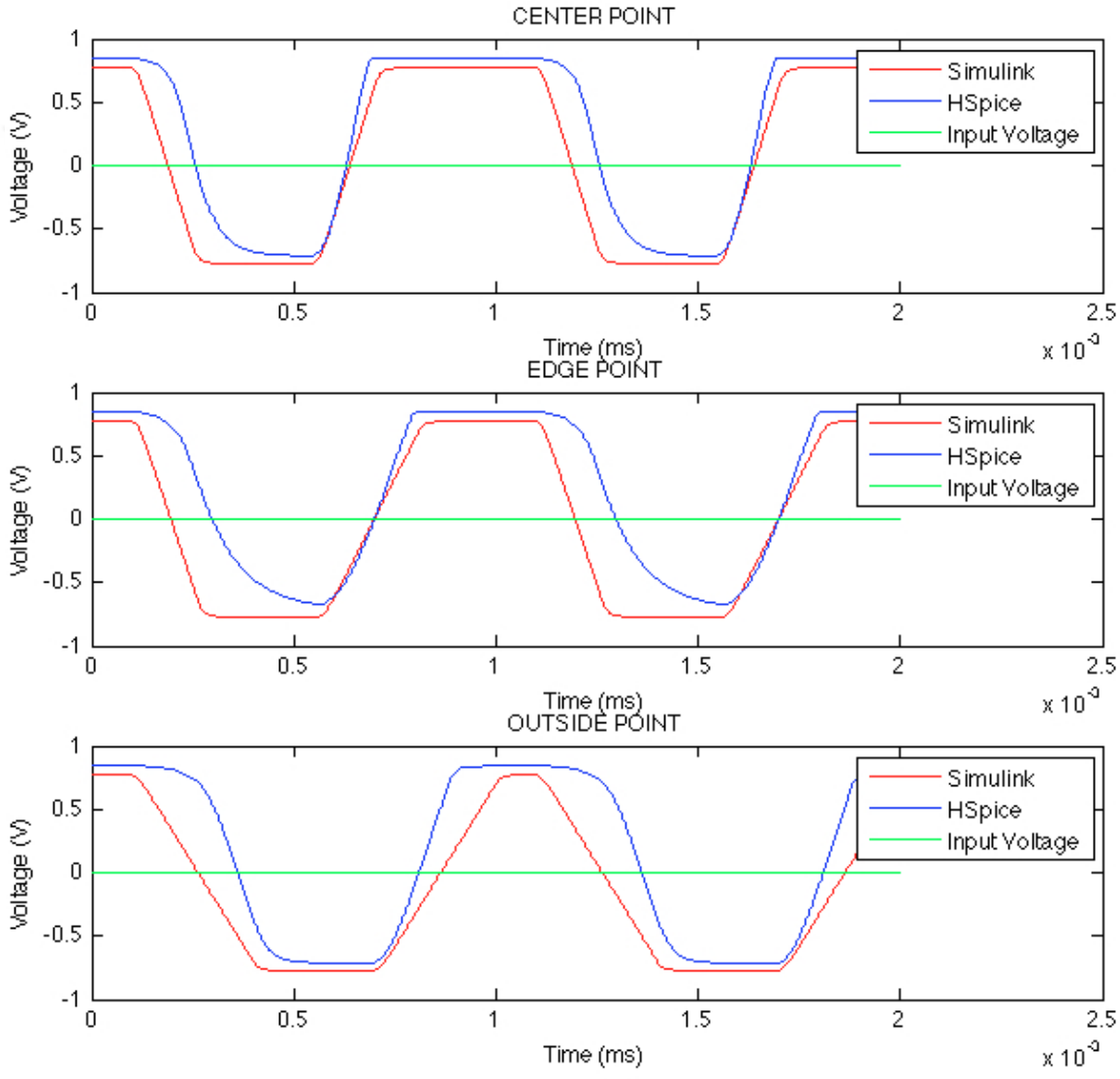

Figure 5.10. Simulink and HSpice simulations for the points at the center, on the edge and on the outside for opamp.

5.3.3. Buck Converter

As the last experiment, we chose the buck converter, which is commonly used in power management IC's in order to step down the voltage. Design of the model

and the circuit implementation, and determination of the parameters for testing the methodology were discussed in Section 4.3.3 in detail. In its basic form, buck converter contains two switches (usually a diode and a transistor) controlling the current through an inductor. The inductor is in series with a parallel connected capacitor and load resistance.

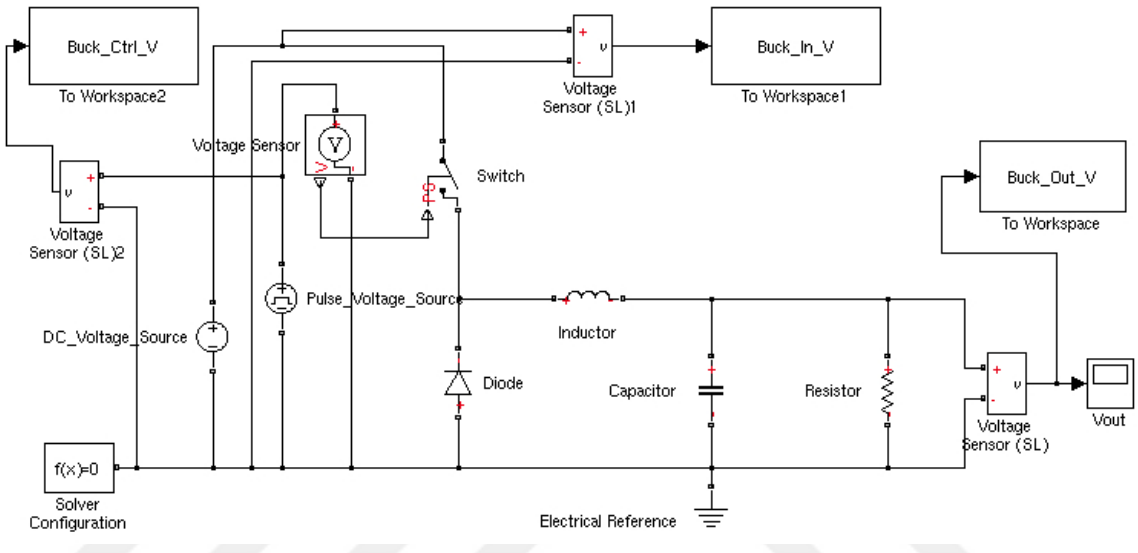

Figure 5.11. Simulink model of the buck converter.

For the Simulink model, we utilized components in the Simscape library. We used switch as the transistor and pulse waveform for switching. Our Simulink model can be seen in Figure 5.11.

In the Simscape library, various parasitic characteristics were integrated to the component models like switch on, diode on, inductor series and capacitor series resistances. This functionality makes it possible to analyze the effects of those parasitic components in the methodology. In the circuit level implementation, we built switch by means of a voltage controlled resistor having a resistance value varying according to the applied control voltage in a piecewise linear relationship. We added series inductor and capacitor resistances to the netlist for analysis. Our circuit topology for buck converter was as in Figure 5.12.

We used the parameters in Table 5.5 as performance and input parameters. It should be noted that for input parameters, two sets of parameters were introduced.

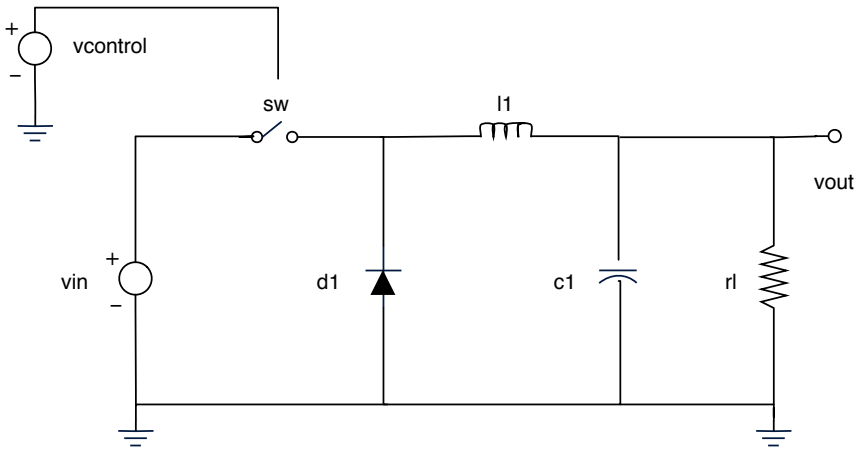

Figure 5.12. Circuit topology for buck converter.

One set contained the major components affecting the functionality and the second set included the parasitic resistances as listed on Table 5.5. We should note that inductor and capacitor resistances are series resistances. Also on-resistances of the switch and the diode were defined to be equal. For ease of demonstration, we limit the number

Table 5.5. Parameters for buck converter example.

Input Parameters (major components)	Input Voltage (v_{in}), Pulse Width of Controlling Voltage Source (pw), Load Resistance (r_l)
Input Parameters (parasitic resistances)	Inductor Series Resistance (r_{ind}), Switch-on Resistance (r_{sw}), Capacitor Series Resistance (r_{cap})
Performance Parameters	Maximum Output Voltage, Mean Output Voltage, Ripple Voltage

of input parameters to three, to show the boundary of equivalence. Since the effects of major components on the performance parameters are more significant compared to minor components, we searched the boundary for major input parameters and fixed values of minor components to their optimal values. In summary, the input parameters are v_{in}, pw and r_l.

We set the success criteria as 10%. In the first phase of our application, we wanted to demonstrate our approach in two dimensions. To that end, we decided to fix the load resistance value and ran our approach in 2 dimensions for pw and v_{in}. We

determined lower and upper bounds for input parameters as [50, 1] and [330, 20] for pw and v_{in}, respectively. We fixed r_l to 400 ohms. The number of individuals in the population was 99 and the number of generations was 30. Following the application of the approach, we obtained Figure 5.13. As it is observed from the figure, the boundary is found in two dimensions and it encapsulates the successful points.

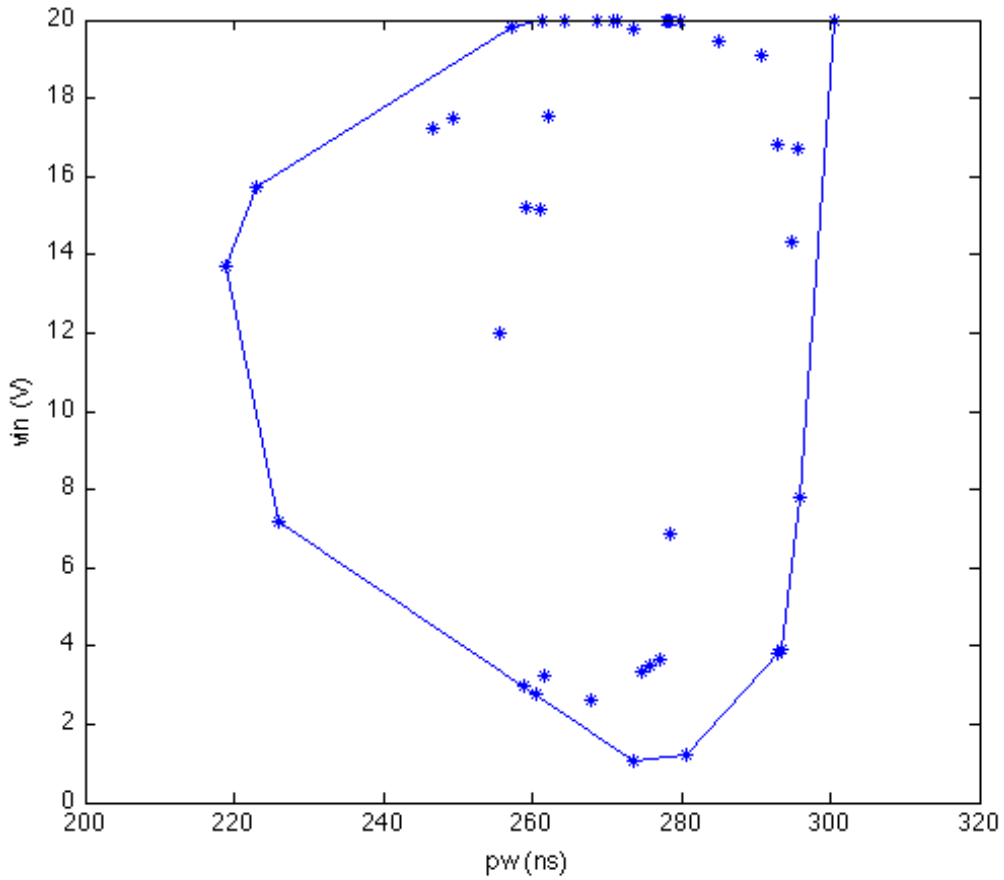

Figure 5.13. Boundary in 2d for buck with population size 99 and generation number 30 and r_l=400.

Then, we applied our approach in three dimensions. We determined lower and upper bounds for input parameters as [50, 1, 1] and [330, 20, 2000] for pw, v_{in}, and r_l, respectively. pw was in ns, v_{in} was in volts and r_l was in ohms. The number of individuals in the population was 99 and the number of generations was 30. As the result of the run, we obtained Figure 5.14. As can be seen from Figure 5.14, our approach was able to find the equivalence boundary. As buck design works better for

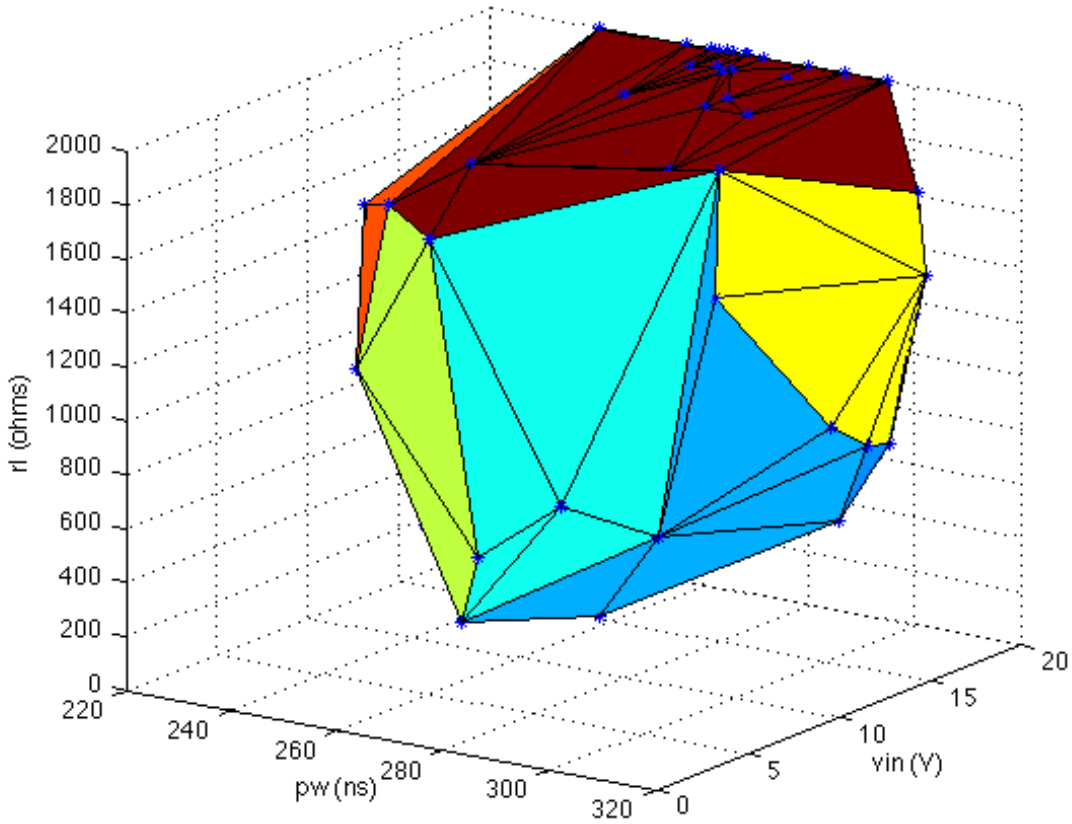

Figure 5.14. Boundary for buck with population size 99 and generation number 30.

higher input voltages and load resistances, the boundary extended to the upper bounds for those input parameters.

Like the previous experiments, we wanted to provide simulation waveforms for three points on Figure 5.14. The values for pw, v_{in}, and r_l and the average error percentages for the performance parameters are listed in Table 5.6. Simulation waveforms

Table 5.6. Locations of points at the center, on the boundary and on the outside for buck converter.

	$pw\ (ns)$	$v_{in}\ (V)$	$r_l\ (ohms)$	$average\ error\ (\%)$
center	268	494	1800	6.81
edge	265.7	5.09	283.4	9.49
out	280	4	200	15.62

for those points are shown in Figure 5.15.

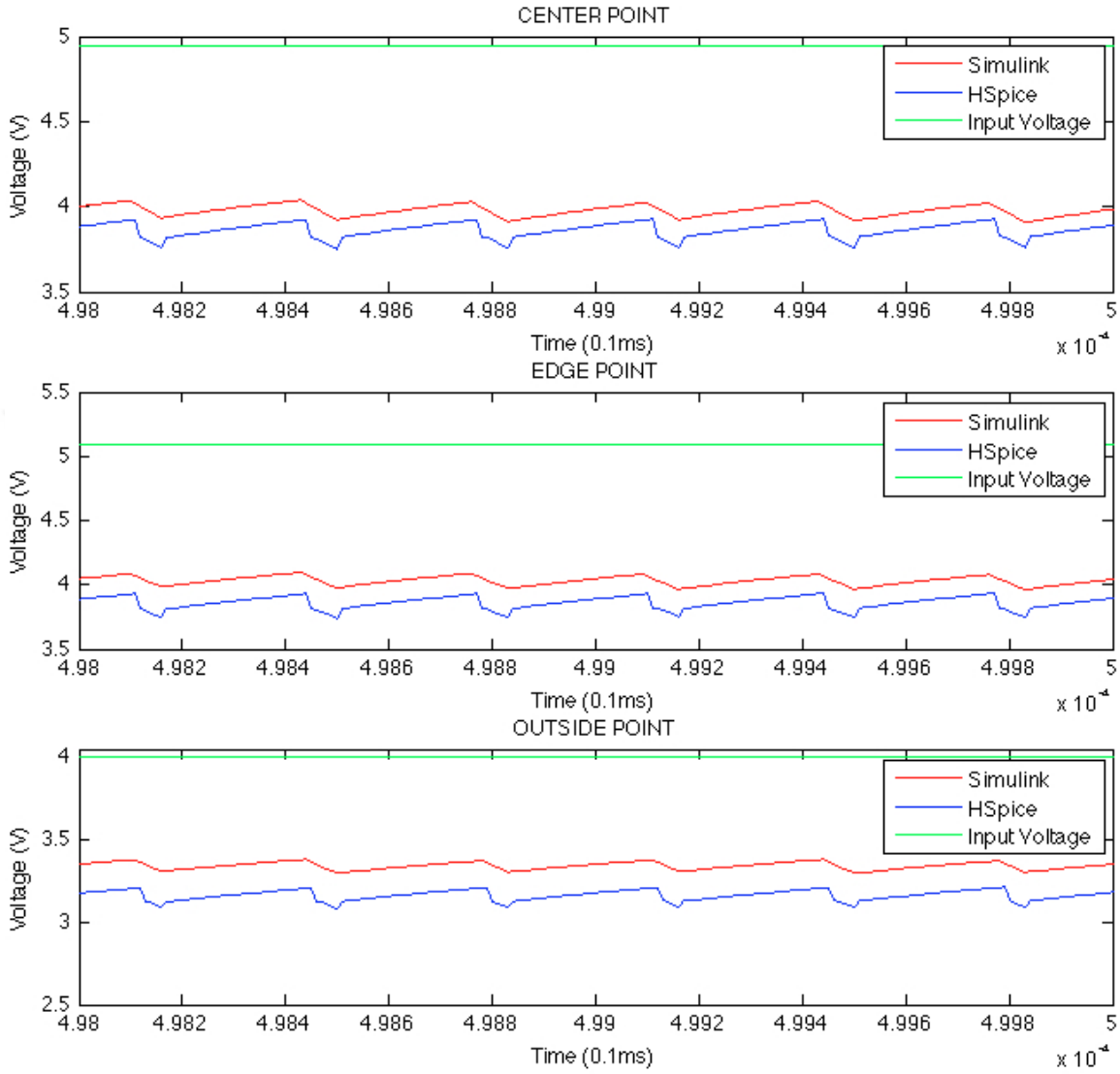

Figure 5.15. Simulink and HSpice simulations for the points at the center, on the edge and on the outside for buck converter.

As it can be observed from Figure 5.15, the voltage mismatch between Simulink and HSpice simulations increases from center point to edge point and from edge point to outside point. To be more specific, the voltage mismatch is 100 mV for the center point, where it is 150 mV for edge point and finally it is about 180 mV for the outside point. Also it should be taken into consideration that the regulated output voltages for center and edge points are about 4 V, whereas it is about 3.3 V for outside point, which makes the error percentage for outside point bigger. These results are again

inline with our expectation that matching between waveforms is the best for the center point and the worst for the outside point, and so verify that our methodology achieves its goal.

5.4. Conclusion

In this chapter, a simulation-based analog equivalence boundary search methodology for high level Simulink models and their low level HSpice counterparts was presented.

The equivalence of high and low level designs is determined by comparing a set of predefined performance parameters measured during the simulation of both models. The proposed methodology investigates the search space to obtain boundary of input parameters, where both models have equivalent performance parameters. An optimization problem is built, where the error percentage between the performance parameters of both models being less than a specified threshold is defined as success criteria. In this problem, input parameters are determined by utilizing evolutionary computation. At the end of the optimization, the border of equivalence for the models is found for input parameters satisfying the success criteria.

The validity of our approach was demonstrated on three designs, an inverter, an operational amplifier, and a buck converter, where our approach proves to be an efficient tool in finding an equivalence boundary of analog circuits and models.

6. VARIABILITY ANALYSIS ON ANALOG APPROACH

6.1. Introduction

As feature sizes decrease more and more due to advances in process technologies, nano-scale effects become a problem for designing ICs. There is a continuous need for updating design and manufacturing processes to cope with those effects. Process variation is one of the most important effects in nano-scale design. In addition, time-to-market pressure is another factor whose influence is felt seriously in industrial projects. The largest component of the time spent on industrial integrated circuit projects is the verification of the design. In order to decrease the time spent on verification, simulation times should be decreased and design bugs should be detected at early phases of the project. All these facts necessitate the use of model-based design (MBD), speeding up simulations and enabling the discovery of bugs in early phases. In MBD, functionality of the model should be checked against its implementation in order to ensure the validity of the model and process variations should be taken into consideration.

Methodologies and tools utilized in analog equivalence checking are far from being mature. There is a limited number of studies on analog verification that integrate process variation analysis. In this chapter, we propose a methodology generating behavioral equivalence boundary under the effect of process variations. We use Simulink as high-level model because it is a widespread design environment for both digital and analog models considering both the availability of numerous built-in functions and its ability to integrate user-defined functions. For the low-level implementation, we use HSpice.

There are various approaches investigating analog equivalence checking (AEC) problem. In [29, 30, 64, 65], linearity of analog circuits is assumed and checking is accomplished by comparing linear transfer functions of the specification and the implementation. But transfer function generation is not an easy task and linearization requires manual intervention. Another approach utilized in [8–10] is state space sam-

pling by using non-linear transformations, where difficulty of finding the correct transformations and low possibility of automation affects the soundness of the approach. Another approach is based on input stimuli generation covering the complete state space [7]. Although the approach is reasonable, its applicability is limited by the exponential increase in the run time due to the growth in the number of variables. In order to overcome the state-space explosion problem, we decided to follow an approach similar to [12], which presents a hierarchical, optimization based semi-formal (simulation-based) equivalence checking methodology. In that approach equivalence is verified over a constrained input space and the optimization problem is formulated.

We built an analog equivalence boundary search methodology utilizing evolutionary computation in [35]. That evolutionary computation is a modified version of SPEA2 [32]. The approach in [35] is simulation-based and decides equivalence by comparing performance parameters measured from simulations of Simulink model and HSpice netlist for the same input parameters, which characterize the states of inputs. However the approach in [35] does not integrate process variation effects in the analysis.

Process variations were modeled by [87]. Although there is a formal approach to verify analog circuits in the presence of noise and process variation in [77], it is limited to linear time-invariant designs, since it is dependent on the closed form solution for the statistical differential equation (SDE). In [78], again SDE is utilized in verification of analog circuits in presence of noise and process variations using pattern matching. The same limitation of SDE applies here as well and the methodology is limited to fixed simulation step-size.

Two example use cases of our equivalence checking method are its application in analog design and model generation. Considering the analog design flow, the architectural exploration of the design space is more easily performed at the system level. This is achieved by utilization of high level models. However, the accuracy of these models has to be validated before usage. The second usage of our method is for model generation. During the process of model generation, the equivalence of the model to the actual design should be checked under the effect of process variations. By usage

of our methodology, designers can be sure about the operation of the model utilized in the system level simulations.

We make the following contributions by the research presented in this chapter:

- We analyze the effects of process variations on AEC, where we generate an equivalence boundary. We investigate the change in equivalence boundary by filtering for different target equivalence threshold values.
- We utilize evolutionary computation in our process variations-aware AEC analysis.
- We apply Quasi Monte Carlo (QMC) method in process variations-effect analysis, which makes it possible to estimate the yield with fewer samples compared to Monte Carlo (MC).
- We build a multi-objective optimization problem, where in addition to equivalence value, we consider closeness to the equivalence boundary by defining a parameter for each point of concern.
- We demonstrate the validity of our approach on three designs, an inverter, an operational amplifier and a buck converter.
- We utilize Simulink and HSpice only for validating the proposed approach. The overall approach is independent of them, any high level model and any circuit simulator can be used in our process variations-effect analysis.

The chapter is organized as follows. In Section 6.2, we present our methodology following its flow graph. In Section 6.3, we present the applications of our methodology on three examples: Inverter, Operational Amplifier and Buck Converter and provide the results of those applications. In Section 6.4, we discuss the results of the experiments. In Section 6.5, we conclude the chapter.

6.2. Methodology

In this section, we first describe our approach to check analog equivalence and explain our process variations-effect analysis for analog behavioral equivalence boundary

computation. Then, we define the inputs of our yield-analysis and explain the details of each step in the analysis in subsections.

To perform equivalence checking we utilize a set of performance and input parameters for high-level (Simulink) and low-level (HSpice) designs. *Performance parameters* capture the essential characteristics of designs during their simulation. Some typical performance parameters are propagation delay, slew rate, and output voltage. *Input parameters* characterize the state of the inputs to the design, e.g., rise time of the input voltage, input voltage value, and parasitic resistances. In our approach, we simulate Simulink model and HSpice netlist with different input parameters and measure performance parameters from those simulations. We decide analog equivalence by comparison of those performance parameters.

The goal of our methodology is to determine the effect of process variations on the equivalence boundary of input parameters. To reach that goal, the difference between the performance parameters of the models being below a given threshold is defined as our success criteria. We use a user-defined *success threshold* because analog signals cannot be exactly identical. Hence, we can determine the regions, where both designs are equivalent to each other within a tolerance value under the effect of process variations.

Now, we describe our process variations-effect analysis for analog equivalence boundary computation, which is shown in Figure 6.1.

6.2.1. Inputs of Analysis

We perform process variations-effect analysis on an equivalence boundary, which is a set of points. Those points are collections of input parameters. As explained in [35], those points are determined by building a multi-objective optimizer. That optimizer provides the values of input parameters necessary to simulate both HSpice and Simulink models. Then, for each simulation, a set of performance parameters is measured. Following that, the mismatches between the performance parameters for

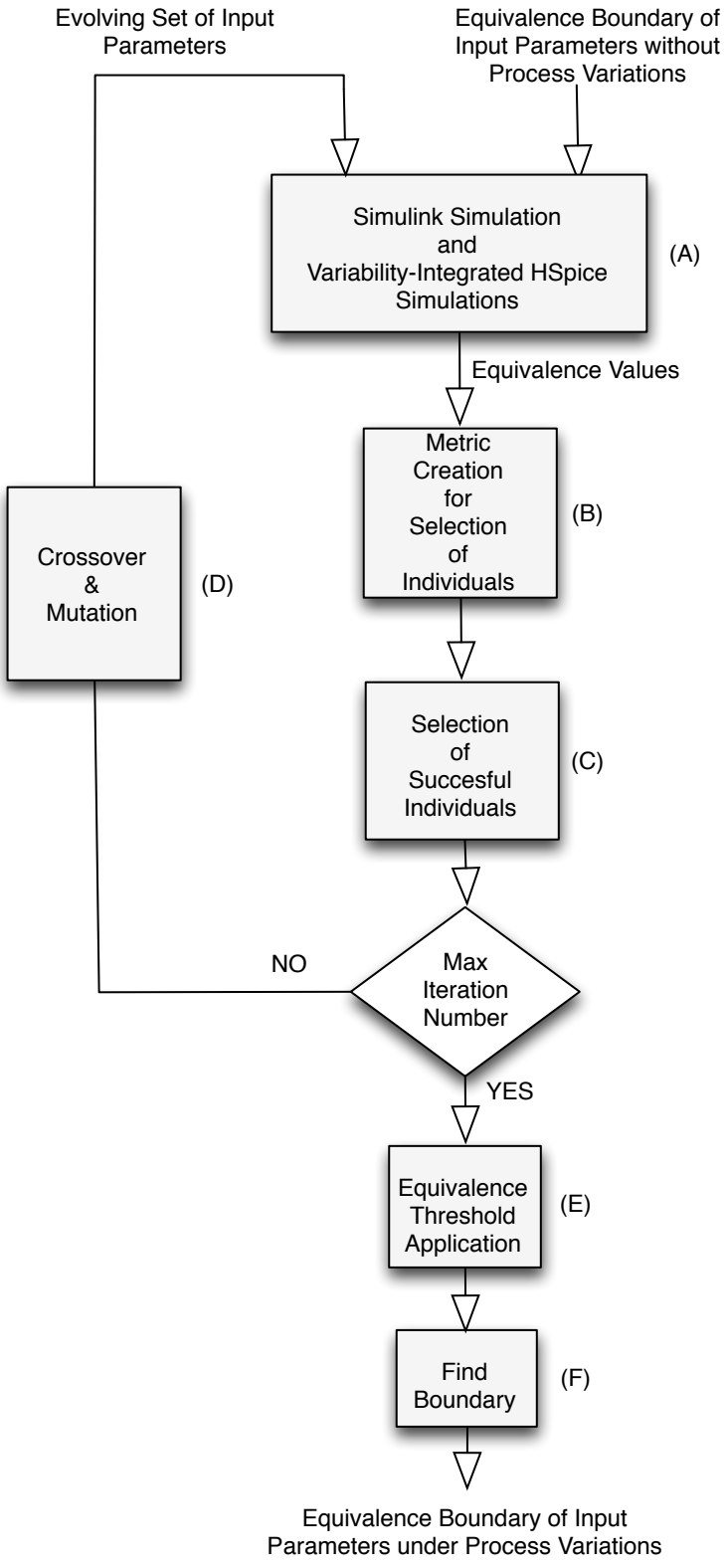

Figure 6.1. Flow diagram for process variations-effect analysis for analog behavioral equivalence boundary computation.

HSpice and Simulink results are measured in percentages and given as input to the optimizer. By applying a success threshold level as a success criterion, feasible set elimination is done. As part of evolutionary computation approach used, successful points on the boundary are selected to archive. Finally, successful individuals are utilized to create the next generation individuals by applying cross-over and mutation. Repeated application of this method results in the boundary points of input parameters.

Those boundary points are the inputs of our yield analysis, which investigates the effect of process variations. In brief, genetic algorithm with multiple generations of individuals is utilized in our analysis, the mechanics of which be detailed in the following sections in parallel with the flow graph in Figure 6.1. For all generations, individuals, which symbolize the input parameters of concern, will be analyzed. The first generation of individuals will be the boundary points provided by the equivalence boundary search methodology.

6.2.2. Simulations

In this subsection, we describe the functionality performed by block (A) in Figure 6.1. Our approach is based on comparison of performance parameter values measured from simulations of Simulink models and circuit netlist. Simulink and HSpice simulations are performed for different sets of points corresponding to different combinations of input parameters. Simulink simulations are performed only on nominal high-level models, as process variations are only reflected on HSpice netlist.

The impact of variations on the netlist is achieved by inserting technology specific parameters to the netlist. As process variations are random, analysis of their effects should handle that randomness. This is generally achieved by generating samples around the nominal point utilizing the MC approach and simulating the circuit for those samples. For the generation of samples, we employ the Sobol sequence, which applies the QMC approach. The generation of the samples by using Low Discrepancy Sequences (LDS) in QMC is advantageous compared to MC since more homogeneously distributed initial samples result in fewer samples [85]. As our methodology

is simulation-based, reaching the result with fewer samples can make it much more applicable in terms of run time. Those generated samples are represented in the netlist as variations in width (w) and length (l) of transistors as well as threshold voltage (v_{th}) and oxide thickness (t_{ox}) as discussed in Pelgrom model [87]. Once all samples are generated, HSpice simulations are carried out and performance parameters are measured for each sample.

In the next step, the error percentages for performance parameters are calculated considering the Simulink performance parameters as golden. Then, those error percentages are averaged and the result is compared with the success threshold, since analog signals can not be identical exactly. The success criterion for that sample is set as the average error percentage being smaller than the success threshold. After each sample is classified as success or fail case according to that success criterion, the number of successful samples are divided by the number of all samples. By multiplying the division result by 100, the equivalence value for a boundary point is calculated.

It should be noted that the determination of the success criterion as application of threshold to the average of performance parameters' error percentages is just for ease of demonstration. As each error percentage for each performance parameter is measured individually, the success criterion could be defined to be different values for each performance parameter without introducing much complexity.

6.2.3. Metric Creation for Selection of Individuals

As block (B) in Figure 6.1 suggests, a metric is to be created in order to filter the boundary points following the simulations step. In our methodology, we focus on finding the boundary points with equivalence values greater than a certain equivalence threshold. Hence, for the selection of the points, we define an objective function including both the equivalence value and closeness to the boundary. We define a variable parameterizing closeness to boundary. We design an algorithm assigning a boundary number to each point. The algorithm processes all existing points and assigns value 1 to all the outermost points. Those points are removed from the set of points. Then, the

algorithm assigns the incremented boundary number value 2 to the next outer points. This continues recursively until each point in the start is assigned a boundary number. Determination of the outermost boundary points is achieved by Matlab's *boundary* function. In order to visualize the operation of the algorithm, a conceptual illustration for the two dimensional case is shown in Figure 6.2.

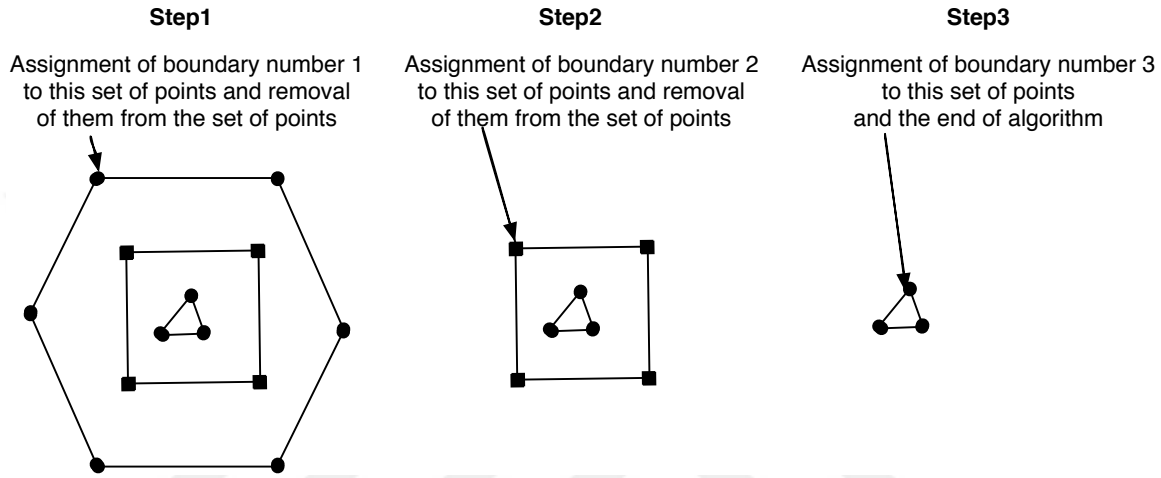

Figure 6.2. Illustration for boundary number assignment algorithm.

The objective function for a point p to be maximized is as follows:

$$f(p) = equivalence(p)/100 + 1/boundary_number(p) \tag{6.1}$$

As Equation 6.1 suggests, the effect of equivalence is normalized and the effect of boundary number decreases with the distance from the boundary.

6.2.4. Selection of Successful Individuals

In this subsection, we describe the functionality performed by block (C). Following the calculation of the objective function value, each point has a metric which makes categorizing them possible. Rather than eliminating the points below a certain equivalence value, we rank the points according to the objective function value. We select the more successful half of the points. This approach is employed to prevent a greedy search on the equivalence values. The selected half of the points are included in the archive.

At the end of this step, we check the maximum iteration number. If the loop count is smaller than the maximum iteration number, the loop is continued with crossover and mutation.

6.2.5. Crossover and Mutation

In this subsection, we describe the functionality performed by block (D). After the archive is filled, a new population is constructed from the members of the archive by using the crossover operation. In this step, randomly chosen archive members are used as parents, which produce two offsprings. In addition, the mutation operation described in [86] is used to increase the diversity of the population. Modeling of input parameters as chromosomes and the generation of offsprings are illustrated in Figure 6.3. Offsprings are generated three at a time until selected population size is reached. The members of the new population are input to block (A).

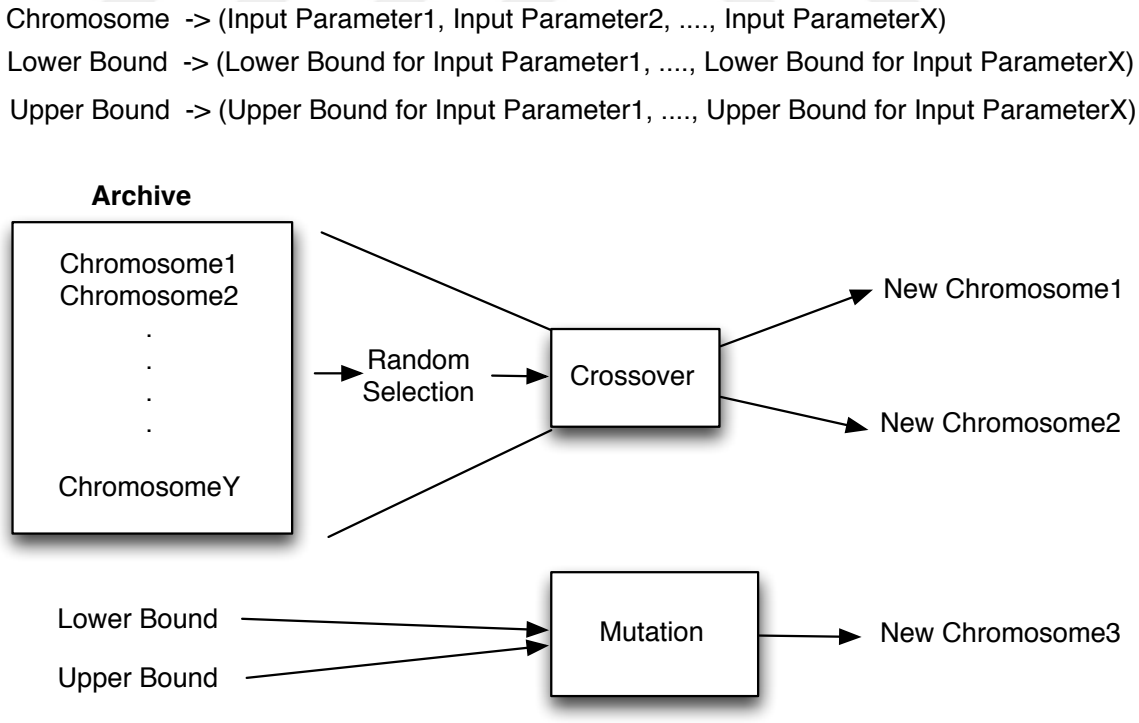

Figure 6.3. Modeling of chromosomes and generation of offsprings.

6.2.6. Equivalence Threshold Application and Finding Boundary

In this subsection, we describe the functionalities performed by blocks (E) and (F). When the loop is concluded, our methodology returns an archive, which contains successful individuals from each iteration. As explained above, those individuals were selected according to the objective function value. Since our main goal is to obtain the individuals satisfying a equivalence threshold, individuals below the desired equivalence threshold are discarded. The boundary function of Matlab is applied to the remaining points, generating a process variation-aware equivalence boundary.

6.3. Experiments

The algorithm outlined above has been coded in Matlab since it supports Simulink as the modeling platform for analog designs. In Simulink, we generated analog high-level models in two ways: by user-defined Matlab functions or by built-in libraries, specifically the Simscape library, which contains a broad range of electronic component models. We also used HSpice to simulate low-level analog designs. It is important to emphasize that these tools have been used only for validating the proposed approach. Therefore, the overall approach is independent of the design environment. As a result, any high-level model and circuit simulator could be utilized in this analysis.

To test the validity of our approach, we performed experiments on three designs, namely inverter, operational amplifier and buck converter. Design of the model and the circuit implementation, and determination of the parameters for testing the methodology were discussed in Sections 4.3.1, 4.3.2, and 4.3.3 for inverter, opamp and buck converter in detail, respectively. Details in those sections are summarized for each experiment in the following sections for ease of read.

6.3.1. Inverter

The Simulink model of the inverter was built by using two resistances, R_p and R_n, representing on-resistances for PMOS and NMOS transistors, respectively. The

Table 6.1. Parameters for inverter example.

Input Parameters	Load Capacitance (C_l), Rise Time (t_r), Fall Time (t_f)
Performance Parameters	Maximum Output Voltage, Minimum Output Voltage, Propagation Delay High to Low, Propagation Delay Mean, Propagation Delay Low to High, Fall Time, Rise Time

internal capacitance (C_{int}) in addition to the load capacitance (C_l), are charged or discharged through those resistances depending on the input voltage being either greater or smaller than the switching voltage. Internal design parameters, R_p, R_n and C_{int}, which are static, were extracted from transient analyses of the circuit. Then, the Simulink model was designed manually using the Embedded Matlab function considering also an additional load capacitance.

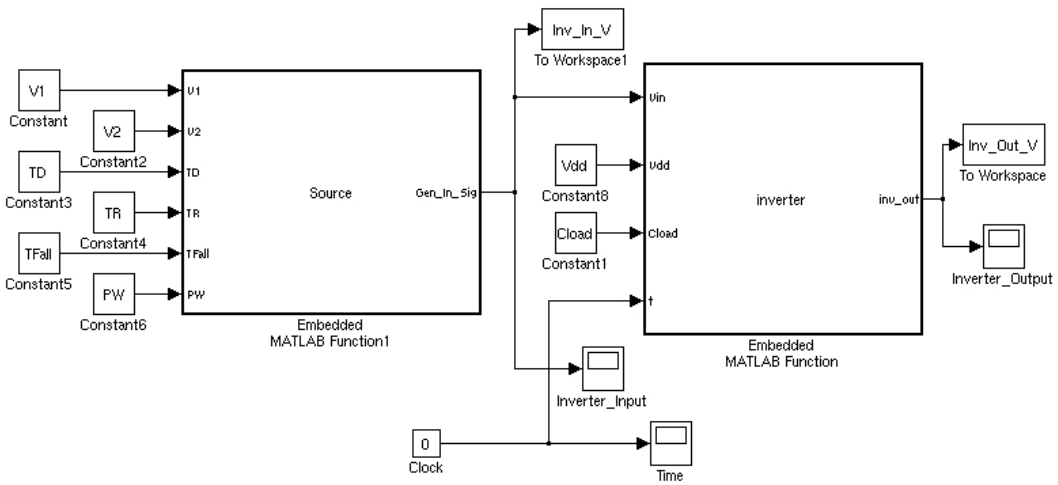

Figure 6.4. Simulink inverter model with input generator.

For testing the methodology, we put both implementations in a configuration, which applies a pulse waveform on the input and has a load capacitance on the output as shown in Figure 6.4. We determined the parameters in Table 6.1, as performance parameters, which characterize the performance and as input parameters, which modify the design state.

As it would be more explanatory to show the application of our approach in two dimensions, we decided to fix the value of C_l and run our methodology for two input parameters: t_r, t_f. To that end, the first step was to run the equivalence boundary search algorithm in [35] for this example for two input parameters. The effect of process variation would be analyzed on the resulting boundary. We set the success criterion for average error percentage between performance parameters as 5%. We set lower and upper bounds for input parameters as [1 ps, 1 ps] and [300 ps, 200 ps] for t_r and t_f, respectively. After running the equivalence boundary search methodology for 20 generations, where each generation was made up of 99 individuals, we obtained the boundary points shown in Figure 6.5 as the outermost boundary.

The effect of process variations were represented in the netlist. As explained in Chapter 6.2, variability was modeled according to Pelgrom model [87]. Process variation parameters are represented in the netlist via values for A_{VT} and A_β, which are provided for a 180nm process in [88]. The equations for them are given below:

$$\sigma^2(\Delta V_T) = A_{VT}^2/W.L \tag{6.2}$$

$$(\sigma\Delta\beta/\beta)^2 = A_\beta^2/W.L \tag{6.3}$$

We included those variations in the analysis by parameterizing them for two transistors in the inverter design. We generated different samples having different values for variation parameters by Sobol sequence as a method for QMC. We generated 40 samples for each boundary point and applied our proposed methodology for 5 generations of 99 individuals. At the end of the run, we filtered archive members for equivalence values of 70%, 80%, 90% and 100%. The result of the run is shown in Figure 6.5.

As seen in Figure 6.5, addition of the process variation effects results in contraction of the area of equivalence, as it was expected. Also it can be observed that area of equivalence contracts with more rigorous equivalence value constraints. For better

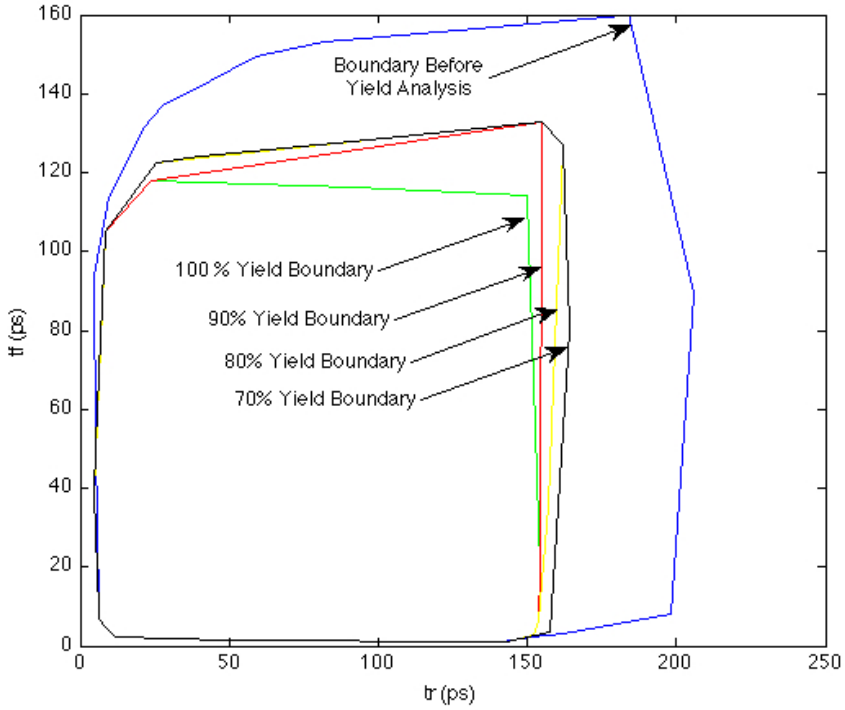

Figure 6.5. Equivalence boundaries for inverter before process variations-effect analysis and for different equivalence values with population size 99 and generation number 20.

visualization, the archive of points for 5-generation run filtered for 70% yield together with the boundary is shown in Figure 6.6.

As the next step, we ran our methodology for three input parameters including load capacitance (C_l) in addition to t_r and t_f. We set lower and upper bounds for input parameters as [1 ps, 1 ps, 20 fF] and [300 ps, 200 ps, 100 fF] for t_r, t_f and C_l, respectively. Again the success criterion was 5% and generations consisted of 99 individuals. The algorithms remain the same when the number of dimensions go up. Like the case for two input parameters, we ran our process variations-effect analysis with same lower and upper bounds for input parameters for 5 generations of 99 individuals. At the end of the run, we filtered archive members for 70% equivalence value. The equivalence boundary without process variations is shown in Figure 6.7 and the final boundary after running process variations-effect analysis for 5 generations with equivalence value greater than 70% is shown in Figure 6.8.

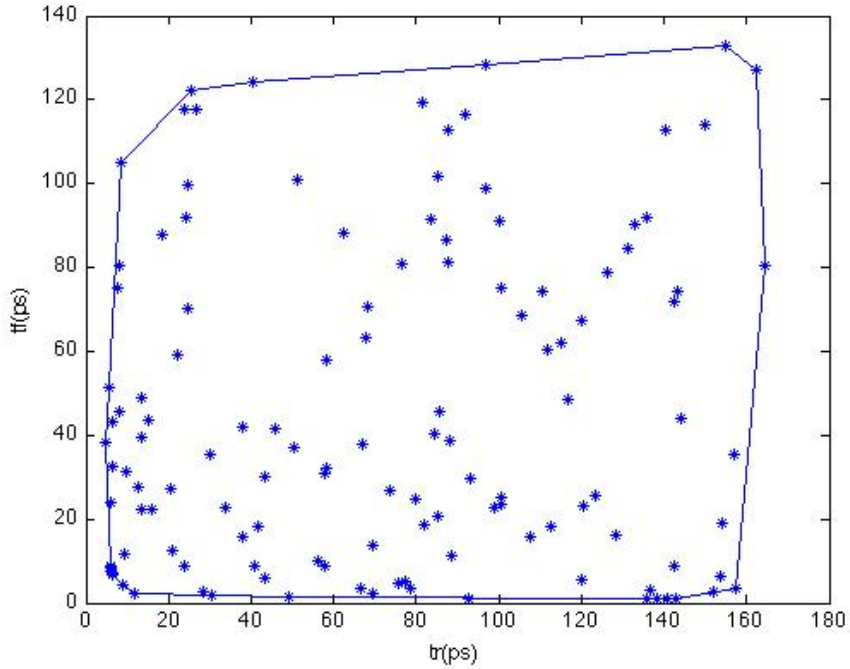

Figure 6.6. The archive for 5-generation run filtered for yield greater than or equal to 70%.

Table 6.2. Volume encapsulated in the equivalence boundary for inverter.

Equivalence	Original Case	60%	70%	80%	90%	100%
Volume	215090	38649	29032	15883	4180	1422

As it can be seen from Figures 6.7 and 6.8, volume contracts when process variations are taken into consideration. In Table 6.2, volumes encapsulated by the equivalence boundary are shown for different equivalence values. We observe that volume is decreasing with the increasing equivalence value, as expected.

In order to check our analysis, we determined 3 points from Figure 6.8. Those points were located at the center, on the boundary and on the outside with reference to the 70% equivalence boundary. For all those points, process variations-effect analysis was performed using 200 samples of Monte Carlo analysis. The points and their corresponding equivalence values are given in Table 6.3. As it can be seen from Table 6.3, the equivalence values validate our approach. The equivalence values increase from the outside point to the boundary point and from the boundary point to the inside point. Also the equivalence values for the points at the center and on the boundary

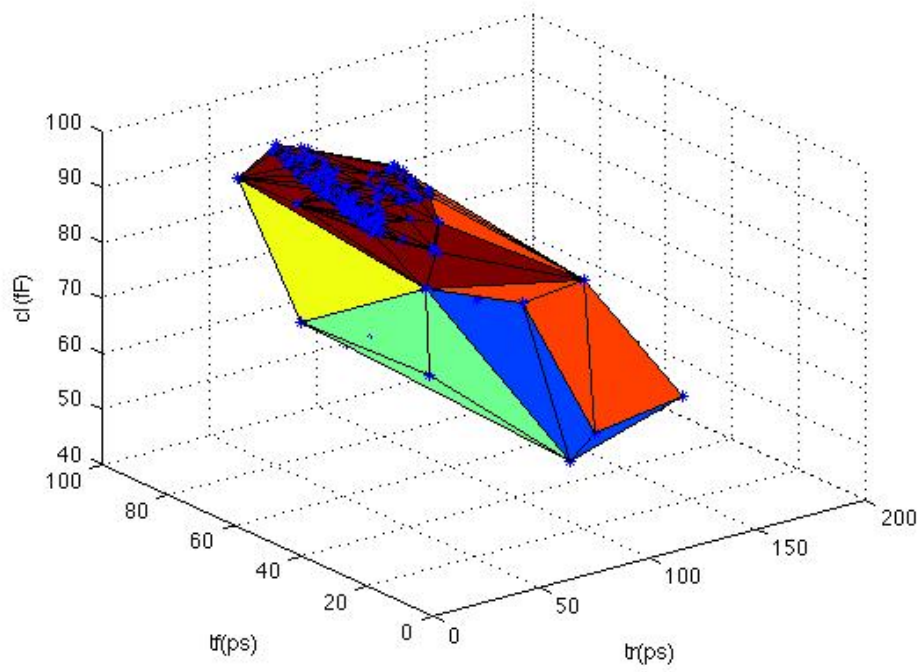

Figure 6.7. Boundary for inverter with population size 99 and generation number 20.

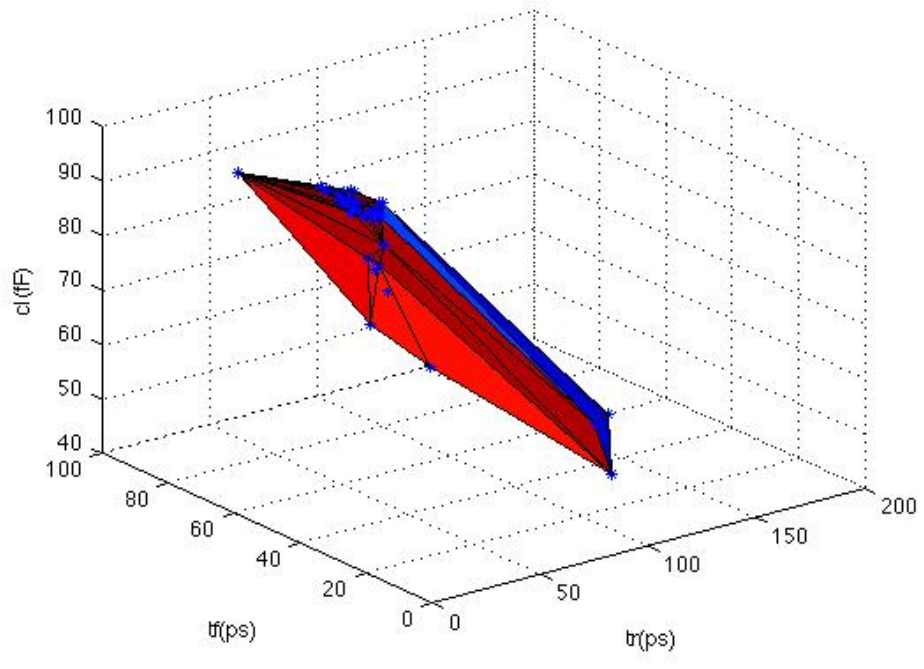

Figure 6.8. 70% equivalence boundary for inverter with population size 99 and generation Number 5.

Table 6.3. Locations of points at the center, on the boundary and on the outside for inverter.

location	t_r (ps)	t_f (ps)	c_l (fF)	*equivalence value* (%)
center	20	32	90	100
boundary	15.11	24.55	97.42	81
out	160	60	50	21.5

Table 6.4. Classification of 49 points on the 70% equivalence boundary according to percentage equivalence value for inverter.

$Interval\ for\ Equivalence\ Value$ (%)	$Number\ of\ boundary\ points$
$70 \leqslant Equivalence\ value < 75$	30
$75 \leqslant Equivalence\ value < 80$	9
$80 \leqslant Equivalence\ value < 85$	4
$85 \leqslant Equivalence\ value < 90$	1
$90 \leqslant Equivalence\ value \leqslant 100$	5

are greater than 70%. The equivalence value for the outside point is smaller than 70%. All these facts confirm our result in Figure 6.8.

In order to validate our methodology further, we performed process variations-effect analysis on all the equivalence boundary points using 200 samples of Monte Carlo analysis. There were 49 points on the 70% equivalence boundary. As the result of the analyses, the percentage level equivalence for each point was determined. The boundary points were classified according to the percentage level equivalence and they are tabulated in Table 6.4. As it can be seen from Table 6.4, most of the points have the equivalence values between 70% and 75%, which is inline with our expectations. There are some boundary points, which have higher equivalence values. This is acceptable, since an equivalence boundary point staying on the 70% equivalence boundary means that the percentage level equivalence value for that point is greater than or equal to 70%. The results of the analysis validate our methodology further.

6.3.2. Operational Amplifier (Opamp)

Now, we perform our analysis on a typical analog design, opamp. In order to generate the HSpice netlist for opamp, we utilized an in-house optimization tool [15]. We provided the performance targets to the tool, which generated the HSpice netlist for two stage operational transconductance amplifier in Figure 6.9.

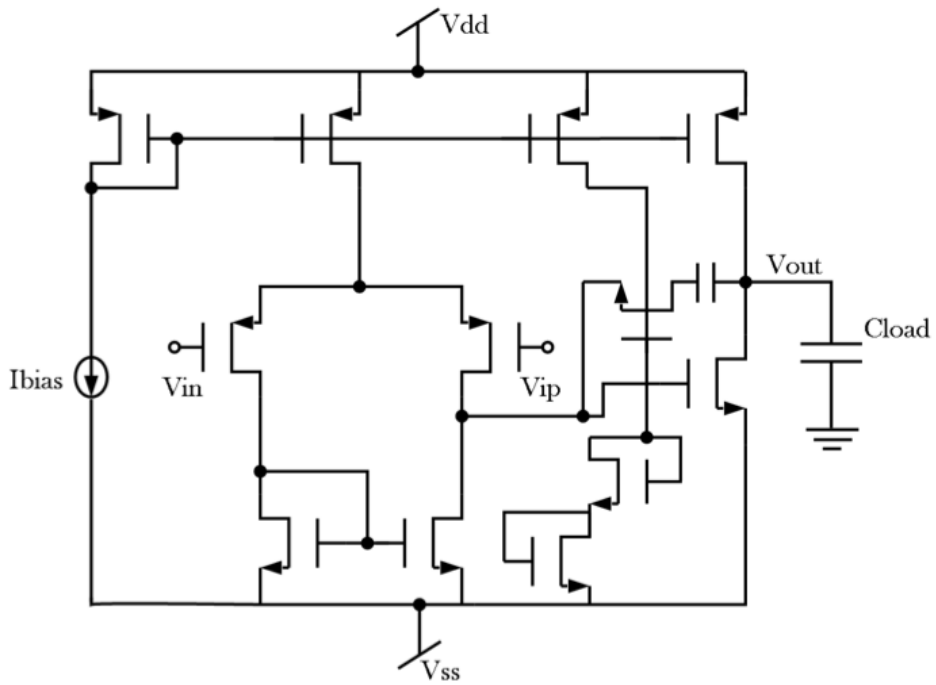

Figure 6.9. Circuit diagram for two stage operational transconductance amplifier [15].

For the Simulink model, we utilized the band-limited opamp in the Simscape library having the following parameters: *Gain*, R_{in}, R_{out}, V_{min}, V_{max}, V_{dot} (maximum slew rate), *Bandwidth* and *Initial Output Voltage* (V_0).

Characterization of the opamp necessitates multiple configurations, since the design has both time and frequency domain characteristics. A pulse waveform between input ports with feedback and a load capacitance on the output allows us to analyze the time domain characteristics. For the frequency domain, we replaced the pulse waveform with an AC sinusoidal waveform with adjustable frequency. The frequency domain configuration is depicted in Figure 6.10.

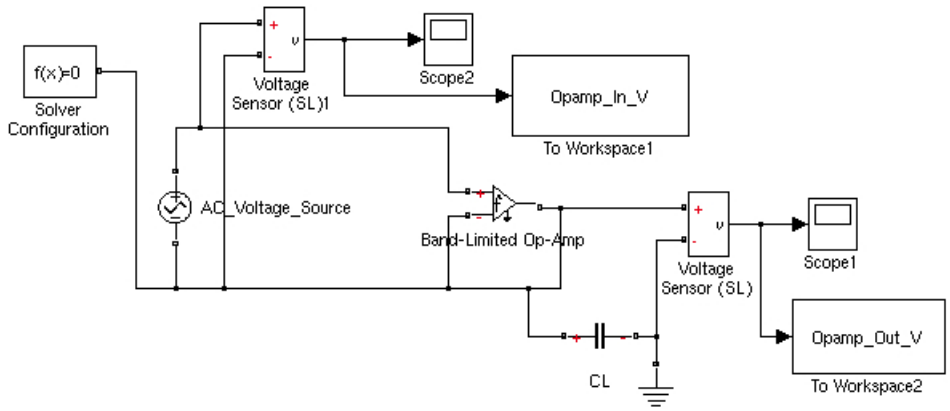

Figure 6.10. Operational amplifier configuration for frequency domain check.

Table 6.5. Parameters for opamp example.

Input Parameters (time domain)	Load Capacitance (C_l), Rise Time (t_r), Fall Time (t_f)
Input Parameters (frequency domain)	Load Capacitance (C_l)
Performance Parameters (time domain)	Maximum Output Voltage, Minimum Output Voltage, Slew Rate High to Low, Slew Rate Low to High, Fall Time, Rise Time
Performance Parameters (frequency domain)	Cut-off Frequency

We used the parameters in Table 6.5 as performance and input parameters, where the parameters for time and frequency domain configurations are listed separately. Different from the inverter case, application of the analysis requires executing multiple configurations. After execution of each configuration as a single step of the optimization process, we extracted the performance parameters related to each configuration. At the end of each step, we calculated the percentage errors between the performance parameters.

Similar to the inverter case, we found the equivalence boundary for opamp. We set the success criterion as 20%. We set lower and upper bounds for input parameters as [0.03 ms, 0.03 ms, 10 ms] and [0.3 ms, 0.3 ms, 200 pF] for t_r, t_f, and C_l, respectively. The number of individuals in the population was 99 and the number of generations

Table 6.6. Volume encapsulated in the equivalence boundary for opamp.

Equivalence	Original Case	60%	70%	80%	90%	100%
Volume	0.6919	0.6225	0.4696	0.4087	0.3288	0.2812

Table 6.7. Locations of points at the center, on the boundary and on the outside for opamp.

location	t_r (ms)	t_f (ms)	c_l (pF)	*equivalence value* (%)
center	0.15	0.18	75	95
boundary	0.1585	0.1857	87.77	74.5
out	0.2	0.1	60	0

was 30. Then, we included process variations in the analysis. It should be noted that for both configurations same process variation parameter values were used. We generated 40 samples for each boundary point. We applied our proposed methodology for 5 generations of 99 individuals with the same lower and upper bounds for input parameters. At the end of the run, archive members were filtered for 70% equivalence value. The equivalence boundaries before process variations-effect-analysis and after analysis are shown in Figures 6.11 and 6.12, respectively.

From the figures, we observe that process variation shrinks the equivalence boundary. Furthermore the boundary gets smaller with the increasing equivalence target as shown in Table 6.6.

As we did in the inverter example, we determined three points from Figure 6.12. Those points were located at the center, on the boundary and on the outside with reference to the 70% equivalence boundary. We ran our process variations-effect analysis for those points by generating 200 samples by Monte Carlo. The points and the corresponding equivalence values are given in Table 6.7. The equivalence values confirm our result in Figure 6.12.

As we did in the inverter example, we performed process variations-effect analysis on all the equivalence boundary points using 200 samples of Monte Carlo analysis.

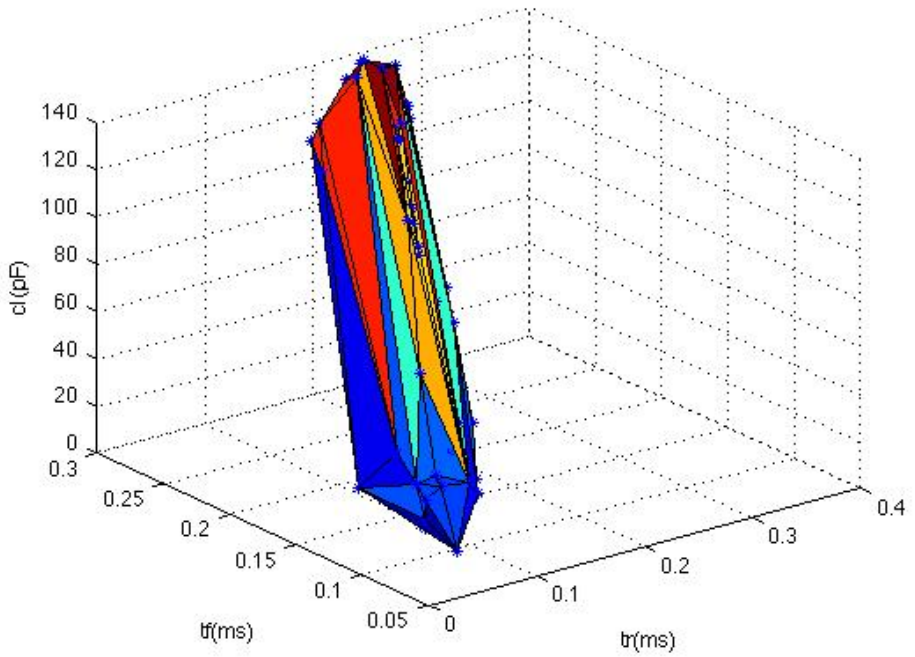

Figure 6.11. Boundary for opamp with population size 99 and generation number 30.

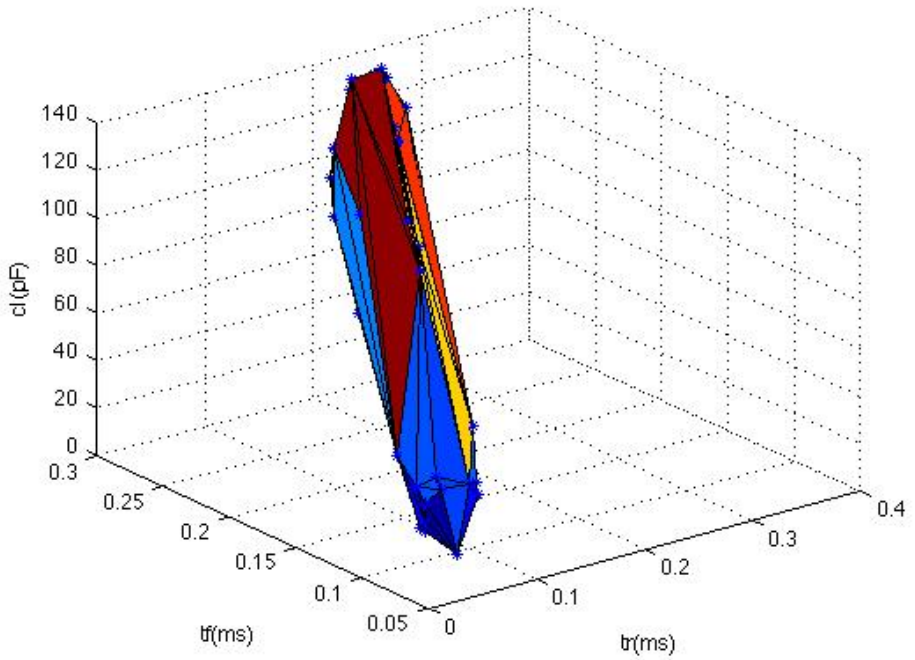

Figure 6.12. 70% equivalence boundary for opamp with population size 99 and generation number 5.

Table 6.8. Classification of 45 points on the 70% equivalence boundary according to percentage equivalence value for opamp.

$Interval\ for\ Equivalence\ Value\ (\%)$	$Number\ of\ boundary\ points$
$70 \leqslant Equivalence\ value < 75$	31
$75 \leqslant Equivalence\ value < 80$	5
$80 \leqslant Equivalence\ value < 85$	3
$85 \leqslant Equivalence\ value < 90$	5
$90 \leqslant Equivalence\ value \leqslant 100$	1

There were 45 points on the 70% equivalence boundary. The classification of the boundary points according to the percentage level equivalence is tabulated in Table 6.8. As it can be seen from Table 6.8, most of the points have the equivalence values between 70% and 75%, which is expected. The existence of boundary points having higher equivalence values is inline with our expectations, since an equivalence boundary point staying on the 70% equivalence boundary can have an equivalence value greater than or equal to 70%. Consequently, our methodology is validated further by this analysis.

6.3.3. Buck Converter

We performed our analysis on buck converter, which is commonly used in power management IC's in order to step down the voltage. In its basic form, the buck converter contains two switches (usually a diode and a transistor) controlling the current through an inductor. The inductor is in series with a parallel connected capacitor and load resistance.

For the Simulink model, we utilized components in the Simscape library. We used a switch as the transistor and pulse waveform for switching. Our Simulink model can be seen in Figure 6.13. In the Simscape library, various parasitic characteristics were integrated to the component models like switch on, diode on, inductor series and capacitor series resistances. This functionality makes it possible to analyze the effects of those parasitic components in the methodology.

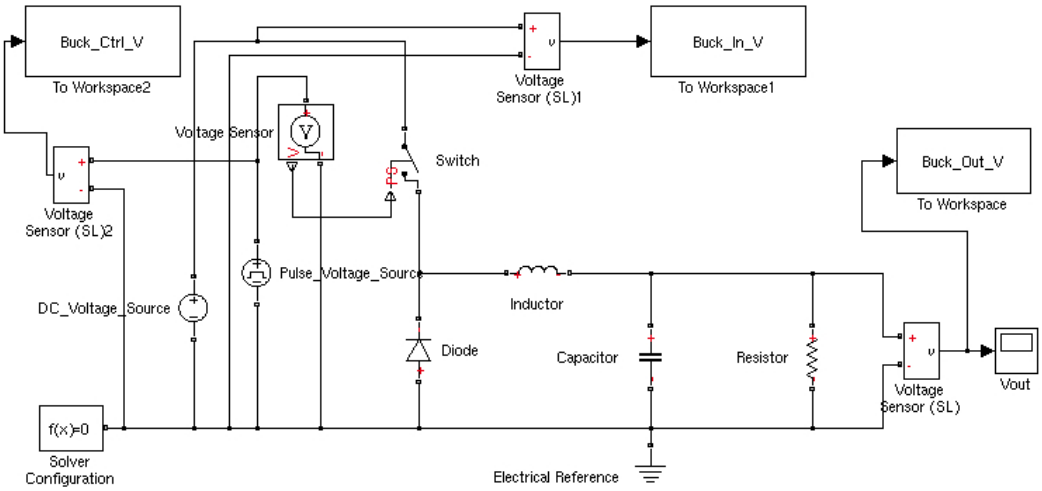

Figure 6.13. Simulink model of the buck converter.

In the circuit level implementation, we built the switch by means of a voltage controlled resistor having a resistance value varying according to the applied control voltage in a piecewise linear relationship. We added series inductor and capacitor resistances to the netlist for analysis. Our circuit topology for buck converter is shown in Figure 6.14.

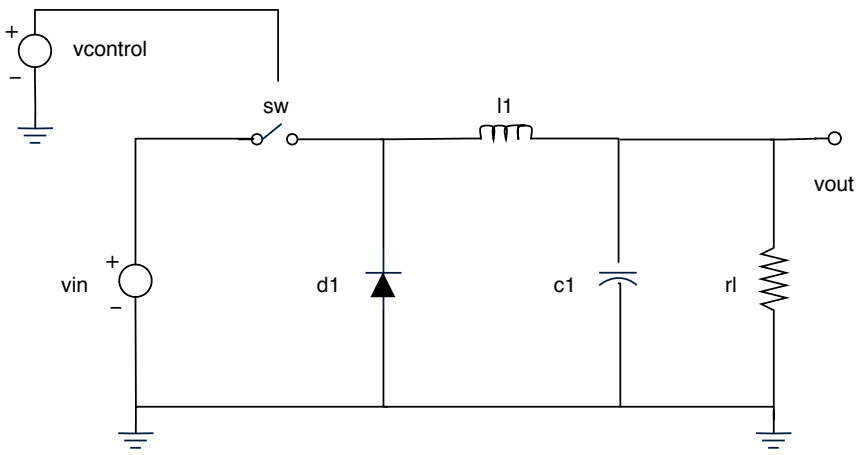

Figure 6.14. Circuit diagram for buck converter.

We used the parameters in Table 6.9 as input and performance parameters. It should be noted that there are two types of input parameters. The first type contains the major components affecting the functionality, whereas the second type includes the parasitic resistances as listed in Table 6.9. We note that inductor and capacitor resistances are series resistances. Also on-resistances of the switch and the diode were defined to be equal. For ease of demonstration, we limit the number of input param-

Table 6.9. Parameters for buck converter example.

Input Parameters (major components)	Input Voltage (v_{in}), Pulse Width of Controlling Voltage Source (pw), Load Resistance (r_l)
Input Parameters (parasitic resistances)	Inductor Series Resistance (r_{ind}), Switch-on Resistance (r_{sw}), Capacitor Series Resistance (r_{cap})
Performance Parameters	Maximum Output Voltage, Mean Output Voltage, Ripple Voltage

eters to three, to show the boundary of equivalence. However, the approach can be applied for more than three dimensions. Since the effects of major components on the performance parameters are more significant compared to minor components, we searched the boundary for major input parameters and fixed values of minor components to their optimal values. We use v_{in}, pw and r_l as input parameters in our experiments.

We found the equivalence boundary without process variations in the first step with the success criterion of 10%. We determined lower and upper bounds for input parameters as [50 ns, 1 V, 1 ohms] and [330 ns, 20 V, 2000 ohms] for pw, v_{in}, and r_l, respectively. The number of individuals in the population was 99 and the number of generations was 30. Then, we applied process variations. As the buck converter netlist in our case did not consist of transistors, we applied variations to the other components. To that end, we determined 7 parameters: inductor, capacitor, series-parasitic inductor resistance, series-parasitic capacitor resistance, two resistors utilized in building of the switch and the diode resistance. For their values, we determined maximum variation as 5%. We generated 40 samples for each boundary point and used 5% for normalization. We applied our proposed methodology for 5 generations of 99 individuals with the same lower and upper bounds for input parameters. At the end of the run, archive members were filtered for 70% equivalence value. The equivalence boundaries before and after the process variations-effect analysis are shown in Figures 6.15 and 6.16, respectively.

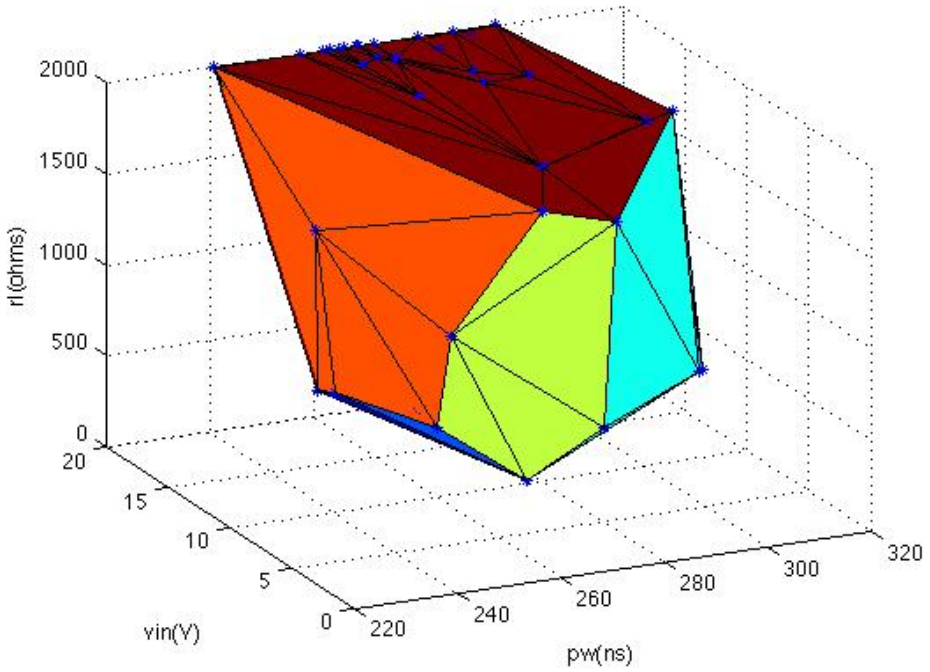

Figure 6.15. Boundary for buck converter with population size 99 and generation number 30.

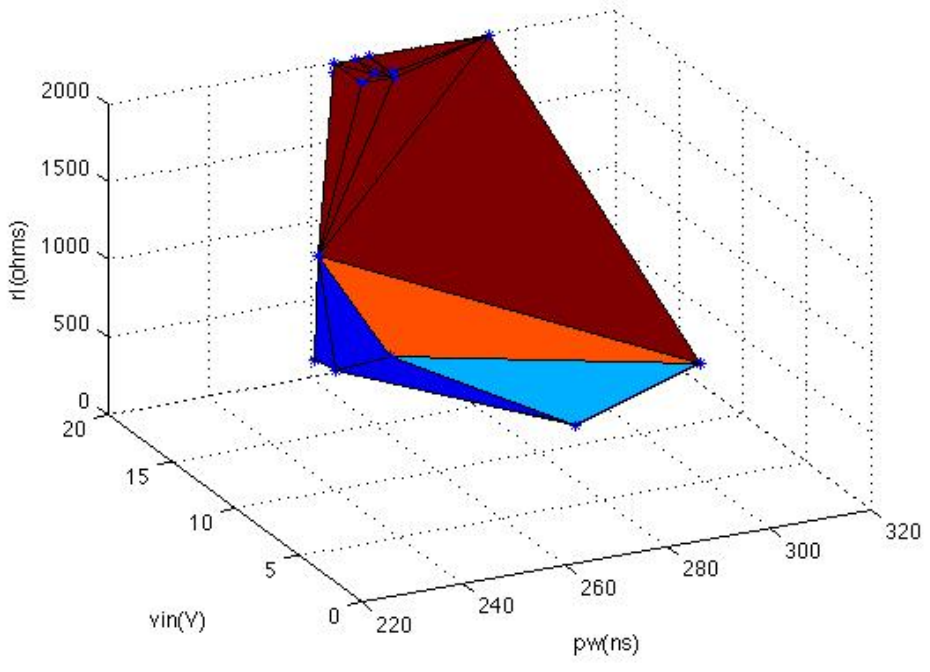

Figure 6.16. 70% equivalence boundary for buck converter with population size 99 and generation number 5.

Table 6.10. Volume encapsulated in the equivalence boundary for buck.

Equivalence	Original Case	60%	70%	80%	90%	95%	100%
Volume	144880	107510	59772	54196	13951	13059	0

Table 6.11. Locations of points at the center, on the boundary and on the outside for buck converter.

location	*pw* (ns)	v_{in} (V)	r_l $(ohms)$	*equivalence value* (%)
center	280	18	1000	78
boundary	301	5.704	711.7	70.5
out	250	4	400	57.5

It can be observed from the figures that the volume for equivalence boundary contracts when process variation effect is considered. The archive was filtered for different values. The volumes surrounded by the equivalence boundaries for different equivalence values are shown in Table 6.10. As it is observed from Table 6.10, volume is decreasing with the increasing equivalence value, as expected.

As we did for the previous experiments, we determined three points from Figure 6.16. The locations of those points with reference to the 70% equivalence boundary and the equivalence values for them are given in Table 6.11. Our process variations-effect analysis was run for those points by generating 200 samples by Monte Carlo. The equivalence values confirm our result in Figure 6.16 and validate our approach.

As we did in the previous examples, we performed process variations-effect analysis on all the equivalence boundary points using 200 samples of Monte Carlo analysis. There were 26 points on the 70% equivalence boundary. The classification of the boundary points according to the percentage level equivalence is tabulated in Table 6.12. Inline with our expectations, most of the points have the equivalence values between 70% and 75%, as it can be seen from Table 6.12. Similar to the previous experiments, there are boundary points, which have higher equivalence values. This is expected, as an equivalence boundary point staying on the 70% equivalence boundary can have an equivalence value greater than or equal to 70%. The results of the analysis

Table 6.12. Classification of 26 points on the 70% equivalence boundary according to percentage equivalence value for buck converter.

$Interval\ for\ Equivalence\ Value\ (\%)$	$Number\ of\ boundary\ points$
$70 \leqslant Equivalence\ value < 75$	12
$75 \leqslant Equivalence\ value < 80$	5
$80 \leqslant Equivalence\ value < 85$	6
$85 \leqslant Equivalence\ value < 90$	3
$90 \leqslant Equivalence\ value \leqslant 100$	0

for buck converter verify our methodology.

6.4. Discussion

For all the experiments, we observe that volumes encapsulated by the equivalence boundaries shrink, when the effects of process variations are taken into consideration. This result is inline with our expectations. Increase in equivalence threshold value results in a lower volume encapsulated by the equivalence boundary for all experiments. In all the experiments, we run our approach for 5 generations which consist of 99 individuals. We do this in order to be able to compare the results of experiments. We observe that 5 generations is enough for the evolutionary process to stabilize in most cases.

There are some differences between experiments. Inverter is a rather simple design and Simulink model for it was developed manually. Both of these facts are the reasons for selection of 5% success criterion. They also explain why the equivalence boundary search methodology in [35] is run for 20 generations, whereas for other experiments we run for 30 generations. For the opamp experiment, built-in Simulink model and in-house tool generated netlist are utilized by setting the same performance metrics. Considering this and the fact that opamp is a more complex design, setting of 20% success criterion is logical. The buck converter design is the most complex in all the experiments. However, we utilize low-level components in Simulink implementation. These facts allow us to set success criterion as 10%, which is between 5% and 20%.

In our technique, we utilize an evolutionary computation method, which is a modified version of SPEA2 [32]. Different methods can also be used, however we believe that utilization of evolutionary computation in AEC is significant.

Run times for the experiments for 5-generation process variations-effect analysis are as follows: i. inverter: 6 hours 44 minutes, ii. opamp: 4 hours 35 minutes, iii. buck converter: 34 hours 49 minutes. It should be noted that the proposed analysis will be applied to possible usage cases of analog design or model generation, where it will be run in the milestones of a project. Because of this fact, run times will not pose any harm to usage of the methodology. In order to show the impact of analyzing the effect of process variations, we want to present run times for each experiment for 5 generations without considering process variations. Run times for 5 generations without considering process variations are as follows: i. inverter: 24 minutes, ii. opamp: 53 minutes, iii. buck converter: 1 hour 38 minutes. For all the experiments, 40 samples were generated for each boundary point, so number of simulations increased 40 times. However, factors of increase in run times for process-variation effect analysis for each experiment are as follows: i. inverter: 16.8, ii. opamp: 5.2, iii. buck converter: 21. Increase in run times were smaller than 40, as a single Simulink simulation was run for one boundary point. The reason of the difference in the factor of opamp compared to other cases is the utilization of two Simulink configurations for opamp experiment. Due to that fact, overall run time for process variations-effect analysis is not dominated largely by the HSpice simulations for the opamp example, which was the case for the other two examples.

6.5. Conclusion

Equivalence checking (EC) is a crucial component of integrated circuit (IC) design. EC problem has become even more challenging with the ever-worsening process variations. In the research presented in the previous chapter [35], we researched optimization-based analog equivalence checking (AEC) between Simulink and HSpice models, where we proposed a methodology to find a boundary of equivalence. Although the significance of the effect of process variations is widely accepted, there is

limited number of studies addressing the impact of process variations on AEC. In this chapter, we propose a novel technique to incorporate process variations in AEC. We build a multi-objective optimization problem utilizing evolutionary computation. In this problem, we search for the boundary of equivalence both considering the equivalence value resulting from the effect of process variations and closeness to the boundary. In process variations-effect analysis, we utilize Quasi Monte Carlo (QMC) method to generate samples, which makes it possible to estimate the yield with fewer samples compared to Monte Carlo (MC) method. We generate process variations-aware equivalence boundaries for different equivalence values. We validated our analysis on three designs, an inverter, an operational amplifier, and a buck converter. Our approach proved to be a credible tool for investigating the effect of process variations on the equivalence boundary.

7. CONCLUSIONS AND FUTURE WORK

Simulink is a widely used system level design platform in the IC Design industry. Traditional workflow in IC Design projects is that the system is designed in Simulink and verified by Simulink simulations; then, RTL implementation is carried out and verified. In this workflow, a notion of formal equivalence between Simulink model and its HDL counterpart was missing. In the first part of the thesis, we proposed a methodology addressing that problem. Our methodology was exercised by standardized, publicly distributed complex designs and proven to be working. As our methodology is dependent on off-the-shelf tools, we are limited by the capabilities of such tools. Our methodology can potentially be adopted in IC design projects as we further investigate and improve our methodology.

As Simulink is also used in modeling Analog/Mixed Signal designs, extension of our methodology to check for equivalence of Analog/Mixed Signal models and their low level implementation counterparts was our next step. Methodologies for checking equivalence of analog models are at their infancy, and recent research mainly comes across the problem of state space explosion. Driven by that fact, we presented a semi-formal methodology which aims at bypassing the state-space explosion problem.

Our methodology in Chapter 4 checks the equivalence of Simulink and Spice designs by comparing *performance parameters* extracted from model and circuit level simulations. At first, optimal *input parameter* set characterizing the state of the inputs of the design is searched by means of an optimization function in Matlab. Secondly, objectives of the optimization problem are relaxed, in order to find ranges for each input parameter, where average of the errors for performance parameters stays below 10%. The result is an optimal input parameter set where both designs are similar below a given threshold for error percentage in performance parameters.

Our methodology has been tested on three experiments covering different fields of electronic design, as inverter is a basic building block of digital design, opamp is a

key block for analog design, and buck converter is a system level design both including switched and continuous behavior. The results of our experiments have shown that the methodology provides good outputs in a reasonable time.

Our methodology in Chapter 5 inherits the approach in Chapter 4 for checking the equivalence of Simulink and HSpice designs. It compares *performance parameters* extracted from simulations of these designs. Specifically, we used evolutionary computation to generate input parameters such that the error percentage between those performance parameters are below a certain threshold. According to this success criterion, we create a group of successful input parameter sets and determine the boundary represented by input parameter sets utilizing Matlab's built-in boundary function. By using crossover and mutation operations, we evolve the boundary to its final position. Consequently, we are able to converge to an equivalence boundary successfully in a much shorter run time compared to Monte Carlo method.

We validated our approach of applying evolutionary computation to find the equivalence boundary on three designs. These experiments were inverter, opamp and buck converter. The results of our experiments have shown that the methodology provides good outputs in a reasonable time.

As process technologies advance to decrease the feature size, nano-scale effects become a problem in IC design. Process variation is one of the most important nano-scale effects. Inline with that fact, in Chapter 6, we presented a semi-formal methodology analyzing the effect of process variations on the equivalence boundary utilizing evolutionary computation, QMC and multi-objective optimization.

As a result of our analysis of the effect of process variations on AEC, we generate an equivalence boundary. Furthermore, we investigate the impact of changing yield threshold values on the equivalence boundary by filtering the results of the computation for different yield values. We utilize QMC method in process variation, which makes it possible to estimate the yield with fewer samples compared to MC. In addition to yield value, we take closeness to the equivalence boundary into consideration by building a

multi-objective optimization problem where closeness to boundary is one objective. In our approach, we utilize an evolutionary computation algorithm, which is a modified version of SPEA2 [32]. We validate our approach on three designs, an inverter, an operational amplifier and a buck converter. We observe the effects of process variations and change in yield value on those designs.

As Simulink has widespread usage in industry and its library is extendible for specific needs, our approach can be applied to various analog designs. In addition to that, taking existence of similar studies for Simulink digital models into consideration [33], the analysis can be extended to mixed-signal designs. As future work, we plan to merge our methodologies for digital and analog designs. Furthermore, we utilize Simulink and HSpice only for validating the proposed approach. The overall approach is independent of them, any high level model and any circuit simulator can be used in the application of our approach. As another future work, we plan to handle hierarchical designs in equivalence checking. In order to speed up our analog equivalence checking methodology, we plan to develop heuristics to decrease the run time. In that future work, we plan to decrease the number of generations to run methodology by checking the volume expansion percentage for each additional run. We believe those future steps will enhance our methodology further.

REFERENCES

1. EETimes, *Sequential Equivalence Checking*, 2010, `http://www.eetimes.com/design/eda-design/4004118/Sequential-equivalence-checking-for-RTL-models`, accessed at December 2017.

2. Vasudevan, S., J. Abraham, V. Viswanath and J. Tu, "Automatic decomposition for sequential equivalence checking of system level and RTL descriptions", *Design Automation For Embedded Systems*, 2006.

3. Markovic, D., C. Chang, B. Richards, H. So, B. Nikolic and R. Brodersen, "ASIC Design and Verification in an FPGA Environment", *Custom Integrated Circuits Conference (CICC)*, 2007.

4. Araki, D., A. Nakamura and M. Miyama, "Model-based SoC design using ESL environment", *International System on Chip Design Conference (SOCC)*, 2010.

5. Ryabtsev, M. and O. Strichman, "Translation validation: From Simulink to C", *Computer Aided Verification (CAV)*, Vol. 5643, 2009.

6. Brillout, A., N. He, M. Mazzucchi, D. Kroening, M. Purandare, P. Rummer and G. Weissenbacher, "Mutation-based Test Case Generation for Simulink Models", *Formal Methods for Components and Objects*, 2010.

7. Steinhorst, S., *Formal Verification Methodologies for Nonlinear Analog Circuits*, PhD Dissertation, der Johann Wolfgang Goethe-Universitat in Frankfurt am Main, 2011.

8. Hartong, W., R. Klausen and L. Hedrich, "Formal verification for nonlinear analog systems: Approaches to model and equivalence checking", *Advanced Formal Verification*, 2004.

9. Steinhorst, S. and L. Hedrich, "Advanced methods for equivalence checking of analog circuits with strong nonlinearities", *Formal Methods in System Design*, Vol. 36, 2010.

10. Hedrich, L. and E. Barke, "A formal approach to nonlinear analog circuit verification", *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 1995.

11. Salem, A., "Semi-formal verification of VHDL-AMS descriptions", *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2002.

12. Singh, A. and P. Li, "On Behavioral Model Equivalence Checking for Large Analog/Mixed Signal Systems", *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2010.

13. Doorn, T., E. ter Maten, J. Croon, A. D. Bucchianico and O. Wittich, "Importance Sampling Monte Carlo simulations for accurate estimation of SRAM yield", *European Solid State Circuits Conference (ESSCIRC)*, 2008.

14. Hakhamaneshi, B., *A Hardware Implementation of the Advanced Encryption Standard Using SystemVerilog*, M.Sc. Project Report, California State University Computer Engineering Department, 2009.

15. Afacan, E., G. Berkol, G. Dundar, A. Pusane and F. Baskaya, "A Deterministic Aging Simulator and An Analog Circuit Sizing Tool Robust to Aging Phenomena", *International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, 2015.

16. Martin, G., *The Myths of EDA - the 70% Rule*, 2008, `http://www.chipdesignmag.com/martins/2008/11/27/the-myths-of-eda-the-70-rule/`, accessed at December 2017.

17. Auger, D., "Programmable Hardware Systems Using Model-Based Design", *IET*

and Electronics Weekly Conference on Programmable Hardware Systems, 2008.

18. Vardi, M., "Formal Techniques for SystemC Verification", *ACM/IEEE Design Automation Conference (DAC)*, 2007.

19. Gajski, D., T. Austin and S. Svoboda, "What input-language is the best choice for high level synthesis (HLS)?", *ACM/IEEE Design Automation Conference (DAC)*, 2010.

20. Mathur, A. and V. Krishnaswamy, "Design for Verification in System-level Models and RTL", *ACM/IEEE Design Automation Conference (DAC)*, 2007.

21. Moon, I., P. Bjesse and C. Pixley, "A Compositional Approach to the Combination of Combinational and Sequential Equivalence Checking of Circuits Without Known Reset States", *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2007.

22. Clarke, E. and D. Kroening, "Hardware verification using ANSI-C programs as a reference", *Asia and South Pacific Design Automation Conference (ASPDAC)*, 2003.

23. Clarke, E., D. Kroening and K. Yorav, "Behavioral consistency of C and Verilog programs using bounded model checking", *Design Automation Conference (DAC)*, 2003.

24. Clarke, E., D. Kroening and K. Yorav, *Behavioral consistency of C and Verilog programs using Bounded Model Checking*, Technical Report, Carnegie Mellon University School of Computer Science, 2003.

25. Huang, W., P. Tang and M. Ding, "Sequential equivalence checking using cuts", *Asia and South Pacific Design Automation Conference (ASPDAC)*, 2005.

26. Eijk, C., "Sequential Equivalence Checking Based on Structural Similarities",

IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 2000.

27. Haldar, M., G. Singh, S. Prabhakar, B. Dwivedi and A. Ghosh, "Construction of concrete verification models from C++", *ACM/IEEE Design Automation Conference (DAC)*, 2008.

28. Vasudevan, S., J. Abraham, V. Viswanath and J. Tu, "Automatic decomposition for sequential equivalence checking of system level and RTL descriptions", *ACM and IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE)*, 2006.

29. Lim, B., J. Kim and M. Horowitz, "An Efficient Test Vector Generation for Checking Analog/Mixed-Signal Functional Models", *Design Automation Conference (DAC)*, 2010.

30. Horowitz, M., M. Jeeradit, F. Lau, S. Liao, B. C. Lim and J. Mao, "Fortifying Analog Models with Equivalence Checking and Coverage Analysis", *Design Automation Conference (DAC)*, 2010.

31. MathWorks, *Optimization Toolbox*, 2010, `https://www.mathworks.com/products/optimization.html`, accessed at December 2017.

32. Zitzler, E., M. Laumanns and L. Thiele, "SPEA2: Improving the strength Pareto evolutionary algorithm", *Swiss Federal Institute of Technology (ETH) Zurich Computer Engineering and Networks Laboratory Report*, 2001.

33. Saglamdemir, M. O., A. Sen and G. Dundar, "A Formal Equivalence Checking Methodology for Simulink and Register Transfer Level Designs", *International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, 2012.

34. Saglamdemir, M., G. Dundar and A. Sen, "An Analog Behavioral Equivalence

Checking Methodology for Simulink Models and Circuit Level Designs", *International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, 2015.

35. Saglamdemir, M., G. Berkol, G. Dundar and A. Sen, "An Analog Behavioral Equivalence Boundary Search Methodology for Simulink Models and Circuit Level Designs Utilizing Evolutionary Computation", *Integration, The VLSI Journal*, Vol. 55, 2016.

36. Saglamdemir, M., G. Dundar and A. Sen, "Analog Behavioral Equivalence Boundary Computation under the Effect of Process Variations", *Integration, The VLSI Journal*, Vol. 61, 2017.

37. Roth, J. P., "Systematic Design of Automata", *American Federation of Information Processing Societies Conference (AFIPS)*, 1965.

38. Smith, G. L., R. J. Bahnsen and H. Halliwell, "Boolean comparison of hardware and flowcharts", *IBM Journal of Research and Development*, 1982.

39. Synopsys, *Formality*, 2010, `http://www.synopsys.com/tools/verification/formalequivalence/pages/formality.aspx`, accessed at December 2017.

40. Pixley, C., "A theory and implementation of sequential hardware equivalence", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 1992.

41. Iyer, M. K., G. Parthasarathy and K. Cheng, "SATORI - A Fast Sequential SAT Engine for Circuits", *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2003.

42. Lu, F., M. Iyer, G. Parthasarathy, L. Wang, K. Cheng and K. Chen, "An efficient sequential SAT solver with improved search strategies", *Design, Automation and Test in Europe (DATE)*, 2005.

43. Bombieri, N., F. Fummi, G. Pravadelli and J. Marques-Silva, "Towards Equivalence Checking Between TLM and RTL Models", *IEEE/ACM International Conference on Formal Methods and Models for Co-design (MEMOCODE)*, 2007.

44. MathWorks, *Simulink*, 2010, `http://www.mathworks.com/products/simulink/`, accessed at December 2017.

45. Song, M. and Y. Chung, "SoC Design of Speaker Connection System by Efficient Cosimulation", *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, 2006.

46. Zhou, C. and R. Kumar, "Modeling Simulink Diagrams using Input/Output Extended Finite Automata", *IEEE International Computer Software and Applications Conference (COMPSAC)*, 2009.

47. Ray, R., *Automated Translation of MATLAB Simulink/Stateflow Models to an Intermediate Format in HyVisual*, M.Sc. Project Report, Chennai Mathematical Institute, 2007.

48. Agrawal, A., G. Simon and G. Karsai, "Semantic Translation of Simulink/Stateflow models to Hybrid Automata", *Electronic Notes in Theoretical Computer Science (ENTCS)*, Vol. 109, 2004.

49. Conrad, M., "Testing-based translation validation of generated code in the context of IEC 61508", *Formal Methods In System Design*, 2009.

50. Pnueli, A., "The temporal logic of programs", *IEEE Symposium on the Foundations of Computer Science (FOCS)*, 1977.

51. Clarke, E. and E. Emerson, "Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic", *In Logic of Programs, Lecture Notes in Computer Science*, 1982.

52. Foster, H., E. Marschner and Y. Wolfsthal, *IEEE 1850 PSL: The Next Generation*, 2005, `http://www.pslsugar.org/papers/ieee1850psl-the_next_generation.pdf`, accessed at December 2017.

53. Nickovic, D. and O. Maler, "A property-based monitoring tool for analog systems", *Lecture Notes in Computer Science*, 2007.

54. Maler, O., D. Nickovic and A. Pnueli, "Checking temporal properties of discrete, timed and continuous behaviors", *Pillars of Computer Science, Lecture Notes in Computer Science*, 2008.

55. Jones, K., V. Konrad and D. Nickovic, "Analog property checkers: a ddr2 case study", *Formal Methods in System Design*, 2010.

56. Hartong, W., L. Hedrich and E. Barke, "Model checking algorithms for analog verification", *ACM/IEEE Design Automation Conference (DAC)*, 2002.

57. Hartong, W., L. Hedrich and E. Barke, "On discrete modeling and model checking for nonlinear analog systems", *International Conference on Computer Aided Verification (CAV)*, 2002.

58. Dastidar, T. and P. Chakrabarti, "A Verification System for Transient Response of Analog Circuits Using Model Checking", *International Conference on VLSI Design (VLSID)*, 2005.

59. Dastidar, T. and P. Chakrabarti, "A verification system for transient response of analog circuits", *ACM Transaction on Design Automation of Electronic Systems (TODAES)*, 2007.

60. Grabowski, D., D. Platte, L. Hedrich and E. Barke, "Time Constrained Verification of Analog Circuits using Model-Checking Algorithms", *Workshop on Formal Verification of Analog Circuits (FAC)*, 2005.

61. Little, S., N. Seegmiller, D. Walter, C. Myers and T. Yoneda, "Verification of analog/mixed-signal circuits using labeled hybrid petri nets", *IEEE/ACM International Conference on Computer-aided design (ICCAD)*, 2006.

62. Myers, C., R. Harrison, D. Walter, N. Seegmiller and S. Little, "The case for analog circuit verification", *Electronic Notes in Theoretical Computer Science (ENTCS)*, 2006.

63. Zaki, M., S. Tahar and G. Bois, "Formal Verification of analog and mixed signal designs: A survey", *Microelectronics Journal*, 2008.

64. Balivada, A., Y. Hoskote and J. Abraham, "Verification of transient response of linear analog circuits", *IEEE VLSI Test Symposium (VTS)*, 1995.

65. Seshadri, S. and J. Abraham, "Frequency response verification of analog circuits using global optimization techniques", *Journal of Electronic Testing*, Vol. 17, 2001.

66. Hedrich, L. and E. Barke, "A formal approach to verification of linear analog circuits with parameter tolerances", *IEEE/ACM Design, Automation and Test in Europe (DATE)*, 1998.

67. Liu, B., J. Messaoudi and G. Gielen, "A Fast Analog Circuit Yield Estimation Method for Medium and High Dimensional Problems", *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2012.

68. Singhee, A. and R. A. Rutenbar, "Why Quasi-Monte Carlo is better than Monte Carlo or Latin Hypercube Sampling for Statistical Circuit Analysis", *IEEE Transactions on Computer Aided Design (TCAD)*, 2010.

69. Liu, B., F. Fernandez and G. Gielen, "Efficient and Accurate Statistical Analog Yield Optimization and Variation-Aware Circuit Sizing based on Computational Intelligence Techniques", *IEEE Transactions on Computer Aided Design (TCAD)*, 2011.

70. Graeb, H., *Analog Design Centering and Sizing*, Springer, 2007.

71. McConaghy, T. and G. Gielen, "Globally Reliable Variation-Aware Sizing of Analog Integrated Circuits via Response Surfaces and Structural Homotopy", *IEEE Transactions on Computer Aided Design (TCAD)*, 2009.

72. Owenetal, A., "Latin Super Cube Sampling for Very High Dimensional Simulations", *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 1998.

73. Singhee, A., S. Singhal and R. A. Rutenbar, "Practical, Fast Monte Carlo Statistical Static Timing Analysis: Why and How", *International Conference on Computer Aided Design (ICCAD)*, 2008.

74. Bhardwaj, S., P. Ghanta and S. Vrudhula, "A framework for statistical timing analysis using non-linear delay and slew models", *International Conference on Computer Aided Design (ICCAD)*, 2006.

75. Singhee, A., S. Singhal and R. A. Rutenbar, "Exploiting correlation kernels for efficient handling of intra-die spatial correlation, with application to statistical timing", *Design, Automation and Test in Europe (DATE)*, 2008.

76. Singhee, A. and R. A. Rutenbar, "From Finance to Flip-Flops: A Study of Fast Quasi-Monte Carlo Methods from Computational Finance Applied to Statistical Circuit Analysis", *International Symposium on Quality Electronic Design (ISQED)*, 2007.

77. Narayanan, R., B. Akbarpour, M. Zaki and L. P. S. Tahar, "Formal Verification of Analog Circuits in the Presence of Noise and Process Variation", *Design, Automation and Test in Europe Conference (DATE)*, 2010.

78. Narayanan, R., M. Zaki and L. P. S. Tahar, "Ensuring Correctness of Analog Circuits in Presence of Noise and Process Variations Using Pattern Matching",

Design, Automation and Test in Europe Conference (DATE), 2011.

79. Urard, P., A. Maalej, R. Guizzetti, N. Chawla and V. Krishnaswamy, "Leveraging sequential equivalence checking to enable system-level to RTL flows", *ACM/IEEE Design Automation Conference (DAC)*, 2008.

80. Aziz, S. and S. Sharma, "New Methodologies for High Level Modeling and Synthesis of Low Density Parity Check Decoders", *International Conference on Computer and Information Technology (CIT)*, 2008.

81. MentorGraphics, *Catapult*, 2010, `http://www.mentor.com/esl/catapult/overview`, accessed at December 2017.

82. AES, "Advanced Encryption Standard", *Federal Information Processing Standards (FIPS) Publication 197*, 2001.

83. Rabaey, J. M., *Digital Integrated Circuits: A Design Perspective*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

84. Guerra-Gomez, I. and E. Tlelo-Cuautle, "Sizing Analog Integrated Circuits by Current-Branches-Bias Assignments with Heuristics", *Elektronika IR Electrotechnika*, 2013.

85. Maaranen, H., K. Miettinen and M. Makela, "Quasi-random initial population for genetic algorithms", *Computers and Mathematics with Applications*, 2004.

86. Sonmez, O. and G. Dundar, "Simulation-based analog and RF circuit synthesis using a modified evolutionary strategies algorithm", *Integration, the VLSI Journal*, Vol. 44, 2011.

87. Pelgrom, M., A. Duinmaijer and A. Welbersetal, "Matching Properties of MOS Transistors", *IEEE Journal of Solid-State Circuits (JSSC)*, Vol. 24, 1989.

88. Kinget, P., "Device Mismatch and Tradeoffs in the Design of Analog Circuits",

IEEE Journal of Solid-State Circuits (JSSC), Vol. 40, 2005.