University of Southampton
Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science

Architectural Synthesis of Analogue Filters from Behavioural VHDL-AMS Descriptions

by

Fazrena Azlee Hamid

A thesis submitted for the degree of
Doctor of Philosophy

March 2004

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE & MATHEMATICS

SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

ARCHITECTURAL SYNTHESIS OF ANALOGUE FILTERS FROM BEHAVIOURAL VHDL-AMS DESCRIPTIONS

by Fazrena Azlee Hamid

There is an increasing need for efficient synthesis techniques to support mixed-signal application-specific integrated circuit (ASIC) designs. While digital synthesis from VHDL is already well established the development of corresponding analogue and mixed-signal synthesis methodologies is still lagging. An application area that would greatly benefit from such development is analogue filtering, especially in integrated high-frequency implementations. Hence, the primary aim of this research is to investigate and develop techniques for VHDL-AMS-based synthesis of analogue filters suitable for use in mixed-signal ASICs where behavioural models are especially important. Particular emphasis is put on architectural optimisation with the aim to identify the most suitable circuit topologies.

The novel contributions can be briefly summarised as follows. New methods have been presented to extract synthesisable VHDL-AMS constructs from behavioural filter models using parse trees. They can be extended to support a more general, mixed-signal synthesis system based on VHDL-AMS. An effective architectural optimisation engine for analogue filter synthesis has been developed to minimise transfer function accuracy errors and power consumption. The engine is based on three-tier architectural and parametric optimisation, in which a combination of heuristic and systematic search algorithms provides a possibility of global optimisation. The parametric optimiser relies on full HSPICE simulation to ensure accurate circuit performance evaluation.

The new methods are implemented in a demonstrator system named FIST (Filter Synthesis Tool), and were successfully applied to several case studies of 1 GHz integrated analogue filter circuits designed for implementation in a 0.35μm CMOS technology.

The method of using parse trees to extract circuit-level structures from high-level descriptions proved to be effective. It was shown that parse trees allow for easy detection of synthesisable constructs and support recursive static evaluations of expressions necessary in calculation of filter parameters. This technique provides a groundwork from which more general VHDL-AMS-based synthesis systems can be developed.

Contents

Contents 3

Acknowledgement 6

Abbreviations 7

1 Introduction 9

1.1 Research motivation and aims 10

1.2 Dissertation contents and organisation 11

2 State of the Art 12

2.1 Synthesis from analogue HDL 13

2.1.1 KANDIS (1995) 14

2.1.2 VASE (1997) 15

2.1.3 NEUSYS (2002) 16

2.2 Analogue synthesis 17

2.2.1 Knowledge-based tools 21

2.2.2 Optimisation-based tools 24

2.3 Analogue filter synthesis 33

3 Behavioural Synthesis of Analogue Filters: from VHDL-AMS description to filter information 35

3.1 VHDL-AMS 35

3.2 VHDL-AMS for synthesis 36

3.2.1 Features suitable for synthesis 36

3.3 Modelling synthesisable analogue filters using VHDL-AMS 38

3.3.1 Time-domain modelling with DAE construct 39

3.3.2 Frequency-domain modelling with LTF construct 40

3.4 VHDL-AMS parse tree 42

3.4.1 Parser 43

3.5 Software modules for behavioural synthesis of analogue filters from VHDL-AMS parse trees 46

3.5.1 Synthesis syntax checker 48

3.5.2 Static calculator 56
3.6 Concluding remarks 68
4 Behavioural Synthesis of Analogue Filters: from High-Level Filter Specification to Filter Structure 70
4.1 Synthesis procedures 71
4.1.1 Root finding 73
4.1.2 Filter cell mapping 74
4.1.3 Cell realisability check 77
4.1.4 Construction of performance model 79
4.1.5 Topology selection 79
4.2 Comparative study between synthesis procedures 80
4.3 Architectural and parametric optimisation 85
4.3.1 Architectural optimisation 85
4.3.2 Parametric optimisation 86
4.3.3 Cost function formulation and evaluation 106
4.4 Concluding remarks 118
5 Three-tier Optimisation Algorithm 120
5.1 Stochastic search 120
5.2 Downhill simplex optimisation 121
5.3 HSPICE built-in optimiser 124
5.4 The three-tier algorithm 127
5.4.1 Downhill simplex function evaluation 127
5.4.2 Three-tier description 130
5.4.3 Three-tier optimisation versus one-tier algorithms 137
5.5 FIST – Filter synthesis tool 138
5.5.1 Windows GUI environment 140
5.6 Concluding remarks 142
6 Practical Experiments with Two Case Studies 143
6.1 Case Study 3: Synthesis of an analogue fourth-order 1GHz lowpass filter 144
6.1.1 Performance models 144
6.1.2 Fourth-order lowpass analogue filter candidates 146
6.1.3 Architectural optimisation for Case Study 3 152
6.1.4 Results 154
6.2 Case Study 4: Synthesis of an analogue fourth-order 1 GHz bandpass filter 158

6.2.1 Performance model ... 158
6.2.2 Fourth-order bandpass analogue filter candidates ... 159
6.2.3 Architectural optimisation for Case Study 4 ... 163
6.2.4 Results ... 163
6.3 Application examples of topology Cascade 1 ... 166
6.3.1 Example 1: Chebyshev 0.5 db ripple, cut off at 1 GHz (Gain=1) ... 167
6.3.2 Example 2: Chebyshev 3 db ripple, cut off at 0.5 GHz (Gain =1) ... 168
6.3.3 Example 3: Butterworth, cut off at 0.5 GHz (Gain = 1) ... 168
6.3.4 Experimental results ... 169
6.4 Concluding remarks ... 172
7 Conclusions and Future Work ... 173
Appendix A: Analogue filter netlists for Case Studies 3 and 4 ... 176
A.1 Cascade 1 ... 176
A.2 Cascade 2 ... 176
A.3 Cascade 3 ... 177
A.4 Cascade 4 ... 178
A.5 Cascade 5 ... 178
A.6 Cascade 6 ... 179
A.7 Coupled resonator ... 180
A.8 IFLF1 ... 181
A.9 IFLF2 ... 181
A.10 IFLF3 ... 182
A.11 LC-OTA-C ... 183
A.12 LF1 ... 184
A.13 LF2 ... 184
A.14 LF3 ... 185
A.15 Vertical Cascode ... 186
Appendix B: Alcatel CMOS 0.35μm BSIM 3v3 transistor models ... 187
B.1 NMOS transistor model ... 187
B.2 PMOS transistor model ... 188
References ... 189

Acknowledgement

I wish to extend my deepest gratitude to my supervisor, Dr. Tom Kazmierski, for his guidance, assistance and support during the long course of this research. I also appreciate the suggestions and comments from Dr. Bashir Al-Hashimi who served on my MPhil to PhD transfer examination. To my loving husband, Ahmad Kamsani and to my wonderful and precious sons, Hakeem and Hamzah: I am truly blessed to have you in my life. I am also grateful to have such supportive and inspiring friends; they are like my own brothers and sisters – you know who you are. My mother and father have always been the strong pillars of my life; no words can describe my love and appreciation to both of you. I would also like to dedicate a warm thank you to my siblings (I know you can do it!), my mother and father in-laws and my other relatives. Above all, I humbly acknowledge The All-Mighty Creator; all the good things in my life are due to His infinite Mercy and Grace. *Alhamdulillahirabbil 'aalamiin* – all praises to Allah, the Lord of the worlds.

Abbreviations

(in)Q	in-mode (input) Quantity
(out)Q	output Quantity
AB	Architecture Body
AD	Attribute Designator
ADP	Architecture Declarative Part
Ag.	Aggregate cluster
AN	Attribute Name
AS	Architecture Statement
ASIC	Application Specific Integrated Circuit
ASP	Architecture Statement Part
BDI	Block Declarative Item
CAD	Computer-Aided Design
CD	Constant Declaration
DAE	Differential Algebraic Equation
EDA	Electronic Design Automation
FIST	FIlter Synthesis System
FPAA	Field Programmable Analogue Array
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
IFLF	Inverse-Follow-the-Leader-Feedback
IL	Identifier List
LF	Leap Frog
LHS	Left Hand Side
LRM	Language Reference Manual
LTF	Laplace Transfer Function
MEMS	Micro-Electro-Mechanical Systems
OTA	Operational Transconductance Amplifier
OTA-C	Operational Transconductance Amplifier-Capacitor
PLL	Phase Locked Loop

RHS	Right Hand Side
SE	Simple Expression
SFG	Signal Flow Graphs
SI	Subtype Indicator
SN	Simple Name
SOC	System-On-Chip
SS	Simultaneous Statement
SSS	Simple Simultaneous Statement
TM	Type Mark
VHDL	VHSIC (Very High Speed Integrated Circuit) Hardware Description Language
VHDL-AMS	VHDL (extension to) Analogue and Mixed-Signal (informal name to IEEE Standard 1076.1-1999)

1 Introduction

The need for new analogue synthesis techniques that would facilitate the development of mixed-signal CAD tools is increasing coupled with the advancement in semiconductor technology and the demand from commerce, especially the wireless mobile communication industry. Although usually comprising a small part of a mixed analogue-digital system, the analogue part of an application-specific integrated circuit (ASIC) often causes the bottleneck in design time and effort. The development of appropriate synthesis methodologies to support mixed-signal ASIC designs is still lagging. One of the obvious difficulties has been the absence of a mixed-signal high-level hardware description language commonly accepted as a standard throughout the CAD industry. However, the emergence of VHDL-AMS [1] as a standardised description language for mixed-systems, provides a stimulus for the development of analogue and mixed-signal design automation.

This research aims to narrow the gap between analogue and digital system design automation by developing methods and tools for analogue synthesis, and by utilising VHDL-AMS as the input specification. VHDL-AMS, a superset of VHDL, supports the modelling of analogue systems, i.e. dynamic systems that exhibit continuous behaviour in both time and amplitude. The ability to represent analogue systems on behavioural level is an important feature as the behavioural model is the most abstract representation of a specification. Behavioural models offer two main advantages in terms of automated synthesis. Firstly, the synthesis process is characterised by a high degree of freedom to choose or generate the hardware that implements the specification. Secondly, a top-down hierarchical design strategy may be applied to suit the needs of the complex mixed-signal design process. Also, the wide use and acceptance of VHDL for digital synthesis should stimulate the pursuit of new synthesis methods using VHDL-AMS for analogue synthesis, so that both digital and analogue synthesis techniques could be combined and synergised in a VHDL-AMS-based mixed-signal CAD environment. The use of VHDL-AMS for analogue synthesis makes sense as a mixed-signal design suite would be built upon the strength and success of its digital counterpart. VHDL-AMS-based synthesis – despite its potential in both analogue and mixed-signal design - is still in its infancy. The need to investigate how this potential could be explored becomes therefore the primary motivation for this research.

The techniques being investigated in this research are aimed at a very important class of circuits used in mixed-signal ASICs, namely analogue filters. Since many high-performance

applications use analogue signal conditioning blocks, there is an increasing demand for high-frequency analogue filters embedded in large, mixed-signal designs. Analogue filters are instrumental circuits in band-limiting and signal reconstruction applications that occur in typical mixed-signal ASICs [2].

Therefore, the synthesis methodologies developed here are applied to architectures, or topologies, of analogue filters, which are suitable for integrated and high-frequency applications. The new methods developed in the course of this research have been implemented in a demonstrator synthesis tool for analogue filters, called FIST (Filter Synthesis Tool). FIST uses a simulation-based optimisation approach for the design of analogue filters, where the core of the optimisation engine is the widely-used HSPICE simulator.

1.1 Research motivation and aims

The primary aim of this research is to investigate and develop techniques for VHDL-AMS-based synthesis of high-frequency analogue filters suitable for use in mixed-signal applications. To achieve this aim, the research is divided into two main parts. The first part is to carry out an investigation to establish the feasibility of using VHDL-AMS for the purpose of mapping behavioural descriptions into hardware. The second part of the work will focus on the development of behavioural synthesis techniques, their implementation in an integrated synthesis environment and validation using several case studies of lowpass and bandpass 1GHz filters. Emphasis will be put on methods to support VHDL-AMS-based behavioural synthesis with efficient architectural and parametric optimisation of filter topologies. The focus of the optimisation techniques is to ensure a high correlation between the desired and synthesised transfer functions as well as power consumption. The possibility of using full HSPICE simulations in the evaluation of filter topologies will be investigated to ensure the accuracy of the design process.

To overcome the difficulty in the modelling of high-frequency effects, fabrication foundry transistor models especially developed for the underlying CMOS 0.35μm technology will be used in the simulations. The models are based on BSIM3v3 MOSFET model from Berkeley that were developed for submicron devices, and contain about 120 parameters [3]. As noted in the literature [4], the reason behind the slow acceptance and adoption of analogue circuit synthesis techniques by the design industry - although the research has begun in about the early 1980s - is because of the issue of accuracy and reliability. It is therefore suggested [4] to use full simulation in evaluating the design, to make the synthesis process more reliable.

It is expected that the new synthesis techniques will be compatible and suitable for implementation in future mixed-signal environments for ASIC design. This is because VHDL-AMS offers standardised ways in which analogue filters may be behaviourally described in both

the time- and frequency-domains. Behavioural descriptions are important in the abstraction at higher levels of hierarchy where designers are not yet concerned with any structural details. Integrated high frequency filtering applications are the most likely area where the role of automated synthesis in a mixed-signal environment would be indispensable.

1.2 Dissertation contents and organisation

This thesis is organised into 7 chapters. Chapter 1 presents the overview of the research topic including the motivation and aims. Chapter 2 overviews the most prominent analogue synthesis methods and tools developed over the last two decades. It also summarises some of the existing methods for analogue filter synthesis.

The first part of the research, as specified in the previous section, is presented in Chapter 3. In particular, the process of behavioural synthesis from VHDL-AMS parse trees is explained. VHDL-AMS modelling techniques for analogue filters which are suitable for synthesis are proposed and demonstrated with case studies. Secondly, the algorithms of the tools that have been developed to support VHDL-AMS behavioural modelling are presented. Chapter 3 shows how filter information is extracted from the VHDL-AMS description.

The second part of the research is explained in Chapters 4 and 5. The synthesis procedure, which begins from the extracted VHDL-AMS filter information until the production of a filter netlist is outlined in Chapter 4. Inclusive to this procedure is the architectural and parametric optimisation methodology. The basis for high-frequency synthesis is the use of full HSPICE simulation within a three-tier optimisation loop. The parametric optimisation of each filter candidate is performed using a three-tier algorithm, which is a new implementation of existing tools and algorithms. This three-tier algorithm is detailed in Chapter 5. At the end of this chapter, the full implementation of the developed methods into FIST, a complete automated filter synthesis system is presented.

Chapter 6 demonstrates the feasibility of the presented techniques and the practical operation of the system using another two case studies. In the case studies, several candidate topologies suitable for integrated high-frequency applications are investigated and the best candidate is automatically selected.

Finally, the conclusions as well as suggestions for future research directions are presented in Chapter 7. The research contributions of Chapters 3, 4, 5 and 6 have been published or accepted for publication [5-11].

2 State of the Art

Before moving on to the related works, a typical synthesis flow will be briefly presented so that the following sections are put into perspective. Figure 2.1 shows the sequence of stages that starts from the concept of a desired functionality down to its physical realisation. This scenario presents a global view at a higher level than the circuit level, that is, at the system level, for the synthesis of analogue or mixed-signal IC.

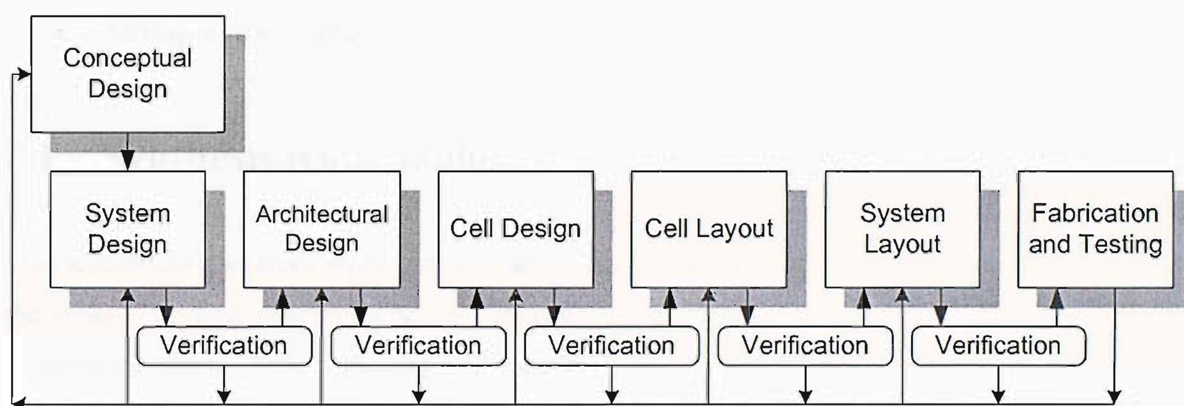


Figure 2.1 The analogue or mixed-signal IC design process.

Referring again to Figure 2.1, the following information is extracted from [12]. The first stage is the *Conceptual Design*, where information regarding the product and its specifications are decided. The *System Design* stage is where the overall architecture of the system is designed. The hardware and software parts as well as the interfaces are defined. Decisions regarding implementation issues such as testing and target technology are made. Then during the *Architectural Design* stage, the hardware part is decomposed into functional blocks and each block is specified in a suitable hardware description language, for example VHDL and VHDL-AMS. This stage is where 'high-level synthesis' begins; any description of the desired functionality is specified with no implication to the physical structure of the circuit. Then, in the *Cell Design* stage, each analogue block will be designed, i.e. a proper circuit topology will be selected and the parameters of the circuits will be sized. Next, in the *Cell Layout* stage, the schematic or netlist of the topology will be translated into geometrical representation. Finally, in the *System Layout* stage, the functional blocks of the whole

system is placed and routed, and if it passes the verification stage, the IC is fabricated and tested as indicated in the *Fabrication and Testing* process box.

It must be mentioned that progress between each stage will only be made if the current stage satisfies the verification process. If it fails, or if a potential problem is detected, the design process backtracks to redesign. The verification processes are done by appropriate simulation tools.

In the light of this research, the focus is directed to the Cell Design stage, where the starting point is an analogue filter's description in VHDL-AMS, and the final product is an HSPICE netlist of the filter circuit. The correctness of the filter's functionality is verified by HSPICE simulation. This will be described more in the subsequent chapters. Following the above synthesis or design flow, this chapter surveys reported work in the main areas that concerns the current research work. The literature review is organised into three parts:

- Synthesis from analogue HDL.
- Analogue circuit synthesis.
- Analogue filter synthesis.

2.1 Synthesis from analogue HDL

This section reviews three works on analogue or mixed-signal synthesis that uses analogue HDL as the input. The first two is KANDIS [13] and VASE [14], which is targeted for mixed-signal, whereas the third is NEUSYS [15], that focuses on the synthesis of neural networks. Both VASE and NEUSYS takes VHDL-AMS as its input, while KANDIS uses VHDL-hybrid. A recent paper on behavioural modelling [16] proposes a behavioural model based on a synthesisable subset of VHDL-AMS, which is developed part of VASE.

2.1.1 KANDIS (1995)

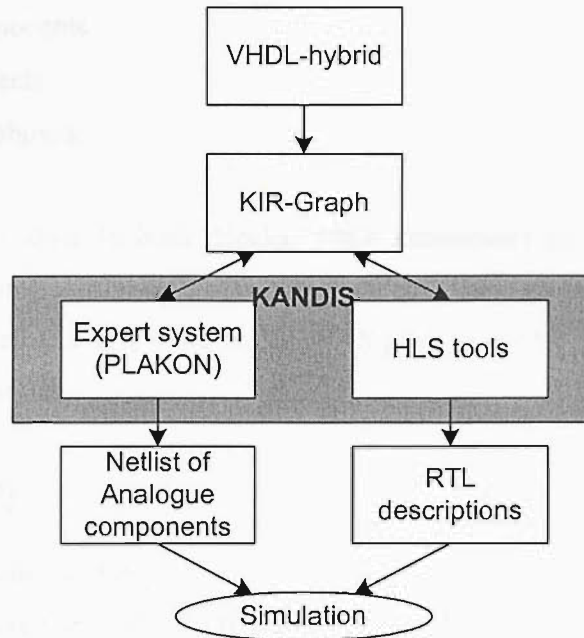


Figure 2.2 KANDIS design flow.

KANDIS [13] is a tool for the construction of mixed analogue/digital systems; using a knowledge-based expert system. Also part of KANDIS is a high-level synthesis (HLS) tool. As shown in Figure 2.2, the input language of KANDIS is VHDL-hybrid. VHDL-hybrid is for the specification of hybrid systems using the syntax of VHDL and is expanded for synthesis. For example, specification of non-conservative analogue systems is done by having features for integration and differentiation of a signal over time.

The front end of KANDIS translates VHDL-hybrid into an intermediate representation – KANDIS intermediate representation (KIR) - which is a graph of edges and nodes. Each node can contain a subgraph. The semantic of a node determines the tools that handles its subgraph, either KANDIS, or the HLS tool. For example, subgraphs interpreted as netlists are passed either to the expert system where they are constructed into functional blocks, or they are transformed into discrete-time graphs. The discrete-time graphs are then passed to the high-level synthesis and estimation tool.

The expert system PLAKON interacts with the high-level synthesis tool in producing the attributes for the algorithmic block and by automatically generating KIR-graph for functional blocks in the digital domain. HLS generates the value of the attributes which are used by the expert system.

Functional blocks are constructed using PLAKON. The construction process starts with the generation of an instance of each block. Constraint nets are produced along with instances. There are four possible construction steps:

- Instantiating sub components by splitting objects
- Integrating components
- Specialising objects
- Parameterising objects

Top-down refinement is done to basic blocks, while estimations for area, power, delays and inaccuracies are propagated bottom-up by the constraint net. These methods produce estimation of design parameters. In a more recent work on KANDIS [17], a new front-end that accepts VHDL-AMS as the input is presented.

2.1.2 VASE (1997)

VASE (VHDL-AMS Synthesis Environment) [18] [14] is used for synthesis of analogue systems from VHDL-AMS behavioural descriptions. The top-down hierarchical modelling technique to automate the synthesis process of analogue CMOS integrated circuits employs VHDL-AMS as the highest levels of abstraction [19-21]. The result is a sized netlist of electronic circuit, which satisfies the performance constraints and minimises the ASIC area.

The input of VASE is VHDL-AMS hierarchical intermediate format (VHIF) representations which are obtained from compiling the VHDL-AMS specification of a system. The VHDL-AMS used here is appended to make it suitable for synthesis [22]. In VHIF, continuous-time behaviour is represented as signal-flow graphs.

The synthesis methodology, shown in Figure 2.3, uses a two-layered design space exploration [23], where there are two phases: exploration and estimation. In the exploration phase, there are two interacting steps, i.e. architecture generation and constraint transformation and component synthesis. The constraint transformation and component synthesis interact with the analogue performance estimator (APE) [24] in the estimation phase.

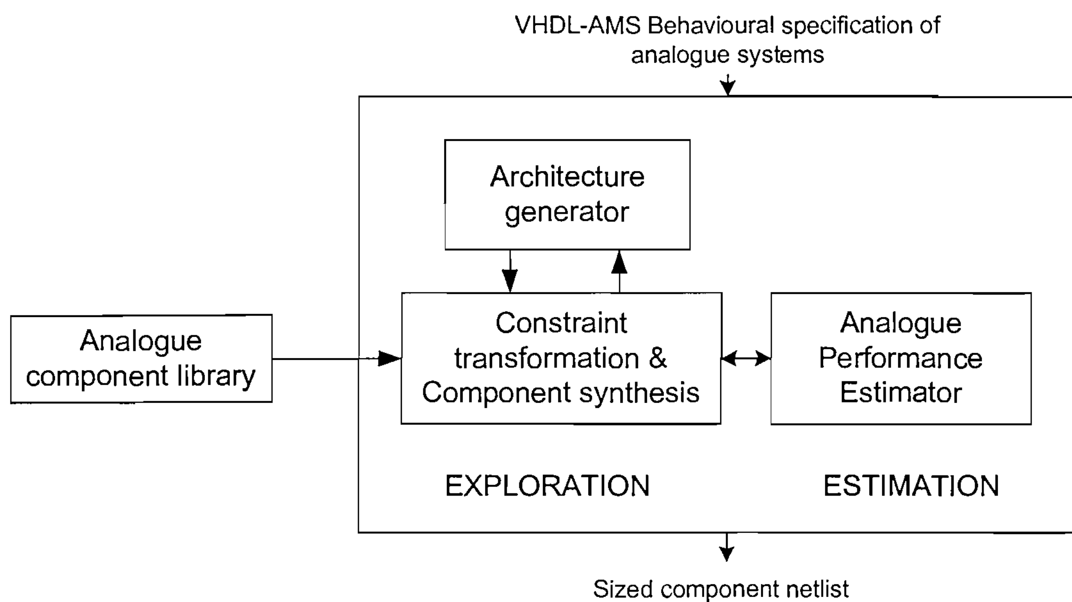


Figure 2.3 VASE analogue behavioural synthesis methodology.

From the behavioural description of an analogue system, an architecture generator [25] generates alternate system topologies. The architecture generator gets the topologies from a library. This analogue components library contains unsized topologies of commonly used analogue circuits. The constraint transformation and component synthesis step then considers each system topology and obtain the subcomponents and their design/performance constraints. A branch and bound algorithm is used for the architecture generation. The constraint transformation and component synthesis uses genetic algorithm (GA) -based heuristic methods.

APE accepts design parameters and the corresponding topology of an analogue circuit and determines the performance parameters and the sizes of circuit elements. The APE provides fast and accurate estimates on systems performance at various abstraction levels.

Design methodologies in VASE has been demonstrated in the development of an environment for implementing mixed signal designs from specifications in VHDL-AMS on rapid prototyping hardware [26, 27], where a decoder is synthesised onto a field programmable gate array (FPGA) and field programmable analogue array (FPAA). Also part of VASE is the methodology for formal verification of the synthesised analogue design [28].

2.1.3 NEUSYS (2000)

NEUSYS is an automated synthesis system for neural networks [15, 29, 30], based on netlist extraction from VHDL-AMS parse trees [31], developed jointly in University of Southampton and Universidad Politecnica de Cartagena in Spain. The technique has been applied to several synthesis examples of dynamic systems with feedback, one of which was Lorenz's chaos oscillator [31].

The architectural synthesis technique in NEUSYS uses VHDL-AMS parse trees that are translated into intermediate SPICE netlists in terms of primitive cells of opamp-level or transistor-level analogue modules such as analogue multipliers and adders. The final optimised structure is obtained by combining and parameterising analogue modules. A solution with the minimum cell count but exhibiting equivalent functionality is selected as the optimal one.

The synthesis technique employed in NEUSYS is based on recursive translation of hierarchical structural descriptions (in terms of VHDL-AMS component instantiations) into HSPICE subcircuits. Other VHDL-AMS formats such as simultaneous statements and break statements are also directly translated and mapped into electronic structures.

2.2 Analogue synthesis

Since most of the existing tools since the past two decades are targeted for a specific analogue module at the circuit level, a typical synthesis flow common to all the cell-level analogue CAD tools reviewed here is shown in Figure 2.4.

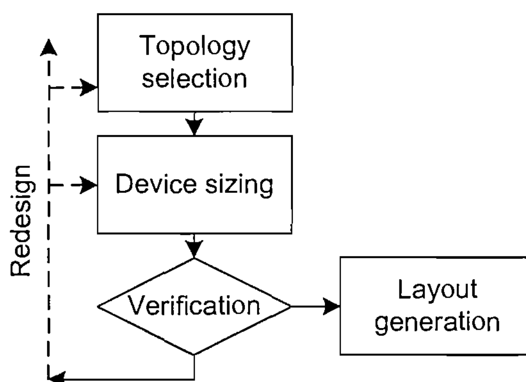


Figure 2.4 Cell-level analogue synthesis.

From Figure 2.4, for the synthesis of analogue circuits, there are several basic tasks common in the design and building of an analogue module, which are, topology generation, device sizing and layout generation [32]. Further review and summary in this chapter will mainly cover the topology generation and device sizing aspects only.

The most common classification of the types of analogue synthesis tools usually refers to how device sizing (and sometime topology generation, or both) is performed. Thus, the existing method for analogue circuit synthesis during the past two decades are knowledge-based and optimisation-based. Knowledge-based tools are among the earlier generation of analogue synthesis tools, and are characterised by having designers' knowledge being codified in the synthesis

algorithm. On the other hand, the architecture of optimisations-based tools typically consists of two main parts, the first being the optimisation engine, and the other is the performance evaluation part [33]. From the evaluation method, Krasnicki etal [33] further classifies the existing methodologies into four categories, which are 1) equation based, 2) symbolic analysis, 3) simulation-lite, and 4) full-SPICE. Further discussion is offered in the coming subsections 2.2.1 and 2.2.2 regarding these two broad categories of knowledge-based and optimisation-based tools.

Another distinct method of the previous approaches is the way the synthesis tool view the synthesis task, either by breaking down the task hierarchically, or by solving the problem in a single step. The first strategy is similar to that in digital is to decompose the synthesis tasks into sub-tasks, that is, going down the hierarchy where the topmost level is the input specifications. Examples are An_Com [34], CAMP [35], OASYS [36], BLADES [37], CHIPAIDE [38], STAIC [39], VASE [14] and AMGIE [47]. In contrary, flattened design view schemes such as that being used by IDAC [40, 41] selects a circuit from a library and optimise the circuit parameters according to the specifications. This approach may be simpler than the hierarchical one although it may need an extensive topological library together with a powerful and efficient optimisation strategy.

Most of the existing tools are for specific types of analogue modules, for example operational amplifiers (OPASYN [42], OAC [43], GPCAD [44]). There are also comprehensive design automation environments for analogue circuits such as ADAM [45], ACACIA [46], and AMGIE [47] and also for mixed analogue-digital systems such as VASE [14].

Based on the discussion above, Table 2.1 summarises several analogue synthesis tools. There are several distinctive features that are used to classify the types of each tool; KB (Knowledge-based) or OB (Optimisation-based), SB (Simulation-based) – where SB is a subset of OB; and A (analogue only) or MS (Mixed-signal). Cell-level tools are marked as C, whereas a design environment is marked as E. Also, H refers to hierarchical, where the synthesis problem is decomposed into several smaller tasks and is solved by refinement. F refers to flattened design view [48] where the synthesis task involves selecting a topology and its subsequent optimisation. The classification is by no means authoritative, but is made based on the most prominent feature of the tool. As a matter of fact, a tool may be a mixture of all or any of KB, OB, or SB.

Name, Year		Origin	Type	Scope	Structure	Notes
1	IDAC 1987	Swiss Center for Electronics and Microtechniques, Switzerland	A	C	F KB	Part of ADAM [45]. Users select a topology from a library.
2	OASYS 1988	Carnegie Mellon Univ. (CMU), USA	A	C	H KB	Part of ACACIA [46]. Top-down hierarchical structure in knowledge application [49]. The layout tool used is ANAGRAM [50].
3	OPASYN 1988	Univ. of California, Berkeley, USA.	A	C	F OB	Silicon compilation of opamps

4	An-Com 1988	General Electric Company, New York, USA	A	C	H KB	Domain knowledge is used for successive decomposition of circuit specification.
5	CAMP 1988	Univ. of Southern California, USA	A	C	H KB	Uses iterative self-reconstructing technique and circuit simulation for a flexible architecture.
6	DELIGHT. SPICE 1988	Harris Corporation, USA	A	C	F OB	Utilises a SPICE simulator as the optimisation core.
7	BLADES 1989	AT&T Bell Labs., USA	A	C	H KB	Uses artificial intelligence to combine formal and intuitive knowledge.
8	ASAIC 1990	Katholieke Universiteit Leuven, (KUL) Belgium	A	E	F OB	Features a symbolic analysis programme, ISAAC [51] and an optimiser, OPTIMAN [52-54].
9	CHIPAIDE 1990	Imperial College, UK	A	C	H KB	Uses a hierarchical approach to produce first-cut circuit topology.
10	OAC 1990	Kyoto Univ., Japan	A	C	F OB	CMOS opamp compiler which runs a simulation-based optimiser as a post-processor.
11	STAIC 1992	Univ. of Waterloo, Ont., Canada	A	C	H OB	Uses description language in its multilevel modelling scheme. Synthesis uses successive solution refinement technique.
12	MINLP-Maulik 1992	CMU, USA	A	C	F OB	Allows simultaneous circuit topology and parameter selection.
13	ARCHGEN 1995	Vanderbilt Univ., USA	MS	C	F OB	Synthesis of filter systems from behavioural specifications.
14	KANDIS 1995	Johann Wolfgang Goethe Univ., Frankfurt	MS	C	F KB	Translates hybrid-VHDL into intermediate representation (KIR graph), which are then used by a high-level synthesis tool and an estimator.
15	FPAD 1995	Ecole Polytechnique of Univ. of Montreal, Canada	A	C	F OB	Use of fuzzy logic in optimisation. Multiple optimisation objective using analytic models.
16	FASY 1996	Univ. of Seville, Spain.	A	C	F OB	Use fuzzy logic to select a topology from a library, done by an expert designer. Selected topology then being optimised by a two-phase optimiser. Tested on CMOS opamps.
17	ASTRX/ OBLX 1996	CMU, USA	A	C	F OB	Uses asymptotic waveform evaluation (AWE) to evaluate circuit performance and simulated annealing for optimisation.
18	VASE 1997	Univ. of Cincinnati, USA	MS	E	H OB	A modified (behavioural) VHDL-AMS specifications are compiled to obtain a hierarchical intermediate representation.
19	GPCAD 1998	Stanford Univ., USA	A	C	F OB	Use of geometric programming to optimise and automate component and transistor sizing for CMOS opamps.

20	MAELSTROM 1999	CMU, USA	A	C	F SB	Synthesis at cell-level, where a simulator is included (encapsulated) in the synthesis process. Uses a combined annealing/genetic optimisation algorithm, and exploit network parallelism in distributing computing tasks.
21	ANACONDA 1999	CMU, USA	A	C	F SB	A fast and robust numerical search strategy based on pattern search. Simulation-based tool where validation is done using an encapsulated industrial simulator for every search. Exploits network parallelism
22	AMGIE 1999	KUL, Belgium	A	E	H OB	This is an analogue IC synthesis environment that utilises tools which covers the complete design flow from specification down to layout generation, with redesign features in case of failure
23	NEUSYS 2000	Spain & Univ. of Southampton, UK	A	C	F KB	An architecture generator translates the parse trees obtained from behavioural VHDL-AMS specifications into intermediate netlists for subsequent optimisation

Table 2.1 Summary of analogue synthesis tools (up to year 2000) described in this chapter.

2.2.1 Knowledge-based tools

This type of tools relies heavily on encoded expert designer's knowledge, where the major concern is usually in obtaining and saving the designer's knowledge in the database. Examples are IDAC [40], An_Com [34], CAMP [35], OASYS [36, 55], BLADES [37], and CHIPAIDE [38]. These will be summarised in the following subsections.

2.2.1.1 IDAC (1987)

IDAC [40] is part of a larger synthesis environment called ADAM [45]. Also included in ADAM is ILAC [56], the layout tool. IDAC (Interactive Design for Analog Circuits) is among the earliest knowledge-based analogue circuit synthesis tool. As an input to the system, user has to specify the building block parameters and choose a topology from the library. The library contains schematics of verified analogue knowledge and formal description of circuit, such as operational amplifiers, comparators and oscillators. Input specifications are translated by formal description to get results such as devices' sizes and bias conditions. A built-in verification system is used to ensure the correctness of the design. If not, the analyser will redefine the input specification and execute another formal description.

The main feature of an improved version of IDAC [41] is that it is open to user's expertise. Designers can store their expertise in the extendable knowledge-base system, such that it can be reused. The built-in knowledge of IDAC consists of sizing algorithms that calculates the sizes of devices. The knowledge about circuit's topology and functionality is placed in a 'Describe' file, while knowledge about sizing sequence is stored in a design plan. A SPICE file can be translated into a 'Describe' file by the symbolic analysis program, BRAINS. The 'Describe' file is divided into 6 sections, where sections 1-3 contain topology knowledge while sections 4-6 contains knowledge of circuit functionality. The design plan is formally implemented in IDAC in the form of sequences of sizing steps in a source code. The source code uses design primitives, which are model manipulators, device calculators and structure calculators. Verification is done using Monte-Carlo analysis.

2.2.1.2 CAMP (1988)

CAMP [35] is a knowledge-based system, where the expert system is used in the iterative analogue design process. Architecture design uses self-reconstructing technique. It is used in the design of a two-stage CMOS operational amplifier.

The expert system first produces an initial design, which is then simulated by a circuit simulator to evaluate the performance. The results are fed back to the expert system, which will

optimise the design by recommending parameters for the next design iteration, where current design is modified.

The initial design is chosen by the expert system from several architectures in the knowledge base. From analytical equations, transistor dimensions are determined. If necessary, the next design cycle will include modifications of the device sizes or/and circuit topology. Circuit primitives are used as 'replacement parts' in circuit reconstruction. The self-reconstructing technique is done by the expert system by substituting the corresponding design equations in the knowledge base. The integration between the expert system and circuit simulation ensures that an accurate design is attained.

The design automation system also includes an automatic layout generator, LAMP. LAMP performs two tasks – circuit primitive recognition and circuit layout generation. The input to the recognition module is a SPICE-like netlist and the output is a set of recognised circuit primitives. From this, the layout generation module generates the final circuit layout based on the analogue layout knowledge contained in the knowledge base.

2.2.1.3 OASYS (1988)

OASYS [36, 49, 55, 57] is another knowledge-based tool that applies the knowledge of analogue circuit design in a hierarchical manner. In this way, complex design task can be tackled in several refinements; from coarse to grain, as opposed to previous methods of designing analogue circuits, which is usually done in a flat manner.

The human knowledge is codified as a planning system, having default design plans for each design style template. The planning system involves several steps, that each involves numerical optimisation before moving down to the next more-refined plan.

At each hierarchical level, a fixed design style is chosen and the performance specification for the subsequent level is translated so that in the end, each device is sized. Optimisation is done - in the form of plan-fixers, as the synthesis process moves down the hierarchy whenever a plan fails to reach its particular goal. OASYS demonstrates the effectiveness of a hierarchical approach in synthesis of analogue circuit elements, as the prototype demonstrates with CMOS operational amplifiers.

Another framework, OASYS-VM (OASYS virtual machine) [55], has been developed as a structure that allows an easier interaction between the designer and the system. There is also a prototype layout tool, ANAGRAM [50] that works together in a larger synthesis framework, ACACIA [46].

2.2.1.4 An_Com (1988)

The An_Com [34] compiler is specifically operational amplifier; where the inputs are performance specifications and technology and process parameters. The compilation result is a SPICE subcircuit and a complete layout.

Domain knowledge is applied as a top-down planning mechanism with a sequence of algorithms, rules and other data stored in a template. The collection of templates is used to dynamically construct the solution space, which is a tree with weighted ordered nodes. A template is made of a behavioural and a structural component.

A sequence of successive decomposition approach is used in the silicon compilation process. The circuit topologies are specified hierarchically. High-level specifications are decomposed into specifications of elements at a lower level of the hierarchy, until reaches the level where elements are leaf cells or have layout generators. The top-down decomposition of behavioural and structural specifications is driven by template knowledge. It proceeds by querying the existence of a generator or predesigned block (typically, non-existence at high levels). An optimisation step may be used to bring the solution closer to the specifications. After the interconnections are specified by the templates, the decomposition is verified by simulation, using macromodels. At the lowest level of hierarchy – the device level - simulation is done using SPICE. Failure recovery and trade-off analysis is done if specifications are not met at the end of decomposition. Failure recovery modifies the instantiated template or current specifications, and if the modifications are not acceptable, failure recovery proceeds to the upper level in the hierarchy to search for other topology and decomposition. After a complete decomposition, final tests, which are parameter extraction, layout verification and a full simulation, are done.

2.2.1.5 BLADES (1989)

BLADES [37] is a knowledge-based tool which uses an expert system. It applies artificial intelligence, which combines the application of formal and intuitive design knowledge. This design environment uses different abstraction levels based on the complexity of the design task. BLADES use three types of analogue hardware description languages at distinct design levels to describe the functionality of circuit components at each level. Each language is then translated from the highest level to the lowest.

An expert system mainly consists of a knowledge base and an inference engine. The knowledge base contains both the formal and intuitive knowledge, and the inference engine is the problem-solving tool based on production rules. The expert system of BLADES implements the rule-base using 'if-then' format. The inference engine matches the data with the 'if' part of the production rules. Once the rule is applied, its 'then' part is executed. Data is added or removed from working memory until no-match terminates the execution.

BLADES system architecture consists of 5 large subsystems: an expert system manager, a subcircuit design expert, a subcircuit knowledge base, a test generation facility, and a circuit design consultant. Subcircuit design experts provides the design primitives. Each expert has its own knowledge base. Each subcircuit function is represented by its topology and a set of design equations to calculate subcircuit parameters. The equations are taken from textbooks, but are less accurate than SPICE models. Design experts are triggered when the expert system fires the rules chosen for a design problem. Design consultants are programs using algorithmic methods for analysis and designing. After BLADES completes a design test, the test generator is invoked and then generates test files along with data for simulation.

2.2.1.6 CHIPAIDE (1990)

CHIPAIDE [38] uses a knowledge-based circuit generation module. It has three other modules for optimisation, layout generation and tolerance design. Its strategy is to produce a first-cut circuit based on descending the hierarchical level, from selecting architecture from the knowledge base until the transistor level. Knowledge is also used in the simple rule-based correcting procedure.

The circuit generator attempts to produce a first cut circuit topology based on the input from user giving performance specifications and fabrication process' constraints. The strategy here is to use a hierarchical approach, starting from the architecture level. An architecture will be selected from the knowledge base, then in the task level, the building blocks are decomposed into task blocks. Task blocks are selected from a knowledge base, which for a particular task, there may exist alternative blocks to offer. Decomposition continues until it reaches the transistor level, which is the final level. The correction procedure is based upon expert knowledge, which points to the parts in the architecture responsible for the discrepancy. The adjustment made to correct the discrepancy is then translated down the hierarchy.

The closed-loop nature of CHIPAIDE lies in its ability to compare simulation results with performance specifications and employ a correction procedure if the comparison fails. The design loop is closed by extracting the associated parasitics of the layout of the transistor level circuit and feeding it back to either the circuit generator or the tolerance design module to minimise or eliminate the parasitic effects. CHIPAIDE is evaluated using operational amplifiers.

2.2.2 Optimisation-based tools

This section highlights and briefly describes the optimisation strategies used in synthesis tools. As stated in [33], the optimisation-based tools can be divided into 4 groups according to the method of performance evaluation. The four classifications made by the author are:

1. *Equation-based:* This constitutes the majority of early approaches as it offers a fast method to evaluate the quality of the proposed candidate over the entire solution space. However, as it is impossible to fully model the behaviour of a circuit topology as a set of closed-form analytical equations without the need of simplification, the accuracy is limited. Also, to create the model and its set of equations itself is a complex and time-consuming task. Krasnicki relates that a number of optimisation strategies are used in conjunction with equation-based techniques:

- Numerical search - OPASYN [42, 58], IDAC [40]
- Combinatorial search - Maulik [59]
- Hierarchical systems that attempt to decompose the evaluation and optimisation - OASYS [36], STAIC [39]
- Qualitative and fuzzy reasoning techniques - FASY [60], FPAD [61]
- Geometric programming - GPCAD [44]

2. *Symbolic analysis:* OPTIMAN [53],

3. *Simulation-lite:* Combination of simulation and equation-based modelling: ASTRX/OBLX [62].

4. *Full SPICE simulation:* DELIGHT.SPICE [63], MAELSTORM [64], ANACONDA [4, 65]

To handle the complex design space of analogue circuits, several preferred algorithms that have the potential to attain a global solution are applied in the optimisation-based tools. Firstly is simulated annealing [66], that has been used, for example, in OPTIMAN [53] and ASTRX/OBLX [62]. Simulated annealing is also used in a over-designed cell method [67] that combines both topology selection and parametric optimisation by eliminating or introducing components in a fixed topology. Another algorithm is the genetic algorithm, as used in VASE [25]. Also, the combination of both genetic and annealing algorithms have been used to increase computational efficiency [64], [68].

2.2.2.1 OPASYN (1988)

OPASYN [42, 58] is a synthesis tool specifically for operational amplifiers (opamp). It uses an optimisation algorithm which uses algebraic evaluation design equations taken from domain specific design knowledge.

User inputs the opamp performance specifications, which will be taken by the tool to select the most suitable topology that it has in its internal database. This database contains design knowledge necessary to make the correct selection of opamp circuit. Selection is done by pruning the decision tree, which leads to leaf nodes of proven topologies. The selected topology is then parametrically optimised to tailor its specifications as being required. This is done by using the analytic model of each of the topology in the database. First order circuit analysis is used to obtain analytic design equations, which are then used together with user-defined design targets to form a

cost function. The steepest descent algorithm is used to explore the search space to obtain an optimal solution. Evaluation is done by comparing the predictions of OPASYN with SPICE simulation results. The final step in this framework is layout generation, where the output is a design-rule-correct mask geometry.

2.2.2.2 DELIGHT.SPICE (1988)

DELIGHT.SPICE [63] is the combination of an interactive optimisation-based CAD system (DELIGHT) and the SPICE circuit analysis program. The optimisation design problem is formulated as a standard mathematical programming problem. A set of objectives are optimised subject to constraints. Constraints are classified as either hard or soft. The optimisation algorithm consists of three phases:

- Attempts to satisfy all hard constraints by minimising the hard constraints violation.
- While maintaining the hard constraints, worst normalised values of objectives and soft constraints are improved progressively.
- While maintaining the hard and soft constraints, the worst normalised objective value is progressively improved.

The output of the program includes a performance comb, which displays how close the objective and constraint values are from their corresponding good and bad values. The SPICE program does the performance and specification evaluation for the optimisation. The constraints and objectives used in DELIGHT depend on the DC, AC and transient analyses of SPICE.

2.2.2.3 ASAIC (1990)

ASAIC (Automatic Synthesis of Analog Integrated Circuits) [48, 52] is a system that combines symbolic simulation, numerical optimisation and knowledge-based techniques. There are four main modules used in this design environment. First is the symbolic simulator, ISAAC [51], which returns analytic models, which are then used by DONALD [54]. DONALD is the equation manipulator that converts the model into a solution plan that would be optimised by OPTIMAN [52, 53, 54], the optimisation program that performs device sizing. Finally, the circuit is laid out by the AUTOLAC [52] program.

ISAAC is a program that does symbolic analysis on analogue integrated circuits to derive AC characteristics in symbolic expression of circuit parameters. Its feature is that the expressions can be simplified showing only dominant terms only, and mismatch terms can be explicitly represented so that second-order effect can be calculated. The program reads in the circuit topology, expands the circuit into their primitive elements and then performs a topology check. Next, the analysis type is chosen where the user may interactively assign the excitation source,

inputs and outputs. After the transformations, the linear circuit equations are set up, and the expressions are simplified using heuristic approximation. The analytic circuit model produced by ISAAC is used by OPTIMAN. The optimisation algorithm is simulated annealing.

2.2.2.4 OAC (1990)

OAC [43] is a CMOS operational amplifier compiler that generates an optimised layout. OAC needs performance specifications, process parameters and constraints as input, and it produces a complete layout of an optimised circuit, a SPICE circuit description and a performance summary. The architecture of OAC consists of:

1. Global design module.
2. Detailed design module: a circuit performance optimiser, and layout design and parameter extraction module.
3. Design library, which stores circuit topologies with their design knowledge and layout descriptions.

The design process starts with a global phase and then proceeds to the detailed design phase. The global phase selects a circuit topology from a library based on given specification, and process the rough circuit sizing by assigning approximate value to each design parameters. The next phase does simultaneous fine device sizing and layout design. This detailed design stage needs an accurate performance evaluation, which is done using SPICE. The optimiser utilises a nonlinear optimisation technique, which is based on specified performance constraints and relies on the simulation results of the circuit extracted from the layout.

2.2.2.5 STAIC (1992)

STAIC [39] is an interactive design tool that synthesises CMOS and BiCMOS analogue integrated circuits. It features the use of a description language for entering hierarchical circuit descriptions and a solver unit that dynamically integrates analytical model equations across the levels of hierarchy. The synthesis method is the successive solution refinement.

The analogue hardware description language used in STAIC is classified by the author as being categoric, specific, device or layout. This classification is based on the circuit complexity level. The entry of a description begins with filling the empty template or form with port and performance attribute declarations. STAIC also uses a multilevel modelling scheme, where in the simplest model level, circuits are described as a collection of linear constraints that can indicate solution regions. Continuous nonlinear equations and derivatives are used to describe first-order and second-order model relationship. Modelling is based on analytical descriptions.

The initial design stages produces a solution using simplified models that capture design tradeoffs. An advanced model then would permit a more realistic account of circuit behaviour. The simplified model produces the starting point for design exploration using the advanced models in the search of a global optimal solution.

Design space exploration and evaluation are done by the scanner and optimiser application modules. The scanner finds an initial solution by performing a coarse grid point search for a solution that is in the neighbourhood of a global optimal point. It can also be used to explore design tradeoffs. The optimiser may adjust the set of designated independent variables to facilitate the minimisation of objective value while ensuring that the final solution lies within the domain space.

2.2.2.6 MINLP - Maulik (1992)

MINLP (Mixed-Integer Nonlinear Programming) [59] is an approach to cell-level analogue circuit synthesis, where simultaneous topology selection and parameter selection is allowed. The problem formulation uses integer variables to model topology choices and continuous variables for the design parameters.

The design of a circuit is modelled as a constrained optimisation problem by using the performance equations derived from circuit topologies. Optimisation is done as a function of power or area, and is called the objective function. The MINLP problem formulation is solved using the branch-and-bound algorithm. This algorithm is suitable for simple and small-sized circuits. Other algorithms that are proposed for solving MINLP are Generalized Bender's Decomposition and Outer Approximation. This methodology where simultaneous topology and parameter selections are allowed greatly reduces design time over exhaustive search.

2.2.2.7 ARCHGEN (1995)

ARCHGEN [69] is a program that allows the synthesis of analogue filter systems. This is one of the earliest attempts to synthesis analogue systems to mimic the synthesis of behavioural digital systems.

A state space exploration is utilised to unify the synthesis process for the vast options of filter designs and implementations. For this, an intermediate architecture stage is introduced for architecture synthesis and evaluation. The input to the synthesis system is the behavioural specifications in terms of mathematical expressions in time and frequency domain. These are then synthesised into functional architectures, and are further synthesised into intermediate architectures, which are implementation specific.

Verification is done by the behavioural simulator ARCHSIM, which performs various analyses for the functional architectures and intermediate architectures. An architecture

specification language is used as the hardware description language for external interface between the synthesis and verification module.

2.2.2.8 FPAD (1995)

FPAD (Fuzzy nonlinear Program for Analog circuit Design) [61] is for the design automation of cell-level analogue circuits using fuzzy set theory to define the design objectives and constraints, and thus in the optimisation as well.

The first step is to get input specifications in terms of objectives or constraints. Constraints are classified into fuzzy (i.e. parameters with no precise values, or with tolerances) or crisp/strict (mathematical). The second step is to get an initial solution for the use of the optimisation module. This initial solution is obtained using an automatic device sizing procedure of circuit knowledge and basic assumptions, which are then evaluated using HSPICE. Alternatively, users may provide their own initial solution.

For the fuzzy objectives and constraints, there are membership functions (which may be linear, exponential or quadratic) with degrees of fulfilment for the fuzzy inequality: where '1' denotes that the objective has been reached, and '0' when it is missed. The problem is solved using a feasible direction algorithm, which however, finds a local solution. However, as fuzzy formulation is independent of optimisation algorithm, a more powerful algorithm can be used. Performance functions are evaluated using analytical models of circuit equations to avoid using costly simulations. HSPICE simulation is done only to evaluate the starting point of the optimisation and for the final evaluation of the designed circuit.

2.2.2.9 FASY (1996)

FASY [60] uses fuzzy set rule in the circuit topology selection process. A topology is selected from others based on the grading concluded by FASY after specification entry. The decision rules are obtained from an expert designer, or can also be automatically produced from previous experiences stored by the program. The selected topology is sized by a two-stage optimiser. A design that is accepted by the user will be stored in the database (i.e. decision surfaces as fuzzy rules), where the experience gained from this design can be used to modify the topology decision rules.

There are two optimisation phases. The first phase uses simulated annealing technique, with analytical models to estimate circuit performances. The result from this phase is used as the starting point in the next optimisation phase, in which SPICE simulation is used to compute the performance. The optimisation algorithm used in the second phase is a standard conjugate gradient algorithm.

2.2.2.10 ASTRX/OBLX (1996)

ASTRX/OBLX [62] uses five key ideas in the synthesis formulation:

1.*Synthesis via optimisation*: The circuit design problem is mapped to the constrained optimisation problem. This constrained optimisation problem is converted to an unconstrained optimisation problem so that simulated annealing can be used. The circuit performance is in the form of a cost function C(x).

2. *Asymptotic waveform evaluation (AWE):* AWE is used to evaluate C(x) without the use of designer-supplied equations. AWE can be applied to linear or linearised circuit and gives accurate results without manual circuit analysis.

3. *Simulated annealing:* This is the optimisation engine to search for the best circuit design in the solution space defined by C(x). Global optimisation can be done in the face of many local minima, plus it is starting point independent and can optimise without derivatives.

4. *Encapsulated device evaluators:* This is a compiled database of industrial models that are used to model active devices. The models are used to linearise non-linear devices, which will then generate a small signal circuit that can be passed to AWE. Models are completely independent from the synthesis system and all aspects of the device's performance and representation are hidden.

5.*Relaxed-dc formulation:* As the models must be treated numerically to solve the DC operating point after each circuit perturbation, this means that a substantial CPU time would be spent on an intermediate synthesis step. Therefore, Kirchoff's laws are explicitly formulated and included in the constraints function. This would be implicitly solved during DC biasing.

2.2.2.11 GPCAD (1998)

In GPCAD [44], CMOS transistors are modelled to be compatible with geometric program (GP), and performance specifications and constraints need to be appropriately expressed. GP is a convex optimisation problem, which can efficiently find a global optimal solution. GP is very fast, and, if a solution exists, convergence is guaranteed. However, problems must be formulated as polynomials or monomials of the design variables. The feasibility of the method in choosing the best architecture for a set of specifications in opamp design implies that geometric programming may be used in combination of better and more accurate models in a local optimisation method. Other advantage of GP is that the computational effort in solving it grows linearly with the number of constraints. Also, sensitivity analysis is possible once a geometric program is solved. However, new models need to be developed for new technologies.

2.2.2.12 MAELSTROM (1999)

MAELSTROM [64] attempts to use an-industrially accepted and approved circuit simulator in its optimisation loop with the argument that in order to make an analogue automation tool acceptable and trustworthy, the validation of the selected circuit must be done by a well-known simulator. To overcome the computationally costly method, three strategies are proposed:

1. *Simulator encapsulation:* This is to separate circuit optimisation and the simulation engines. The commercial simulator is encapsulated, by the use of software 'wrapper' that makes the simulator as an object with a set of methods. By this, the simulator's data format and idiosyncrasies are hidden and insulated from the optimiser.
2. *Using a combined genetic/annealing optimisation algorithm.* The optimiser uses an annealing-type of algorithm which is global and stochastic, but is threatened by its slowness due to the vast number of solution candidates that needs to be visited. With the intention to fully simulate each candidate in the optimisation process, this problem is made worse. A solution to this is to distribute and synchronise the search task over several annealers – using genetic algorithm proves to be useful, which forms the basis for parallel recombinative simulated annealing (PRSA). A search task is distributed over a number of PRSA-node, each of which will run an annealing process, and each node will randomly communicate its result to the other nodes. A PRSA-node may do a recombination of results to generate a new solution.
3. *Exploiting network parallelism of workstations.* The architecture of the network consists of the coordination of three tasks: optimisation done by several PRSA-nodes, scheduling of the evaluation requests by an evaluation master node, and the evaluation process done by a number of evaluation slave nodes.

2.2.2.13 ANACONDA (1999)

ANACONDA [4, 65] is a cell-level synthesis tool that also supports the idea of incorporating a full circuit simulator in the optimisation loop as in MAELSTROM [64]. The novelty here lies in the global optimisation algorithm which applies a parallel stochastic pattern search to solve a complex cost function that defines the synthesis task. The algorithm, together with an encapsulated simulator and a network of workstations is able to perform simultaneously a parallel simulation and numcrical search.

Pattern search algorithm is a direct-search method, having a provable convergence. The algorithm performs a random perturbation to the coordinates of the solution vector. The goal is to find a perturbation that minimises cost, and only an improved solution is accepted. As the process proceeds, the perturbation bounds are decreased as it shrinks around the evolving solution, until there is no more improvement. Implemented as a population of partial circuit solutions, each

population receives a short, randomised pattern search, where finally the best solutions will converge. Thus, a diverse area of the cost surface can be covered, plus poor solution candidate can be eliminated earlier to avoid being trapped in a poor local minima.

Similar to the MAELSTORM framework, ANACONDA also employs a network architecture, which is controlled by an evaluation master that dynamically schedules simulation tasks among a pool of workstations.

2.2.2.14 AMGIE (2000)

The design strategy in AMGIE [47] is a performance-driven hierarchical design sequences, implemented as an integration of different software tools. The tools are for topology selection, sizing and optimisation, verification and layout generation, which are all connected to the design controller. AMGIE system needs two libraries, the first is a 'cell library' containing information about the analogue cells, and the other one is a 'technology library' containing technology specific information.

The topology selection process works by eliminating inappropriate topologies from the candidates in the library and by ranking the remaining ones. Elimination is done by implementing three filtering sequences, namely, boundary checking (to eliminate topologies with unsuitable performance parameters), interval analysis (analysing interdependencies between different performance so that only most important specifications are considered), and rule inferencing.

Generation of sized-models is done using several tools. AC equations are derived using ISAAC or SYMBA, which uses symbolic analysis technique. DC equations can be derived directly from the circuit, while other equations need to be provided by the designer. An equation manipulation tool, DONALD, uses the constraint satisfaction technique to turn the declarative models into a computational plan that indicates how dependent variables are calculated from the independent one. This plan is stored in AMGIE's cell library, to be used by OPTIMAN, the optimisation tool. OPTIMAN allows the user to choose from a number of global or local optimisation algorithms, where the initial solutions and optimisation variables are included in the cell library for every schematic. The following step is verification, where a generic verification script has been written for each circuit. After the verification stage, the LAYLA tool will generate a layout of the circuit, where the placement and routing process have constraints based on the sensitivities of the performances to layout parasitics. Another tool implemented in the AMGIE system is a redesign wizard that automatically starts in the event that a design fails to fulfill all specifications. It looks for problems in design by scanning the history, and presents to the user the procedures to redesign based on scenarios stored in a redesign database.

2.2.2.15 Recent developments (2003)

WATSON [70] is a tool which relies on design space boundary exploration that would give insight on the trade-offs between competing circuit performances. The Pareto hypersurface representing the optimal design boundary is calculated using a multiobjective genetic optimisation algorithm. Trade-off analysis is done using multivariate regression technique.

In another recent work [71], an independent architecture generation technique which is not based on traditional library and cell mapping strategy is introduced. The exploration technique is based on conversion rules for signal flow graphs (SFG). The algorithm varies the signal type between voltage and current on the SFG structures.

An optimisation-based approach for exploring the design space of analogue circuits is demonstrated with the design of an operational transconductance amplifier [72]. The performance metrics are obtained directly from circuit netlist to form symbolic equations that are used in an equation-based optimisation approach [73].

2.3 Analogue filter synthesis

Automated synthesis of filters has been used for many decades. This section discusses some recent analogue synthesis CAD tools for filters.

BECAS [74] is a synthesis tool for OTA-based linear analogue functions. All linear functions are composed from first- and second-order lowpass filter structure.

A PROLOG-based synthesis tool [75] uses OTA-C circuits as its main building block structure. The PROLOG program realises the symbolic expressions for the filter transfer function by replacing the building blocks with integrator, amplifier, open circuit or short circuit.

Another optimisation-based technique for filter synthesis [76] is by having a multi-dimensional search done on a space whose coordinates are the values of the elements of the filters. The filter is described as a network function containing the complex frequency variable of the poles and zeros and the vectors of the filter element values. A scalar error criterion is used as a measure of the element values satisfying the specified approximations. Searching begins from starting points which encompasses the values of each element over a specified range and incremental value. Different searching algorithm produce different configurations of starting points.

ARCHGEN [69] synthesises both continuous and discrete systems, and accepts the behavioural description in the form of transfer functions as its input. In a continuous system, the frequency domain Laplace transfer function is transformed into state-variable descriptions in the time domain. State-variable descriptions consist of differential equations that describe the internal and terminal behaviour of a system. Correspondingly in discrete systems, a z-transfer function in

the frequency domain is transformed into state-variable description of difference equations in the time domain. The model is used to generate a functional architecture, which is composed of continuous analogue functional blocks such as adders, integrators and amplifiers. The next step of synthesis in ARCHGEN is done by replacing each functional component at the architecture level with an implementation style specific realisation.

EXSHOF [77] is an expert system based CAD synthesis tool of high-order active RC filters. The expert systems asks questions to the user at various stages of the design process as it assumes that users have little knowledge of computers and filters. The role of the expert system is in analysing the filters for approximation functions, and trade off between the number of active components and specifications. The knowledge base of EXSHOF contains a library of biquad circuits. A similar, earlier work which was the basis for EXSHOF is CHOose FILter (CHOFIL) [78], a knowledge-based expert system for the synthesis of active biquad filter.

Other existing automated filter synthesis tools such as [79-81] target at specific types of continuous-time filter. In [79], switched-capacitor ladder filters are optimised using a commercial switched-capacitor analysis programme in the optimisation loop. SYSCUF [80] uses switched current integrator as the basic building block. A multi-level optimisation approach [81] is used for the synthesis of switched capacitor filter.

3 Behavioural Synthesis of Analogue Filters: from VHDL-AMS description to filter information

The key to behavioural synthesis is the specification method in which the desired circuit performance is expressed. In this research, the specification is done using VHDL-AMS. This chapter introduces and discusses the concept of behavioural synthesis from VHDL-AMS parse trees. The first part gives the background of the VHDL-AMS language, in particular its new features that can be exploited in analogue synthesis, specifically synthesis of filters.

Secondly, VHDL-AMS modelling of filter characteristics using time- and frequency-domain constructs are discussed. Then, VHDL-AMS parse trees are introduced and finally, the last part of this chapter focuses on the software modules developed in the course of this research to extract filter information from the behavioural VHDL-AMS constructs, namely the synthesis syntax checker and static calculator. The algorithms for the synthesis of behavioural VHDL-AMS models are illustrated by two case studies.

3.1 VHDL-AMS

VHDL-AMS stands for Very-High Speed Integrated Circuit Hardware Description Language for Analogue and Mixed-Signal. It is an IEEE standard hardware description language (HDL) for analogue and mixed-signal systems [1]. VHDL-AMS is a superset of VHDL [82] - a digital HDL already well established and widely used since the late 80s.

VHDL-AMS [1, 83] can be used for modelling and simulation of systems containing discrete-event (digital) and continuous-time (analogue) signals. Continuous-time models are implemented using differential and algebraic equations (DAEs), while event-driven behaviour is modelled by concurrent processes which are sensitive to signal changes.

Complex models can be built using combinations of differential equations, algebraic constraints and logical controls. Furthermore, VHDL-AMS can describe the behaviour of non-electrical dynamic systems like transitional and rotational mechanics, fluid flow, heat flow and

many others. VHDL-AMS has already been used to model semiconductor devices [84, 85] and micro-electromechanical systems (MEMS) [86], as well as neural networks [87].

Several proprietary HDLs have been proposed prior to the release of the VHDL-AMS standard to cater for the problem of modelling and verifying mixed-signal and mixed-domain systems, for example HDL-A by Mentor Graphics [88] and MAST AHDL by Analogy [89]. However, the lack of a portable and widely adopted HDL standard restricted the user to the vendor's own HDL and thus made it impossible to import models between different EDA tools; in contrast to the ease of modelling and simulation of digital systems with VHDL [82], or Verilog [90]. The emergence of VHDL-AMS has changed this and is likely to accelerate the development of more efficient and powerful modelling tools for mixed-system and mixed-domain applications. This is apparent, as recently other mixed-signal extensions to existing languages have emerged, for example Verilog-AMS [91] for Verilog, and SystemC-AMS [92] that uses C++ as the hardware description language.

Several VHDL-AMS compilers have been made available for public use [93-96], including one from University of Southampton [94] which can be accessed from the Internet. Commercial VHDL-AMS simulators are still under development and only a few vendors have so far released their products, for example VeriasHDL from Analogy [97], SystemVision from Mentor Graphics [88, 98], SMASH [99] and Hamster [100].

3.2 VHDL-AMS for synthesis

Although VHDL-AMS is primarily intended for modelling and simulation, a suitable subset for mixed-signal synthesis has been proposed even before the formal release of the standard [22] and used in the development of a mixed-signal synthesis tool named VASE [14, 18]. VASE is reviewed in Chapter 2. Similarly, VHDL-AMS has been used by the Electronic Systems Design Group in University of Southampton for the development of NEUSYS [15], a synthesis tool for artificial neural networks. The work on NEUSYS concentrated around the use of VHDL-AMS for biologically inspired models of complex neural network systems developed in an earlier work [87]. Such systems involve many analogue equations in a mixed-signal environment.

3.2.1 Features suitable for synthesis

VHDL-AMS supports continuous behaviour by introducing new kinds of objects into the classical, digital VHDL. The most important VHDL-AMS features relevant to this research are briefly presented below:

3.2.1.1 Quantities

`quantity` is an object used to represent the unknowns in the DAEs that form the analogue model. In the context of an electronic system, quantities can represent voltages, currents, capacitor charges, inductor fluxes, and so on. A `quantity` can be declared as an interface element in a `port` list of a model's `entity` declaration. This mechanism creates analogue nets which connect `entity` instances together. For example, a filter `entity` with two quantities representing the input and output can be simply described as follows:

```
entity filter is
port (quantity Vin: real;
  quantity Vout: out real);
end entity filter;
```

Another mechanism for creating analogue nets is provided by `terminal` as described in Section 4.3.1.5 in the VHDL-AMS language reference manual (LRM) [1]. A number of standard, predefined attributes have been specified in the VHDL-AMS LRM for quantities, some of these attributes are themselves quantities. For example, a new `quantity` may be implicitly declared using the attribute Q'DOT to represent a time derivative of `quantity` Q. The attribute Q'DOT is essential for describing ordinary differential equations in VHDL-AMS models. Another attribute, Q'INTEG, allows the declaration of the integral of `quantity` Q over time.

3.2.1.2 Simultaneous statements

Simultaneous statements represent the system's DAEs. They express the relationships between quantities. As mentioned in [83], simultaneous statements contain ordinary VHDL expression that can be evaluated in the ordinary way. The behavioural description for electronic circuit, implementing the conservation semantics of Kirchoff's law, is supported using this new class of statements, where predefined attributes, allowing quantities having derivatives or integrals over time may be used in a particular expression. For example, a simple simultaneous statement (SSS) for a DAE in terms of the quantitites `Vin` and `Vout` can be formulated as follows:

```
Vin == τ *Vout'DOT + Vout;
```

which represents a first-order lowpass filter in the time-domain, which is expressed as in (3.1).

$$Vin(t) = \tau \frac{dVout(t)}{dt} + Vout(t) \tag{3.1}$$

3.2.1.3 Predefined attribute - Q'LTF

Q'LTF is a predefined attribute for a `quantity` object which may be used to describe the Laplace transfer function of a system. It takes two one-dimensional vectors as parameters that provide the

numerator and denominator coefficients of the transfer function arranged in the ascending order. In a realisable and hence, a synthesisable transfer functions, the order of the denominator must be greater than or equal to that of the numerator. The LTF attribute provides the most suitable way to model filter transfer functions in the frequency domain. The VHDL-AMS syntax for Q'LTF is shown below:

```
Vout == Vin'LTF(num,den);
```

where `Vin` is the input to the system, `Vout` is the output and `num` and `den` are the coefficients of the system's Laplace transfer function. Note that when the relationship between the input and output quantities is expressed using the LTF attribute, it must be expressed using the SSS syntax as shown above. `num` and `den` must be static expressions of type `real_vector`.

In most examples of frequency-domain filter descriptions used in this dissertation, filter coefficients are defined using static expressions of the type shown below, which hold the values for typical filter parameters such as the gain, cut-off frequency, Q factor, natural frequency and so on:

```
constant frequency:real:= 1e3;
constant Q:real:= 10.0;
constant w:real:= 2*3.142*frequency;
constant a:real:= w*w;
constant b:real:= w/Q;
```

This is a neat and convenient way to specify filter parameters without having to pre-calculate them prior to writing the VHDL-AMS code.

3.3 Modelling synthesisable analogue filters using VHDL-AMS

As mentioned above, analogue filters can be described in VHDL-AMS using two different ways: as time-domain models or alternatively as frequency-domain Laplace transfer functions. Conventionally, the latter method is the more common way to represent a filter to a synthesis system (for example ARCHGEN [69] and BECAS [74]). However, if the filter is to be integrated with digital components in a system-on-chip (SOC) - for example in a communication system, where the digital part can also be modelled using VHDL-AMS - it may be more practical to describe the filter in the time-domain using DAEs.

The following subsections will present the modelling of filter requirements in both the time- and frequency-domain, using VHDL-AMS. Filter described in the time-domain is referred to as the DAE construct, while filter described in the frequency-domain is known as the LTF construct. The synthesisable aspect of the behavioural models will also be explained.

3.3.1 Time-domain modelling with DAE construct

Equation (3.2) is the description of a second-order lowpass filter in the time-domain.

$$V_{in}(t) = Coeff_1 \times \frac{d^2 V_{out}(t)}{dt^2} + Coeff_2 \times \frac{dV_{out}(t)}{dt} + Coeff_3 \times V_{out}(t) \qquad (3.2)$$

Thus in VHDL-AMS this is implemented using SSS with quantities using the Q'DOT attribute to represent time-derivatives, as shown below. The following filter description is specified to have cut-off frequency of 1 GHz and Q factor of 50.

```
architecture behavioural of filter is
      constant pi: real:=3.142;
      constant frequency: real:=1.0e9;
      constant w: real:= 2.0*pi*frequency;
      constant Q: real: = 50.0;
      constant coeff1: real:= 1/(w*w);
      constant coeff2: real:= 1/(Q*w);
      constant coeff3: real:= 1.0;
begin
      vin == coeff1*vout'dot'dot + coeff2*vout'dot +
coeff3*vout;
end architecture;
```

Other examples of the time-domain descriptions of second-order filter cells are shown in Table 3.1. The DAE construct can be used to describe filters in terms of the required gain, frequency and Q factor. Expressions for coefficients $Coeff_1$, $Coeff_2$ and $Coeff_3$ (Table 3.1) in terms of these parameters are shown in Table 3.2, where G refers to the filter gain, ω_0 refers to the natural frequency, and Q is the Q factor. Descriptions of this kind can be synthesised from the corresponding expression clusters in VHDL-AMS parse trees. Examples of equivalent VHDL-AMS simultaneous statements that can be used to generate such trees are shown in the third column of Table 3.1. The synthesiser can determine the required filter type from the left-hand side (LHS) of the corresponding time-domain model, shown in the third column of Table 3.1, while the right-hand side (RHS) of the SSS are identical in each case. Thus the second-order filter type can be determined from the LHS of its DAEs when written in such a form.

2nd-order type	Time-domain equation	Example of an equivalent VHDL-AMS simultaneous statement
Lowpass	$V_{in}(t) =$ $Coeff_1 \times \frac{d^2V_{out}(t)}{dt^2} + Coeff_2 \times \frac{dV_{out}(t)}{dt} + Coeff_3 \times V_{out}(t)$	vin == coeff1*vout'dot'dot + coeff2*vout'dot + coeff3*vout
Bandpass	$\frac{dV_{in}(t)}{dt} =$ $Coeff_1 \times \frac{d^2V_{out}(t)}{dt^2} + Coeff_2 \times \frac{dV_{out}(t)}{dt} + Coeff_3 \times V_{out}(t)$	vin'dot == coeff1*vout'dot'dot + coeff2*vout'dot + coeff3*vout
Highpass	$\frac{d^2V_{in}(t)}{dt^2} =$ $Coeff_1 \times \frac{d^2V_{out}(t)}{dt^2} + Coeff_2 \times \frac{dV_{out}(t)}{dt} + Coeff_3 \times V_{out}(t)$	vin'dot'dot == coeff1*vout'dot'dot + coeff2*vout'dot + coeff3*vout

Table 3.1 Second-order filter cells in the time-domain.

2nd-order type	*Coeff1*	*Coeff2*	*Coeff3*
Lowpass	$\frac{1}{G \cdot \omega_0^2}$	$\frac{1}{G \cdot Q \cdot \omega_0}$	$\frac{1}{G}$
Bandpass	$\frac{Q}{G \cdot \omega_0}$	$\frac{1}{G}$	$\frac{Q \cdot \omega_0}{G}$
Highpass	$\frac{1}{G}$	$\frac{\omega_0}{G \cdot Q}$	$\frac{\omega_0^2}{G}$

Table 3.2 Time-domain description coefficients in terms of filter parameters for second-order DAE constructs.

A second-order cell has three terms in the RHS, namely: *Coeff1*, *Coeff2* and *Coeff3* as shown in Table 3.2. These coefficients may themselves be expressions represented as clusters in the parse tree. If these expressions are static in the VHDL sense, they can be evaluated by scanning the corresponding clusters and subsequently mapped into hardware. Section 3.5.2 describes the static calculator that has been specifically developed for this purpose as part of the architectural synthesiser. Written in this form, the static calculator will be able to get the filter specification that will be used to guide the filter synthesis process.

3.3.2 Frequency-domain modelling with LTF construct

The alternative method to describe filters in VHDL-AMS is to use the frequency-domain and built-in predefined Q'LTF attribute representing a Laplace transfer function. For example, a general, second-order filter cell can be presented in the frequency-domain by the transfer function *H(s)*

shown in (3.3), where k_1, k_2, and k_3 can assume the value +1, -1 or 0; while ω_0 and Q are the natural frequency and Q factor, respectively [101]. The synthesiser can determine the required filter type by analysing the values of k_1, k_2, and k_3.

$$H(s) = \frac{k_1 s^2 + k_2(\omega_0 / Q)s + k_3 \omega_0^{\ 2}}{s^2 + (\omega_0 / Q)s + \omega_0^{\ 2}} \tag{3.3}$$

For example, a second-order lowpass filter behaviour occurs when $k_1 = k_2 = 0$ and $k_3 = 1$ in the transfer function of (3.3). The filter can be described in VHDL-AMS as follows:

```
entity filter is
      port (quantity Vin: real;
            quantity Vout: out real);
end entity filter;

architecture transfer of filter is
      constant pi: real:=3.142;
      constant frequency: real:=1.0e9;
      constant w: real:= 2.0*pi*frequency;
      constant Q: real: = 50.0;
      constant a:real:= w*w;
      constant b:real:= w/Q;
      constant num: real_vector:= (a);
      constant den: real_vector:= (a,b,1.0);
begin
       Vout == Vin'LTF(num,den);
end architecture;
```

Behavioural descriptions of filters of arbitrary orders can normally be found very easily. There exist many filter design tables that can be used in specifications of filter transfer functions [102]. The polynomial coefficients for the filter's transfer function can be obtained directly from filter tables, and are usually normalised to a cut-off frequency of 1 rad/s. Furthermore, frequency transformation techniques [2] can be applied to lowpass filter prototype from design tables, to produce other filter types, such as bandpass, highpass and others. For example, after applying frequency transformation to a Butterworth lowpass filter, the following fourth-order bandpass filter transfer function is obtained:

$$H(s) = \frac{as^2}{b + cs + ds^2 + es^3 + s^4} \tag{3.4}$$

where a,b,c,d,e are coefficients. After de-normalising to 1 GHz, the following VHDL-AMS code can be written to represent the required transfer function:

```
constant a: real:= 0.2e18;
constant b: real:= 1.551e39;
constant c: real:= 1.114e28;
constant d: real:= 7.880e19;
constant e: real:= 0.2828e9;
constant num: real_vector:= (0,0,a);
constant den: real_vector:= (b,c,d,e,1.0);
...
Vout == Vin'LTF(num,den);
```

3.4 VHDL-AMS parse tree

The parse tree is an intermediate version of a VHDL-AMS description. In the context of this work, the parse tree is used to represent an analogue filter, which may be described using either the time-domain model of Section 3.3.1 or the frequency-domain model of Section 3.3.2. Parse tree representations have been used for verification [103], analysis and modelling [104, 105] of electronic circuit applications. Examples of parse tree representations of analogue filter are illustrated in Figure 3.1 and Figure 3.2.

Figure 3.1 (a), (b) and (c) show three parse tree representations generated for a second-order lowpass, bandpass and highpass filter, respectively. In these examples, filter descriptions are specified in terms of Simple Simultaneous Statements (SSS), which represent the time-domain DAE constructs outlined in Table 3.1.

In Figure 3.1, the VHDL-AMS quantities representing the input and the output of the filter are referred to as (in)Q and (out)Q, respectively. Note that the right hand side expressions of the filter equations in Table 3.1 are similar, which consist of three terms. These terms are represented as two-child *clusters* as shown in each parse tree of Figure 3.1. The clusters specify information regarding the filter coefficients as well as the time derivative order of the output quantity (out)Q associated with each coefficient. The filter coefficients may be a constant or expressed in the form of a Simple Equation (SE). As mentioned in Section 3.3.1, the filter type is determined by the left hand side of the SSS, which specifies the time derivative order of the input quantity (in)Q.

On the other hand, Figure 3.2 depicts a generalised parse tree representation of the analogue filter expressed with an LTF construct, explained in Section 3.3.2. Unlike the DAE model, the analogue filter type of the LTF filter construct is implicitly determined by evaluating the numerator and denominator of the filter's LTF. Each of the transfer function's numerator and denominator coefficients has its own parse tree structure, depending on the filter specification. A more elaborate discussion on the parse tree structures of an LTF construct is provided in Section 3.5 in the context of Case Study 2.

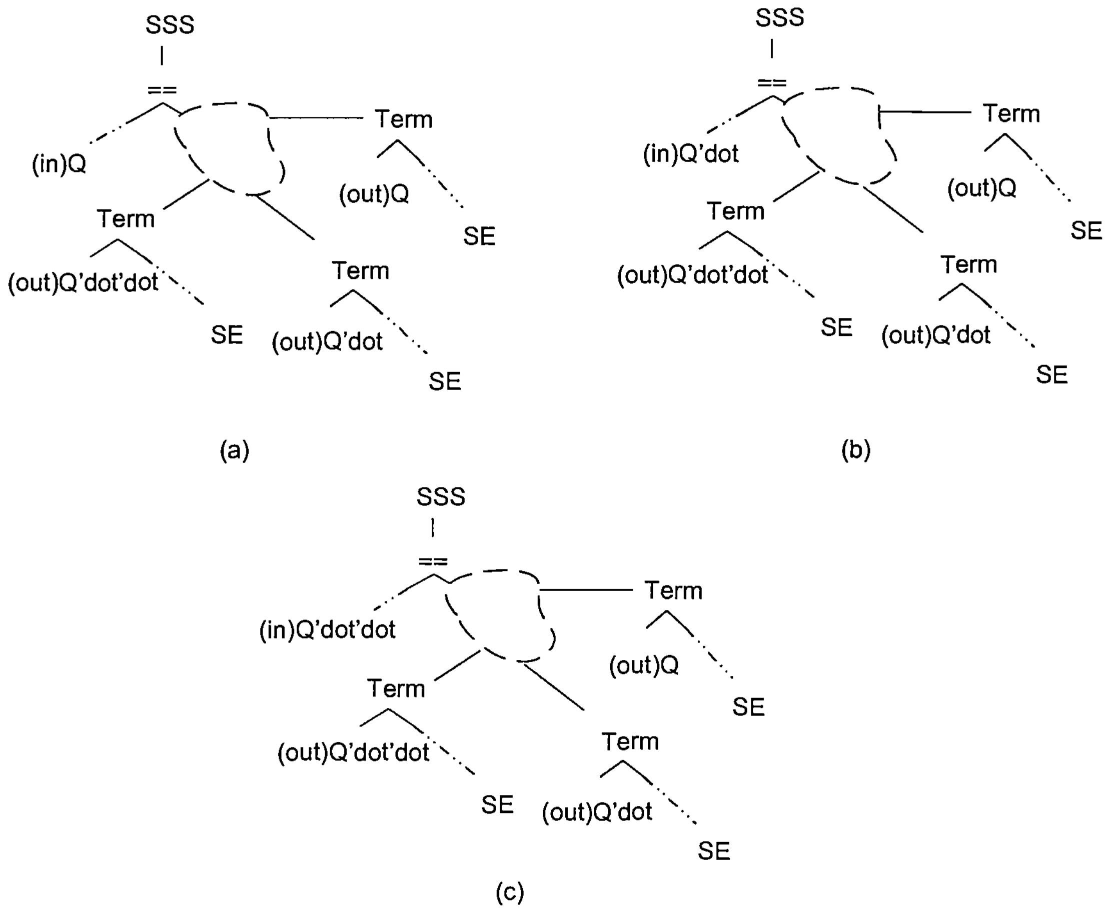


Figure 3.1 Example of VHDL-AMS parse tree patterns representing second-order DAE filter construct, (a) lowpass, (b) bandpass, (c) highpass.

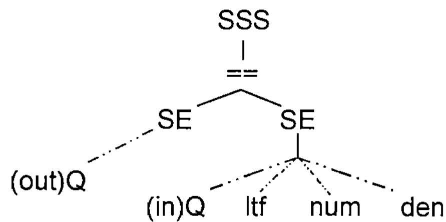


Figure 3.2 Example of the VHDL-AMS parse tree pattern representing an LTF filter construct.

3.4.1 Parser

The parser is a software module that is used to translate any syntactically correct VHDL-AMS description into a parse tree representation. This module is based on the VHDL-AMS parser developed by the Electronics Systems Design (ESD) group of University of Southampton [94]. It provides a full implementation of the VHDL-AMS syntax defined by the IEEE standard 1076.1-1999 [1], and has been made available to the design and research communities worldwide as a free on-line tool [94, 106]. The parser is a top-down LL(k) [107] tree parser.

The main tasks of the parser are lexical analysis, symbol hash table maintenance, declaration scope maintenance, parse tree generation and semantic checks. In a parse tree structure, the leaf nodes in each cluster, which are child nodes without any descendants or brothers, are VHDL-AMS lexical tokens, such as identifiers, decimal literals, operator, separators or keywords. This is true for any VHDL-AMS parse tree produced by the parser.

As mentioned earlier, the parse tree is only an intermediate representation of the VHDL-AMS analogue filter description. The parser will generate a parse tree for any syntactically correct VHDL-AMS description, even though it does not constitute any synthesisable element. In other words, the parse tree is another way of representing the information contained in the VHDL-AMS simultaneous statements. This representation is the basis of analogue filter synthesis in this research, which focuses on methods and algorithms to convert synthesisable patterns in the parse tree to filter structure. Therefore, dedicated software modules have been developed for the sake of identifying as well as extracting synthesisable analogue filter elements from the parse tree. These tasks are carried out by the *synthesis syntax checker* and *static calculator* modules described in Section 3.5.

The relationship between the parser and the two software modules with other processes in the synthesis procedure is shown below in Figure 3.3. In particular, this block diagram shows how the parser and the modules are linked to the filter cell library to finally produce an analogue filter netlist that matches the VHDL-AMS specification of the filter. As can be seen, there is no direct link between the parser and the filter library. The parser, synthesis syntax checker and static calculator produce filter information that is used to build data in terms of AC response compatible with the HSPICE simulator. This data is used in the parametric and architectural optimisation stage that select suitable filter topology(s) from a library. The final result, or output of this process is an HSPICE netlist of the filter that closely matches the VHDL-AMS specification. The details for these procedures are given in Chapters 4 and 5. The focus of the following sections in this chapter is to firstly explain how the parser is used to obtain synthesisable analogue filter descriptions as well as the modifications being made to the parser. Then the operation of the synthesis syntax checker and static calculator that extract information regarding the filter is explained with the aid of two case studies.

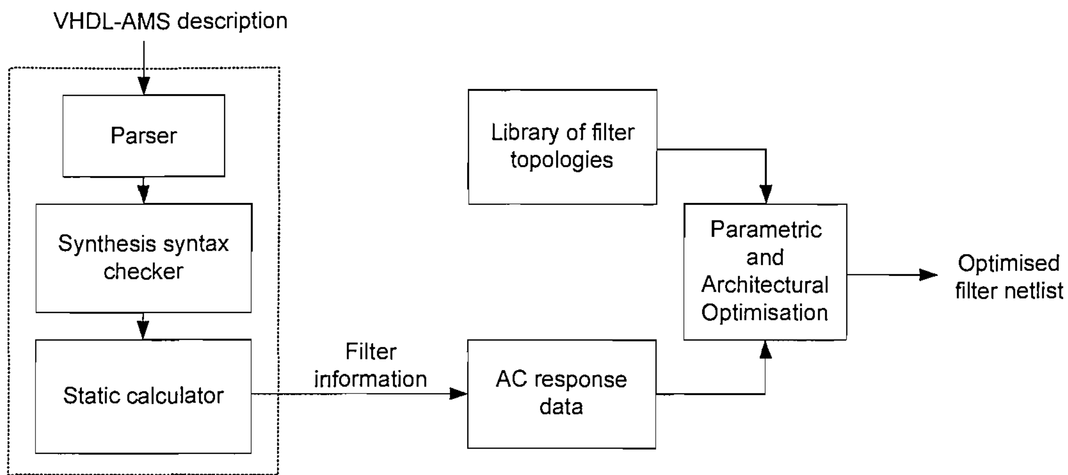


Figure 3.3 Block diagram showing the behavioural synthesis of analogue filter, starting from the input of a filter's VHDL-AMS description until obtaining the filter's optimised circuit netlist.

3.4.1.1 Using the parser for synthesis

As mentioned before, the main functionality of the parser is to produce a parse tree structure that will be analysed by the synthesis syntax checker and static calculator modules. This section will firstly explains parts of the parser that is being used, and as well as how it is being used and the modifications being made to the original parser in order to allow the parser to be linked to and utilised by the two software modules (i.e. the synthesis syntax checker and the static calculator).

The parser engine is implemented in C++, and constitute of several classes. However, the main class being used is the parse tree module which builds the parse tree structure, called class *'ParseTreeT'*. The functions or methods that belong to this class and being used in this work are explained below:

- *FindNodeInCluster():* Searches for a particular 'node kind' in a particular parse tree cluster. A 'node kind' is a node classification according to the grammatical syntax or identifier specified in the IEEE Language Reference Manual (Appendix A of [1]). Examples of node kind are "ArchitectureBodyE", "SimultaneousStatementE" and "SimpleNameE".
- *FindBrotherNode():* Looks for a 'node kind' in a brother node. The difference between this function and the previous one is that this function allow the scope of search to be limited to just the clusters in the same level of the current parse tree hierarchy, i.e. in all the brother nodes of the current parse tree node.
- *FindIdentifierInCluster():* Searches for a particular identifier in a particular tree cluster.

The three functions above are very similar in which the task is to look for a 'node kind' or an 'identifier' in a particular tree cluster. 'Identifier' are variable names, types as well as the numerical values assigned to those variables. For example, a constant is declared in VHDL-AMS as follows:

```
constant a: real:= 0.2e18;
```

When being parsed, 'a', 'real' and '0.2e18' are classified by the parser as 'identifiers'.

Other functions that are being used from *'ParseTreeT'* are *TestPrint()* and *TestPrintPTCluster()* that are used to generate test prints of parse tree sections or clusters for logging and debugging purposes. Finally, another two methods used (mainly for debugging) are:

- *KindString():* to check the 'node kind'
- *NameString():* to check the name of the 'identifier'

These functions return an array of characters that are printed in a log file.

Therefore, it can be seen that in order to allow the above functions to be used by the synthesis syntax checker and static calculator, modifications must be made to solve accessibility issues:

- A function to set a particular parse tree node as a parent node is added to the main parse tree module, *'ParseTreeT'*: *SetParent()*. *SetParent()* is used in the function written for the synthesis syntax checker module to copy the parse tree (See function *CopyParseTree()* in Table 3.3, Section 3.5.1.3).
- The function *NameString()*, in class *'ParseTreeT'* are declared as public instead of private so that it can be directly accessed by the synthesis syntax checker and static calculator. The other functions described in the paragraphs above are already declared as public.

Also, in order for the whole parser engine to be able to be used by the synthesis system (called FIST- Filter Synthesis Tool, which is described in Chapter 5) as a whole, the parser is included in the synthesis system as a dynamically linked library (DLL). Several functions from the parser are used as imported functions, and are listed as follows:

- *AVAMSRunCompiler():* running the compiler by specifying relevant directory paths and the VHDL-AMS input file.
- *AVAMSGetParseTree():* obtaining the parse tree structure of the VHDL-AMS input file.
- *AVAMSCompilerScopeTestPrint():* to print the parse tree structure into a file.
- *AVAMSDeleteCompiler():* to delete the compiler.

3.5 Software modules for behavioural synthesis of analogue filters from VHDL-AMS parse trees

This section describes the software implementation developed for the purpose of extracting the synthesisable analogue filter information from the VHDL-AMS parse tree. The structure of the

software implementation is divided into two modules, namely the *synthesis syntax checker* module and the *static calculator* module. The block diagram of the modules is illustrated in Figure 3.4.

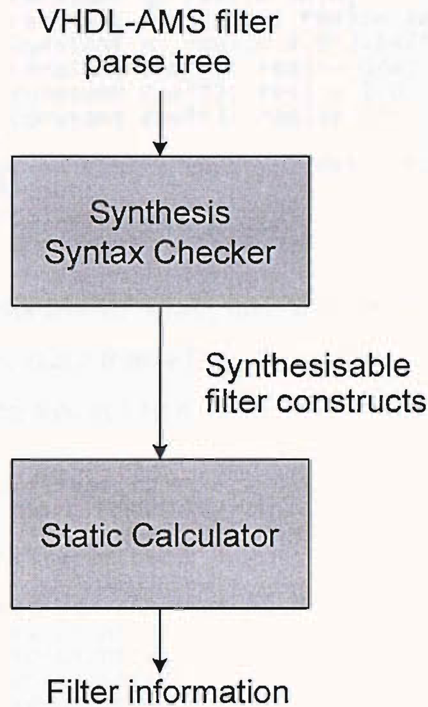


Figure 3.4 Software modules to extract filter information from VHDL-AMS parse tree.

The synthesis syntax checker module analyses the parse tree structure in order to identify synthesisable analogue filter constructs, which are then processed by the static calculator. The static calculator module evaluates the static expressions in the parse tree structure of the VHDL-AMS filter constructs to extract filter information. For a DAE construct, the filter type and coefficient values (`coeff1`, `coeff2` and `coeff3`) are obtained, while for an LTF construct, the system's transfer function is obtained. Such information obtained from both the DAE and LTF constructs are then used to acquire analogue filter structures, which will be discussed in Chapter 4. For the case of extracting analogue filter information from VHDL-AMS parse trees, the following subsections will further describe both the synthesis syntax checker and static calculator modules in more detail with the aid of two case studies.

Case Study 1 is a second-order bandpass filter described in the time-domain; with centre frequency of 1 GHz, and Q factor of 50. The corresponding VHDL-AMS model of the filter is shown below:

```
entity filter is
      port  (quantity vin: real;
             quantity vout: out real);
end entity filter;

architecture behavioural of filter is
      constant Q: real:= 50.0;
      constant frequency: real:= 1e9;
      constant w: real:= 2.0*3.142*frequency;
      constant coeff1: real:= Q/w;
      constant coeff2: real:= 1.0;
      constant coeff3: real:= Q*w;
begin
vin'dot == coeff1*vout'dot'dot + coeff2*vout'dot +
coeff3*vout;
end architecture;
```

Case Study 2 is a lowpass fourth-order Chebyshev filter with passband ripple of 0.5dB. The transfer function coefficients are taken from a filter design table [108], de-normalised to the cut-off frequency of 1 GHz and implemented in VHDL-AMS as follows:

```
entity filter is
       port (quantity Vin: real;
             quantity Vout: out real);
end entity filter;

architecture transfer of filter is
      constant frequency: real:=1e9;
      constant w: real:= 2.0*3.142* frequency;
      constant w2: real:= w*w;
      constant w3: real:= w2*w;
      constant w4: real:= w2*w2;
      constant a:real:=0.3579*w4;
      constant b:real:=0.3791*w4;
      constant c:real:=1.0255*w3;
      constant d:real:=1.7169*w2;
      constant e:real:=1.1974*w;
      constant num: real_vector:= (a);
      constant den: real_vector:= (b,c,d,e,1.0);
begin
       Vout == Vin'LTF(num,den);
end architecture;
```

The details of the software implementation for both the synthesis syntax checker and static calculator are given in Sections 3.5.1.3 and 3.5.2.3 respectively.

3.5.1 Synthesis syntax checker

The synthesis syntax checker traverses the VHDL-AMS parse tree to look for known synthesisable DAE and LTF filter constructs. It recognises filters that are behaviourally modelled as explained in Section 3.3. The parse tree is analysed by firstly finding a specific tree node and then examining the structure of the tree cluster of that node. For both the DAE and LTF construct, the synthesis syntax checker will search for an SSS tree cluster that have two child nodes in the form of an SE. For a DAE construct, the first SE, which is referred to as SE(1) contains information regarding the `quantity` denoting the input term of the DAE, while for an LTF construct, SE(1) have a very simple tree structure that holds the name of the `quantity` denoting the output of the Laplace transfer function of the system. The second SE, SE(2) is analysed in more detail by the synthesis

syntax checker to determine whether the VHDL-AMS parse tree is of an LTF construct, DAE construct or neither. Upon encountering an unidentifiable tree structure, the synthesis syntax checker will exit and return an error message. When traversing and analysing the parse tree structure of SE(2), and finding a DAE or LTF construct, information regarding the filter is also obtained by the synthesis syntax checker. This information is then passed on to the static calculator for further manipulation. The operation of the synthesis syntax checker on both the DAE and LTF construct is demonstrated in the following subsections.

3.5.1.1 Synthesis syntax check for Case Study 1

This example illustrates the synthesis syntax checker's operation using a filter modelled in the time-domain. The VHDL-AMS architecture description of the filter is shown in Figure 3.5. It is a bandpass filter with a Q factor of 50, gain of 1 and cut-off frequency of 1 GHz. The structure of the description as defined by the IEEE Standard 1076.1-1999 [1] is also shown, where at the top of the hierarchy of the description is the architecture body (AB), which is further divided into the architecture declarative part (ADP) and architecture statement part (ASP).

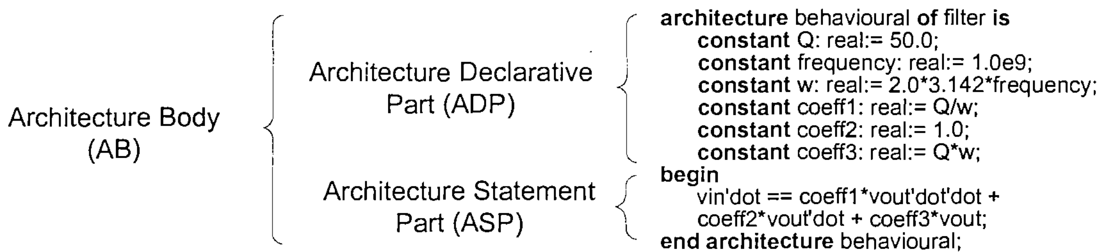


Figure 3.5 Architecture body of the VHDL-AMS code of the second-order bandpass filter.

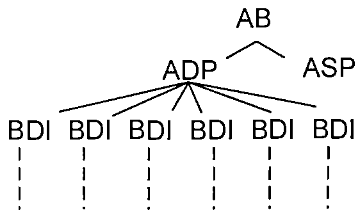
Synthesis syntax check is done by investigating the tree cluster starting at the ASP node, where the main features of the filter description or behaviour may exist. The synthesis syntax checker analyses the ASP cluster to determine whether the description is an LTF construct or a DAE construct, and then follows the corresponding LTF or DAE routines in order to obtain the synthesisable expressions.

The ADP of the architecture body in Figure 3.5 has six constant declarations which are each organised in the parse tree structure as block declarative items (BDI). Figure 3.6 (a) depicts the parse tree structure of the architecture body down to two descendent levels. Figure 3.6 (a), (b) and (c) show the details of the parse tree structure of the architecture body of Case Study 1 except

for the descendant clusters from the six BDIs of the ADP. The details for the BDI clusters will be discussed in the section on the static calculator for this case study.

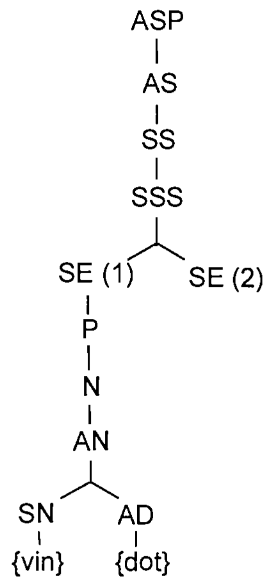
The details of the ASP in Figure 3.6 (b) and (c) illustrates a typical DAE parse tree, where there is an SSS representing the filter equation. The SSS has two simple expressions, SE(1) and SE(2), which are analysed to determine synthesisability. Figure 3.6 (c) shows the parse tree structure of SE(2).

The flow chart in Figure 3.7 outlines the methodology of recognising a synthesisable DAE construct. The RHS of the SE, which is marked as SE (2) in the flow chart, is examined to match the tree cluster pattern as shown in Figure 3.6(c). When a matching tree cluster is found, the syntax checker collects the required coefficient values (`coeff1, 2` and `3`) in an array, and proceeds to analyse the filter type of the description.

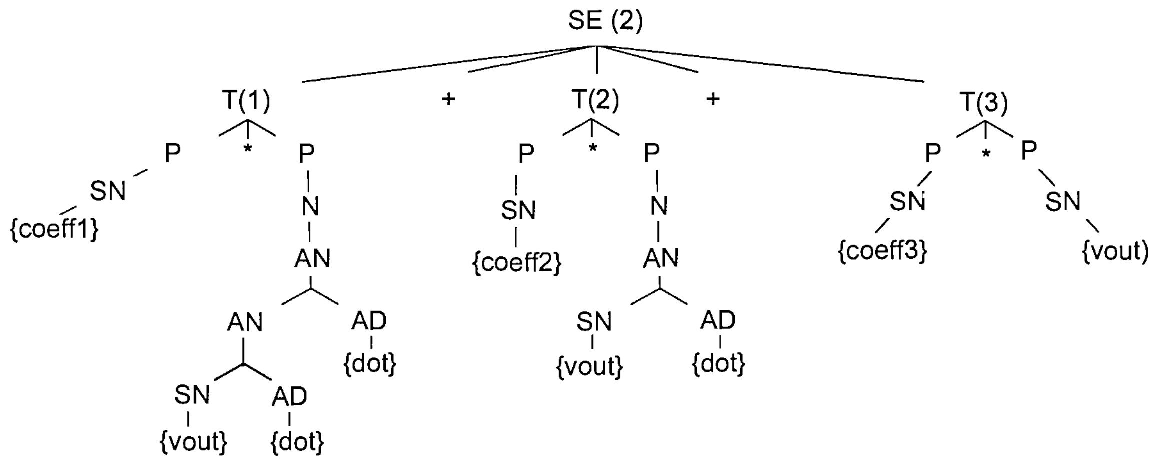


(a)

AB - Architecture Body
ADP - Architecture Declarative Part
ASP - Architecture Statement Part
BDI - Block Declarative Item
AS - Architecture Statement
SS - Simultaneous Statement
SSS - Simple Simultaneous Statement



(b)



SE - Simple Expression
T - Term
P - Primary
N - Name
AN - Attribute Name
SN - Simple Name
AD - Attribute Designator
{..} - identifier

(c)

Figure 3.6 (a) The AB cluster of Case Study 1, (b) The ASP tree cluster, (c) Tree cluster of the second SE in the ASP cluster.

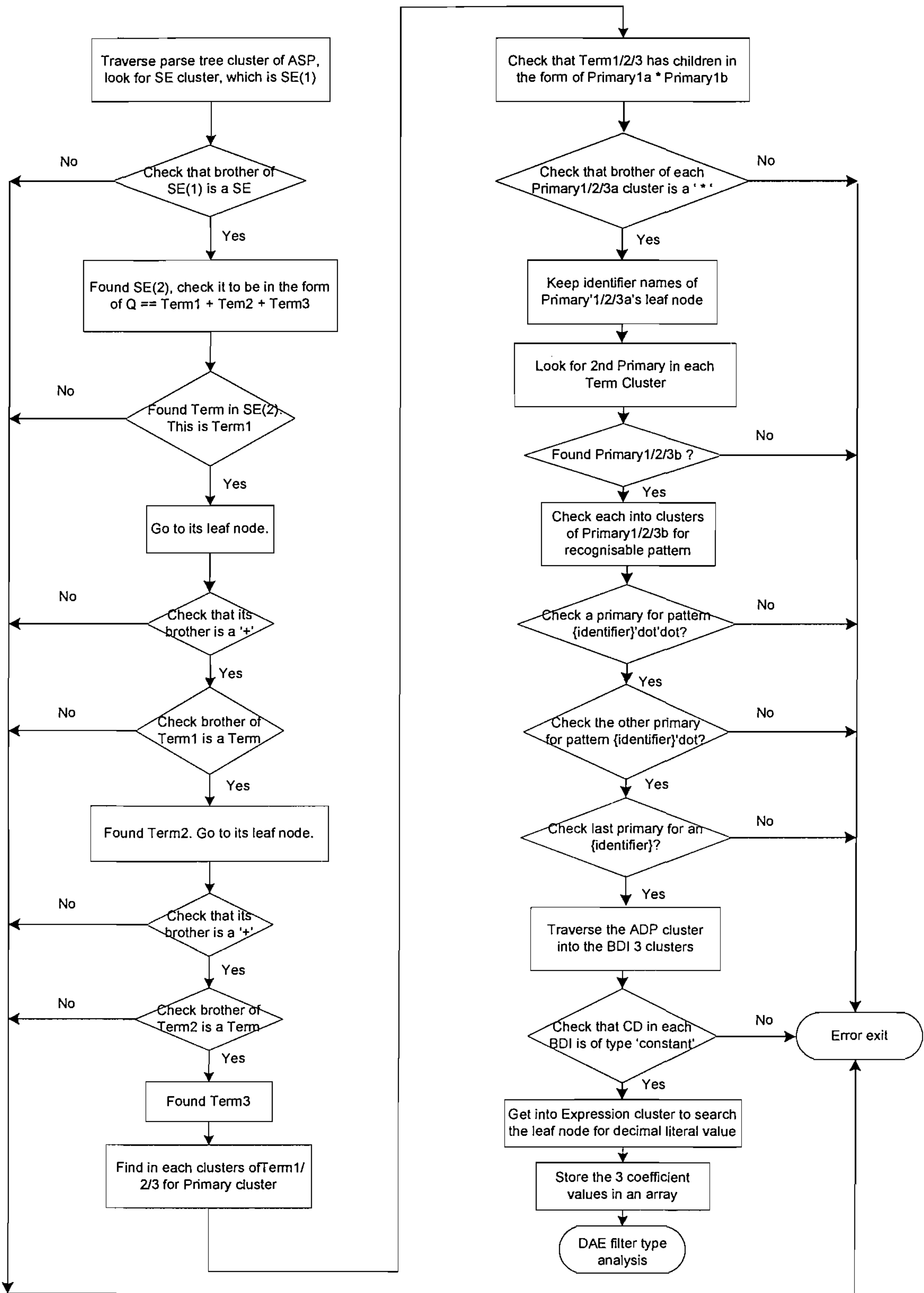


Figure 3.7 Flow chart for the routine that searches for synthesisable DAE constructs in SE (2).

After a recognisable DAE construct is found, the synthesiser calls the DAE filter type analysis routine. The purpose of this routine is to identify the filter type by examining SE (1). This process

is illustrated in Figure 3.8. When the filter type has been identified, the static calculator is called to examine SE (2). The static calculator will be explained further in Section 3.5.2.1.

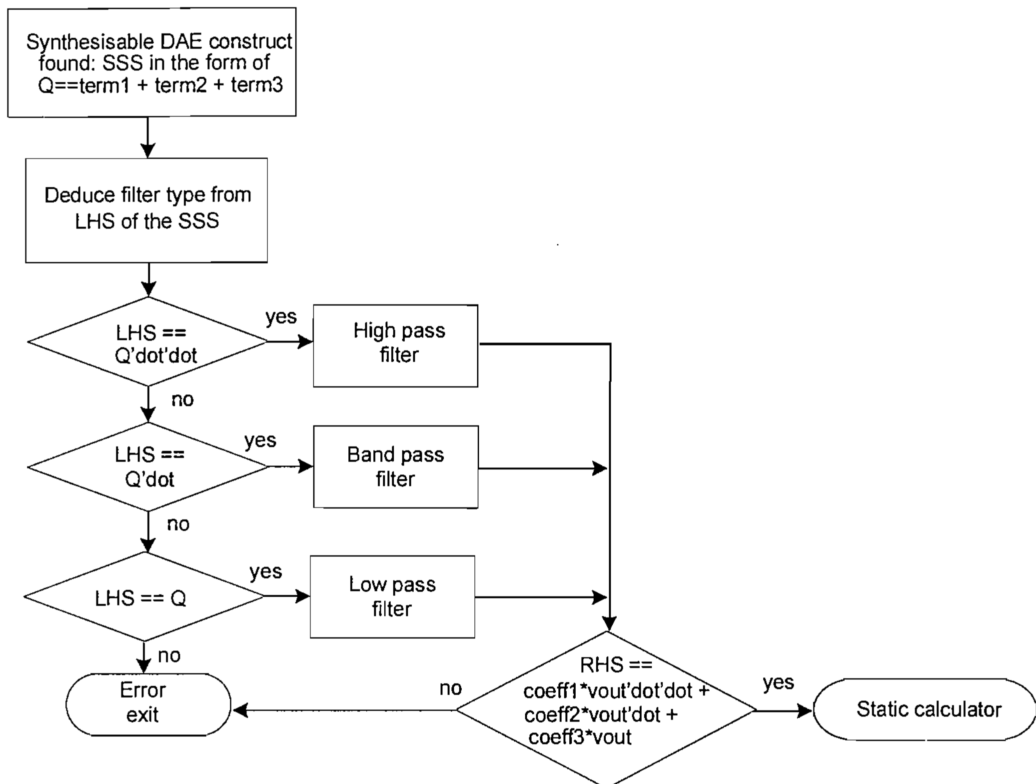


Figure 3.8 DAE routine for filter type identification.

3.5.1.2 Synthesis syntax check for Case Study 2

The fourth-order lowpass Chebyshev filter modelled with the built-in LTF attribute has an architecture body structure as shown in Figure 3.9. The synthesis syntax checker firstly examines the ASP cluster, shown in Figure 3.10, and finds the LTF attribute and associated SSS. The flow chart of the algorithm to search for an LTF construct is shown in Figure 3.11.

After confirming that the parse tree structure of the ASP is of an LTF construct, the programme then proceed to the static calculator module.

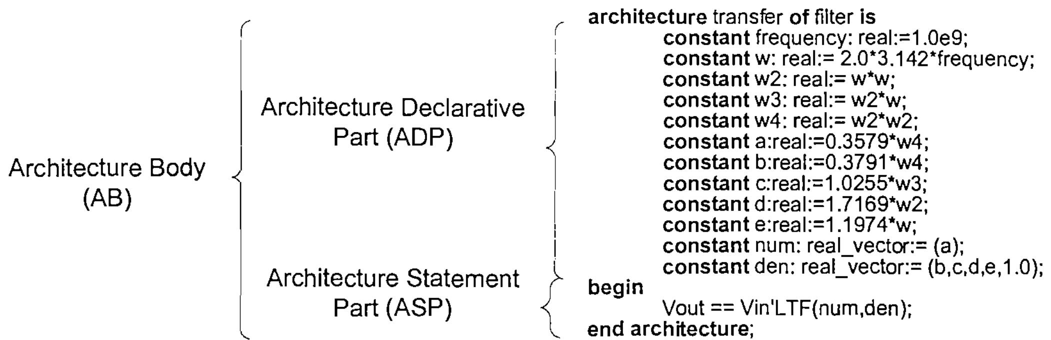


Figure 3.9 Architecture body of the VHDL-AMS code of Case Study 2.

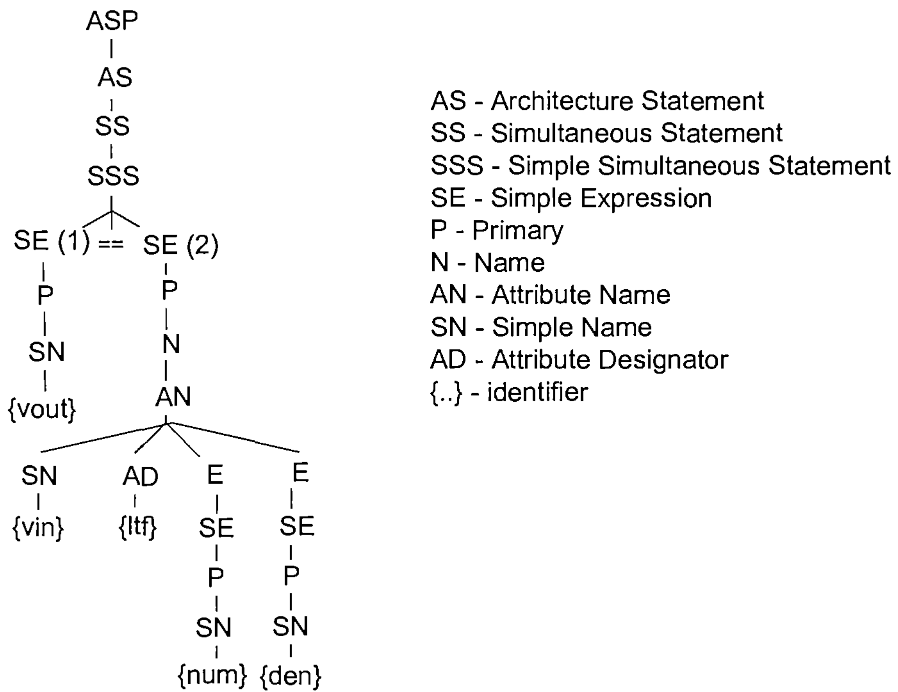


Figure 3.10 ASP tree cluster for an LTF construct.

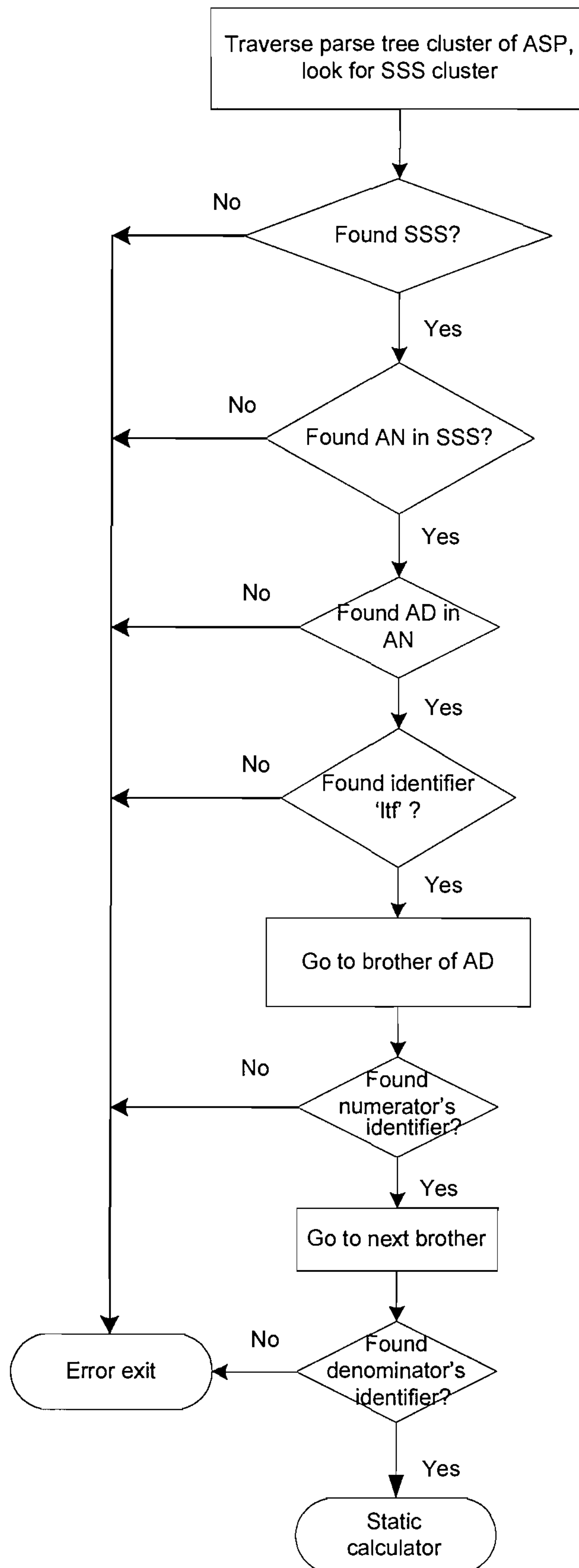


Figure 3.11 Flow chart of synthesis syntax check in an LTF construct.

3.5.1.3 Software implementation

The synthesis syntax checker is implemented in the class named *ParseTreeTS*. This section describes the important methods used in *ParseTreeTS* that perform parse tree analysis, and are listed in Table 3.3.

Name and Syntax	Return Value	Description
void MakeCopyofParseTree (ScopeRecPT OldScope)		It copies the parse tree produced by *AVAMSGetParseTree()*. 'OldScope' is the pointer returned by *AVAMSGetParseTree()*. It points to the beginning of a scope record which contains the parse trees of the entity and architecture declarations. The program loops the parent pointer of each scope record, allocate memory for new scope record and calls *CopyParseTree()*.
PTNodePT CopyParseTree (PTNodePT Cluster)	Pointer to the new parse tree.	Makes another copy of the parse tree. This method copies nodes of tree clusters recursively.
int Search(void)	an integer to indicate error (0), DAE high pass filter (1), DAE band pass (2), DAE low pass (3), and an LTF construct (4)	It first searches the parse tree for LTF filter construct, then try to look for DE filter construct. It calls *SearchLTF()* to find an LTF construct, and *SearchDAE()* for a DAE construct.
StatusT SearchLTF(void)	OK or ERROR status.	Searches the parse tree for an LTF construct. This method is called by *Search()*. Figure 3.10 shows a parse tree cluster that will be recognised by this method. Figure 3.11 shows the algorithm that is being implemented in this method.
StatusT SearchDAE(void)	OK or ERROR status.	Searches the parse tree for a DAE construct. This method is called by *Search()*, and calls *CheckPrimaryPattern()* to determine its return value.
int CheckPrimaryPattern (PTNodePT primary)	An integer indicating which of the three pattern was found, or 0 if no pattern has been found.	Scans a primary cluster for 3 recognisable parse tree clusters in a DAE construct. Refer to Figure 3.6 (c) for the parse tree pattern shown by the three term clusters marked as T(1) – T(3), and Figure 3.7 for the algorithm implementing part of this function.

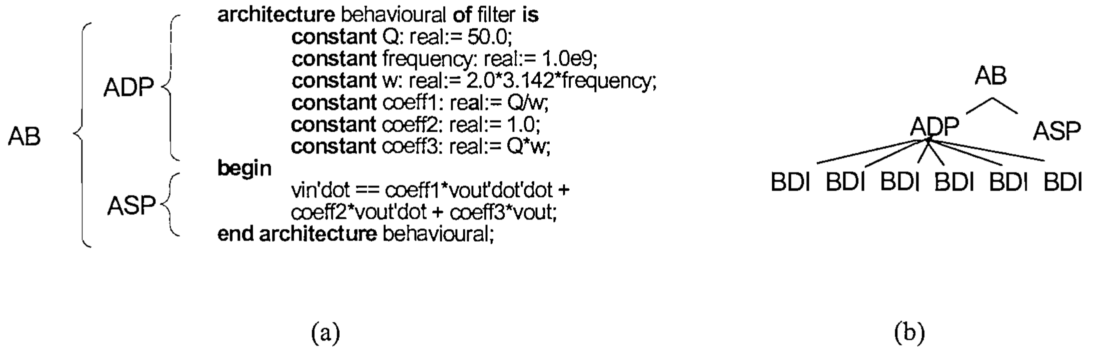
Table 3.3 Functions from class *ParseTreeTS*.

3.5.2 Static calculator

As mentioned before, the recursive static calculator is necessary to evaluate static expressions in all the expression clusters, including declarations that are used in the specification of filter coefficients. The static calculator performs multiple scans of the parse tree in a recursive manner,

and is used to extract filter information from a DAE or LTF filter construct. The filter information is used to obtain analogue filter cells that match the VHDL-AMS behavioural description.

3.5.2.1 Static calculations for Case Study 1



(a) (b)

Figure 3.12 (a) VHDL-AMS code of Case Study 1 (b) Parse tree of the AB.

As previously stated, after scanning the ASP cluster for synthesisability, and finding a synthesisable construct, the next step is to match the identifiers of declared objects, found at the bottom of the expression hierarchy, with their assigned values. This is done by analysing the ADP cluster. In other words, the values of coeff1, coeff2 and coeff3 are found by solving the respective static equations in the ADP. Using the ADP cluster for Case Study 1 for example, the six constant declarations in Figure 3.12 (a) are represented as six clusters of BDIs in the ADP cluster of the DAE construct as depicted in Figure 3.12 (b). The parse tree clusters for the six BDIs are shown in Figure 3.13.

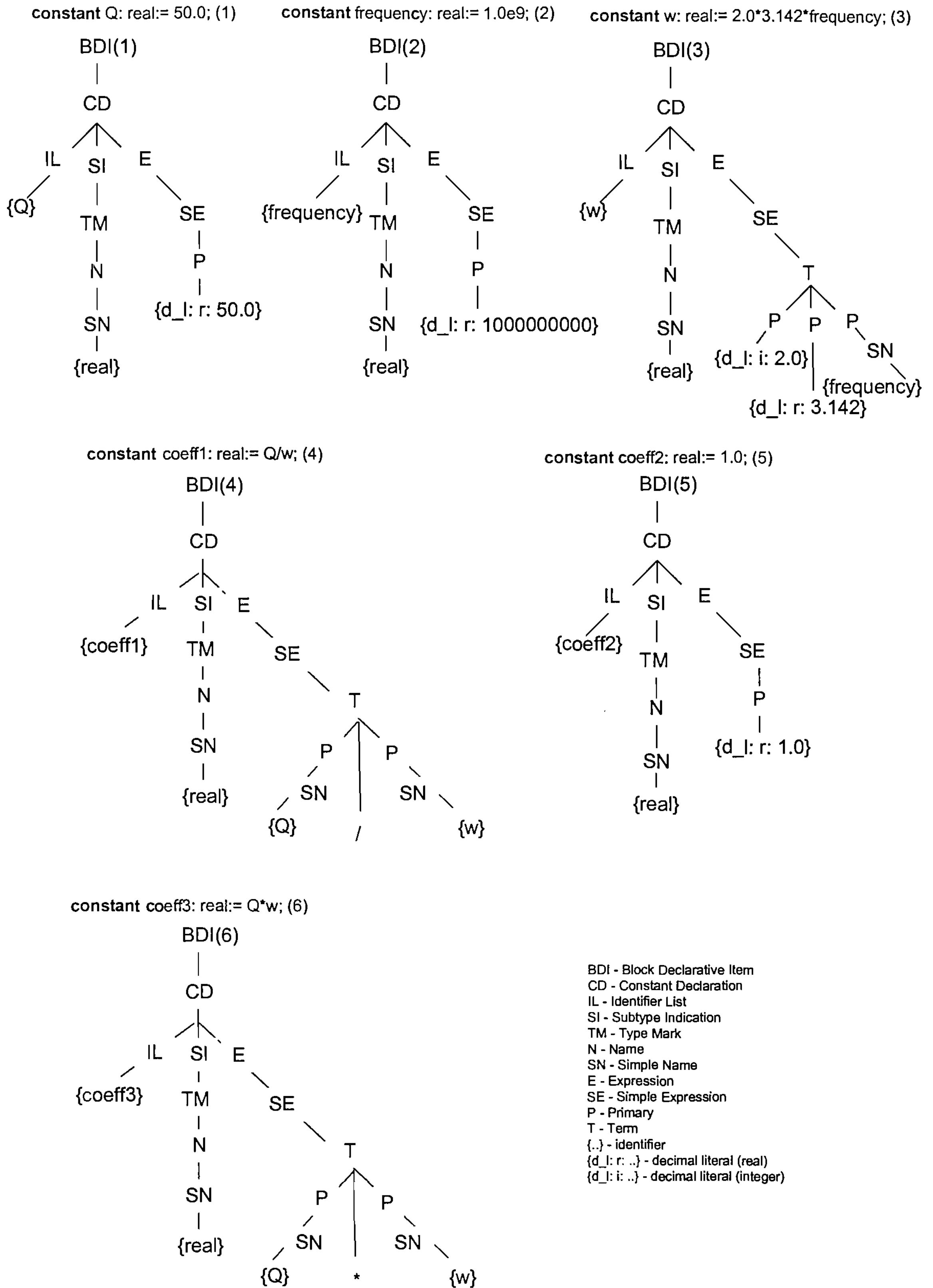


Figure 3.13 The six tree clusters of the BDI for Case Study 1.

Figure 3.14 shows the flow chart of the static calculator that traverses a DAE parse tree construct, to get the coefficient values whose names were previously found when analysing the ASP during the synthesis syntax check. The operation starts with a scan of the AB cluster with the aim to obtain the values of declared objects, which occur in expression leaves. At the end of the scan and calculating operation, the calculator stores the object identifiers and corresponding values in two arrays, named Name and Value, respectively. As constant declarations in both case studies contain expressions, the static calculator needs to scan these expressions before the constant values are known and can be stored. This is a recursive process in which store and scan operations are interleaved. The results calculated by the static calculator will be used to find suitable filter topologies for mapping into hardware. This will be explained further in the next chapter.

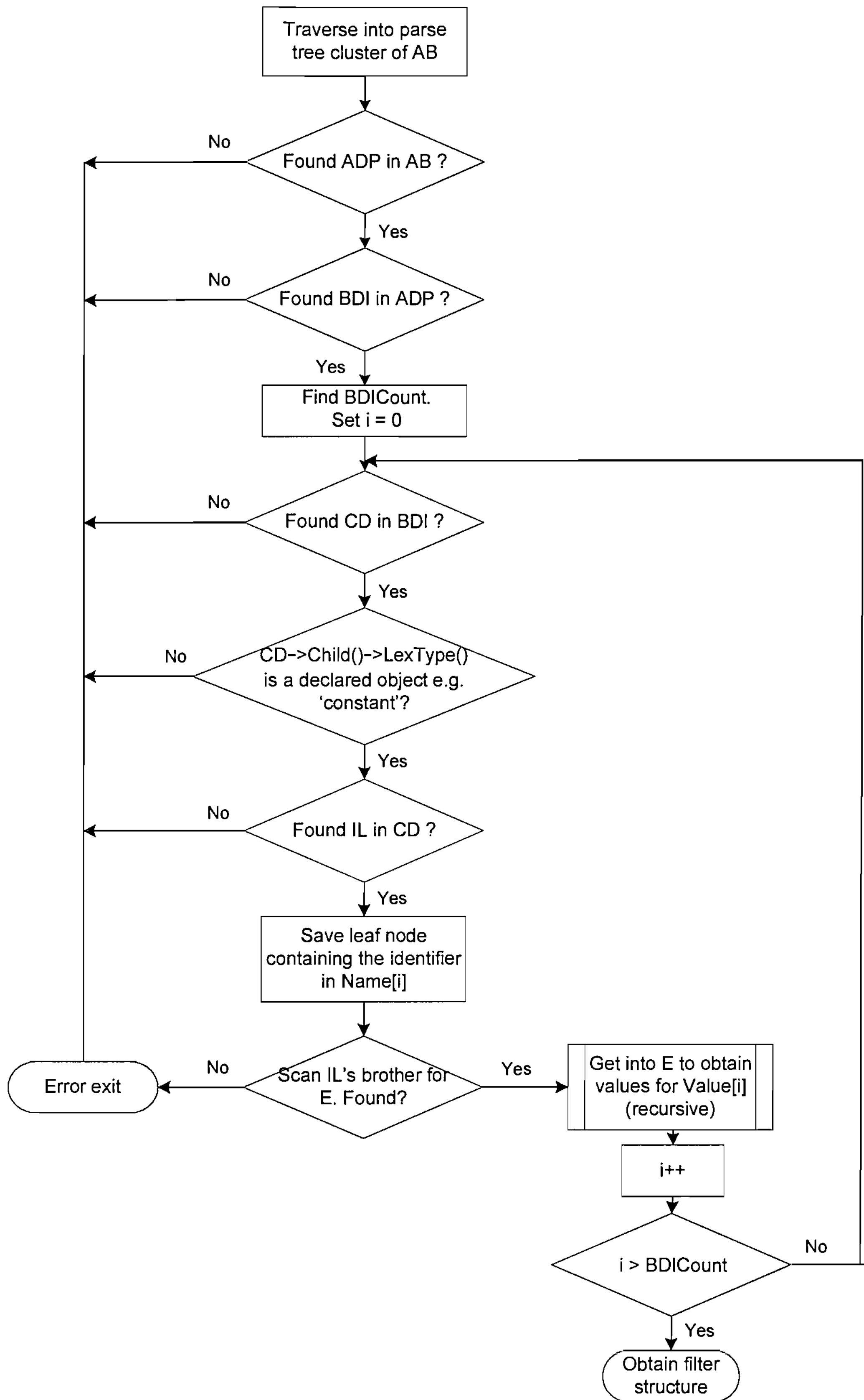


Figure 3.14 Flow chart of the static calculator program for a DAE parse tree construct.

3.5.2.2 Static calculations for Case Study 2

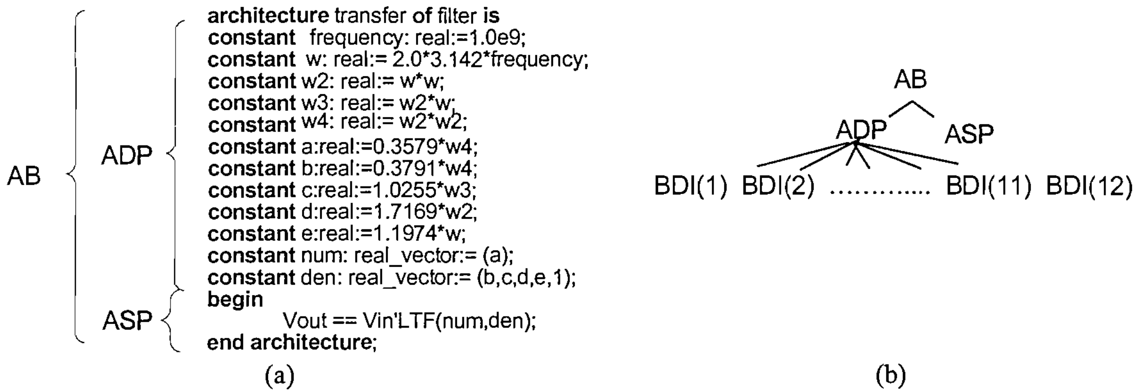


Figure 3.15 (a) VHDL-AMS code of Case Study 2 (b) Parse tree of the AB.

When an LTF construct is found, the static calculator performs its calculations in two stages. The first stage performs the same functionality as for a DAE construct, which is to solve the static expressions in the ADP, and is applicable for all the object declarations except for those of the numerator and denominator vectors. In the second stage, the static calculator arranges and stores the numerator and denominator coefficients in arrays named NumArray and DenArray for further analysis by the synthesiser.

The operation of the static calculator for Case Study 2 can be explained by an analysis of the AB tree cluster shown in

Figure 3.15 (a). The ADP cluster contains 12 constant declarations, which are all arranged into 12 BDI clusters used by the LTF construct

Figure 3.15 (b) in the ASP cluster. In the first stage the calculator evaluates the values for all the ten constants by recursive scans of the BDI clusters. In the second stage the static calculator collects the numerator and denominator coefficients for the transfer function. To do this, it traverses the BDI clusters that correspond to the numerator and denominator declaration. In this example, the numerator is represented by the 11th BDI, while the denominator is represented by the 12th BDI. Figure 3.16 shows the 12th BDI tree cluster which contains the coefficients for the denominator. The calculated coefficients are stored in the corresponding elements of arrays NumArray and DenArray. The position of each coefficient in the corresponding array is found by an analysis of the five subclusters (which are the EA clusters) of the aggregate clusters Ag in the declarations of real vectors num and den.

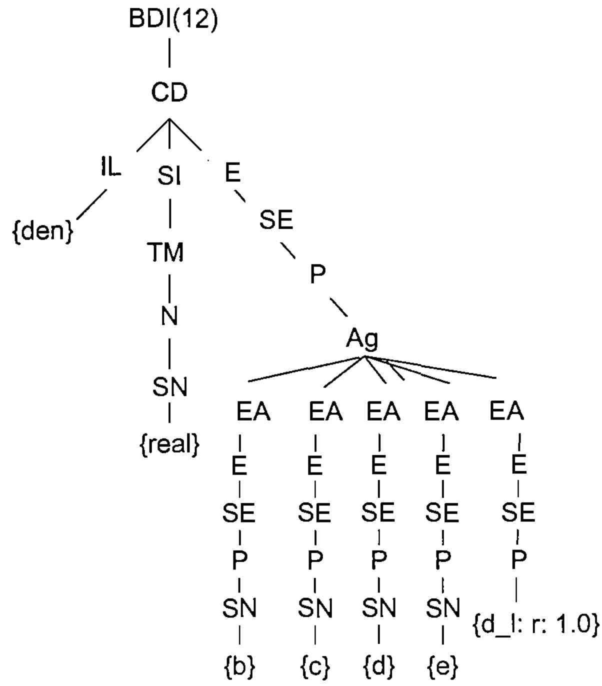


Figure 3.16 Tree cluster of the 12th BDI of the LTF construct of Case Study 2.

Figure 3.17 and Figure 3.18 show the flow charts of both static calculations. When the static calculator completes its analysis, which identifies the numerator and denominator coefficients, the next stage is similar to that for a DAE construct, which is to find suitable filter topologies. This further stage of the synthesis process will be explained in Chapter 4.

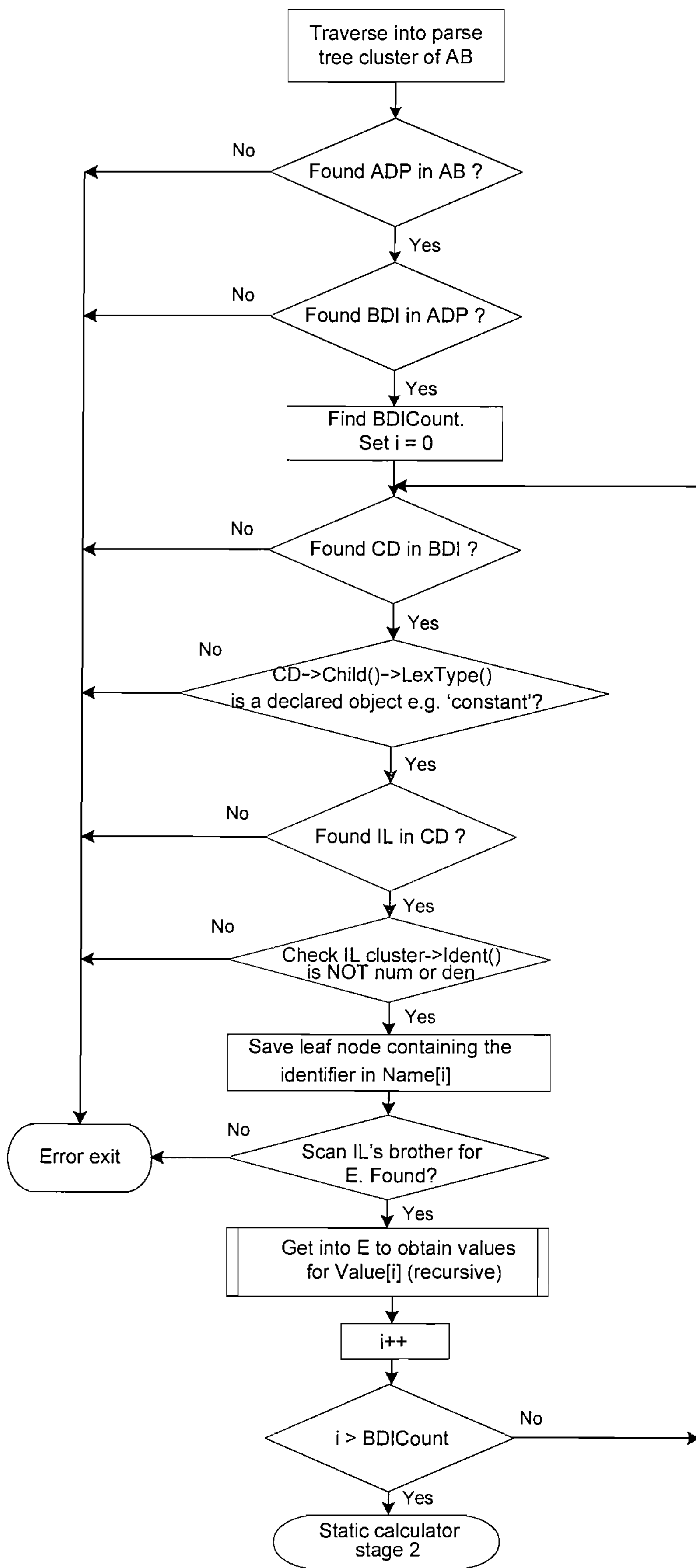


Figure 3.17 Flow chart of the first stage of the static calculator program for an LTF parse tree construct.

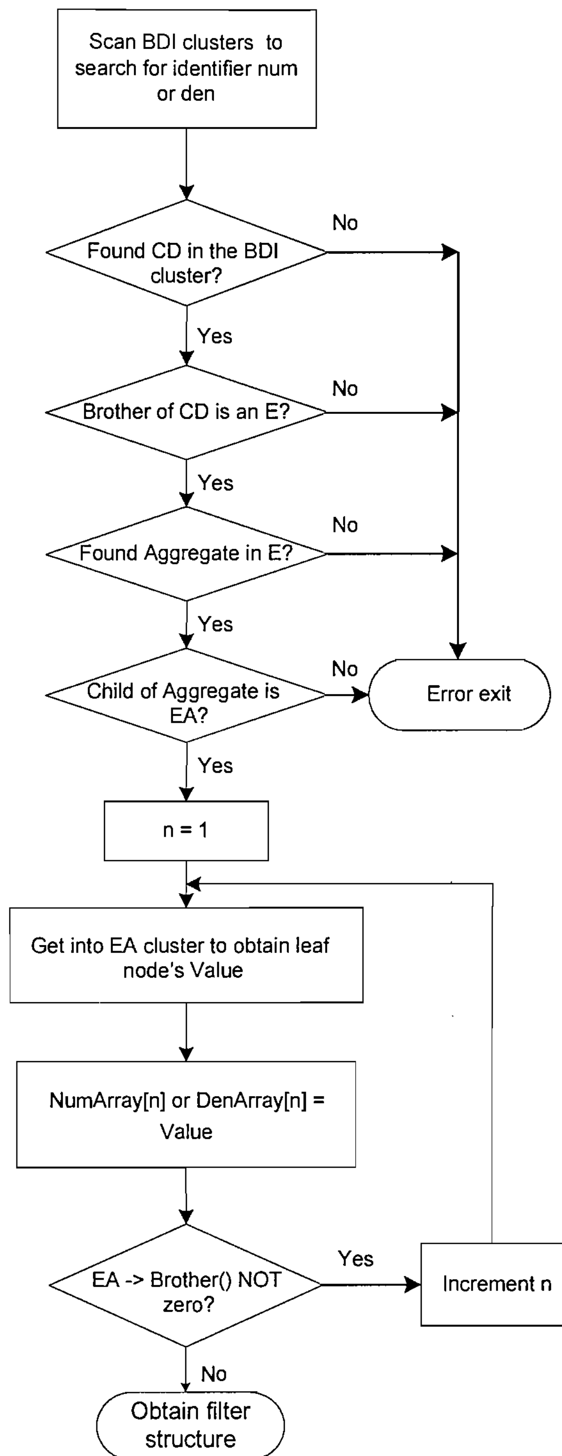


Figure 3.18 Flow chart of the second stage of the static calculator program for an LTF parse tree construct.

3.5.2.3 Software implementation

The static calculator module is implemented in the class named *Calculator*. The main functionality of *Calculator* is to traverse the parse tree to solve the equations declared in the VHDL-AMS source code. This calculator resolves simple arithmetic operations of filter parameters' relationship. The important methods implemented in *Calculator* is defined in Table 3.4.

Function Name and Syntax	Return Value	Description
StatusT CalculatorDAE() (void)	ERROR or OK status	Finding the values of the coefficients in a DAE of a filter description. This method implements the algorithm for the static calculator's operation on a DAE construct, as explained in Section 3.5.2.1 and shown in Figure 3.14.
StatusT CalculatorLTF() (void)	ERROR or OK status	Finding the values of the coefficients of the Laplace transfer function. This method implements the algorithm for the static calculator's operation on an LTF construct, as explained in Section 3.5.2.2 and shown in Figure 3.17 and Figure 3.18.
Double SimpleExpression (PTNodePT SENode)	Solution of the arithmetic operations of the Terms in SENode, or the value in the leaf of SENode.	Obtaining the value of a simple expression (SE). It scans into the Term clusters in an SENode to solve an arithmetic equation, or to obtain the value of its leaf node in the case of an SE consisting of a Primary without a Term. An SENode consisting of arithmetic equations must contain a Term. The Term will have two or more Primary clusters. The algorithm implementing this method is shown in Figure 3.19. The methods being used by *SimpleExpression()* are shown in the diagram as well.
double Sign(PTNodePT SENode)	1.0 or –1.0	This checks whether a Term is a positive or a negative value. The sign is determined by checking the value of *LexType()* of SENode.
double Term(PTNodePT node)	Solution of arithmetic operation.	This performs arithmetic operation between two Primary values. The algorithm is very similar to that of *SimpleExpression()*, the only difference that it does not contain the loop as *Term()* is specific for solving an arithmetic operation having two operands only.
double Factor (PTNodePT PrimaryCluster)	multiply: 1.0, divide: 2.0, open bracket: 3.0, close bracket: 4.0, semicolon: 5.0, plus: 6.0, minus: 7.0	This is to determine the arithmetic operation between operands. The method checks the *LexType()* of 'PrimaryCluster''s brother node.
Double Primary(PTNodePT PrimaryCluster)	Content of the leaf node in a Primary tree cluster.	This scans a Primary tree cluster to obtain the value in its leaf node. The algorithm implementing this method is shown in Figure 3.20.
Double PrimaryAggregate (PTNodePT node)	Solution of the arithmetic equation of the Simple Expression.	This scans a Primary Aggregate tree cluster to obtain the value in its leaf node. A Primary Aggregate tree cluster contains several Terms that are enclosed within a bracket '( )', which implies an arithmetic equation that contains precedence. For example consider (3.5). For a correct calculation for *Result*, the operation in the brackets must be solved first.

		Result = A + (B/C) (3.5) A Primary Aggregate will contain an SE cluster, thus *SimpleExpression()* will be called.
int CalcPrimary (PTNodePT node)	The number of Primary clusters in a SE tree node.	This is to count the number of Primary clusters in a SE by counting the brother nodes of the first Primary cluster.

Table 3.4 Functions from class *Calculator*.

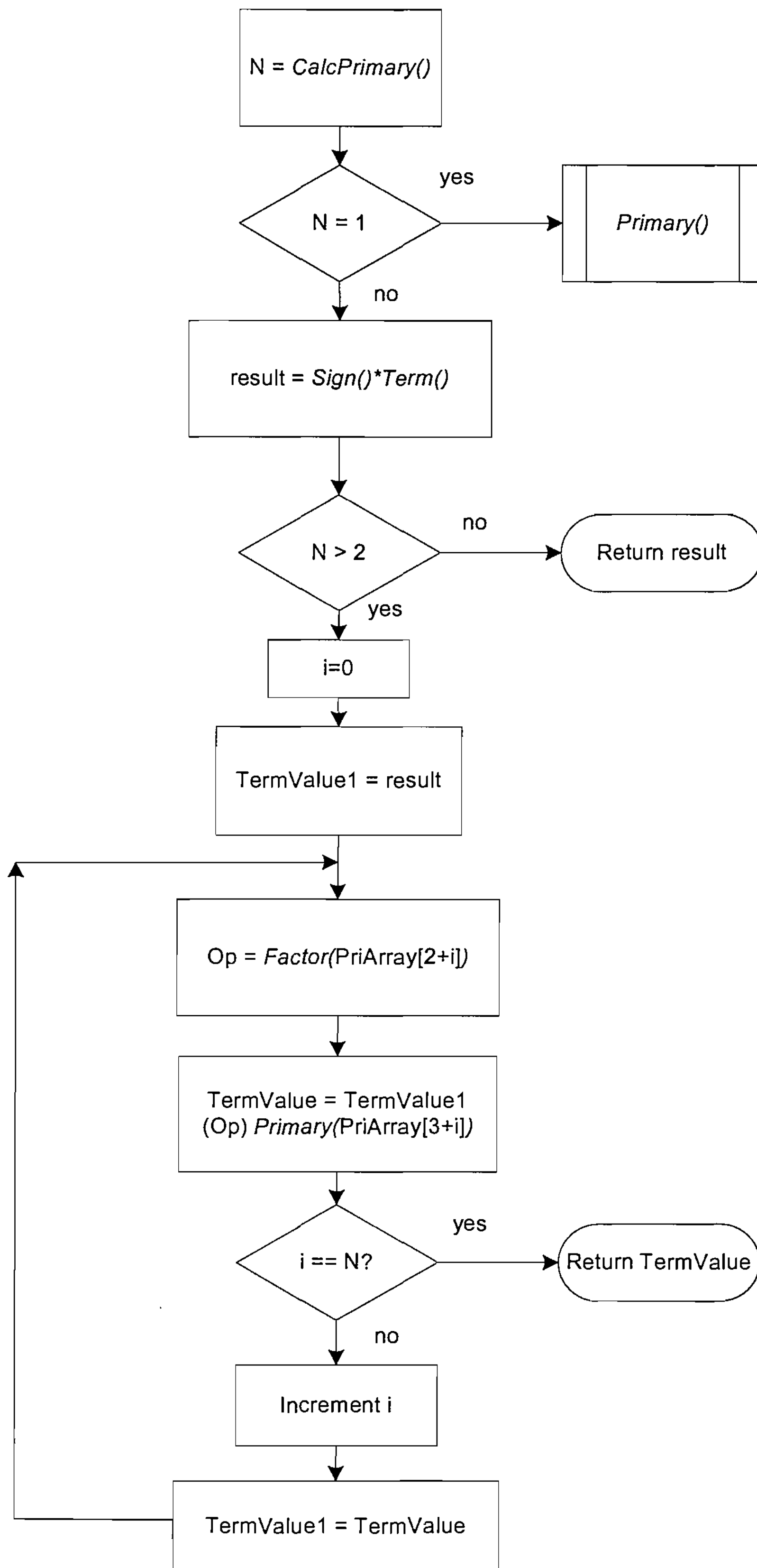


Figure 3.19 The algorithm implementing *SimpleExpression()*.

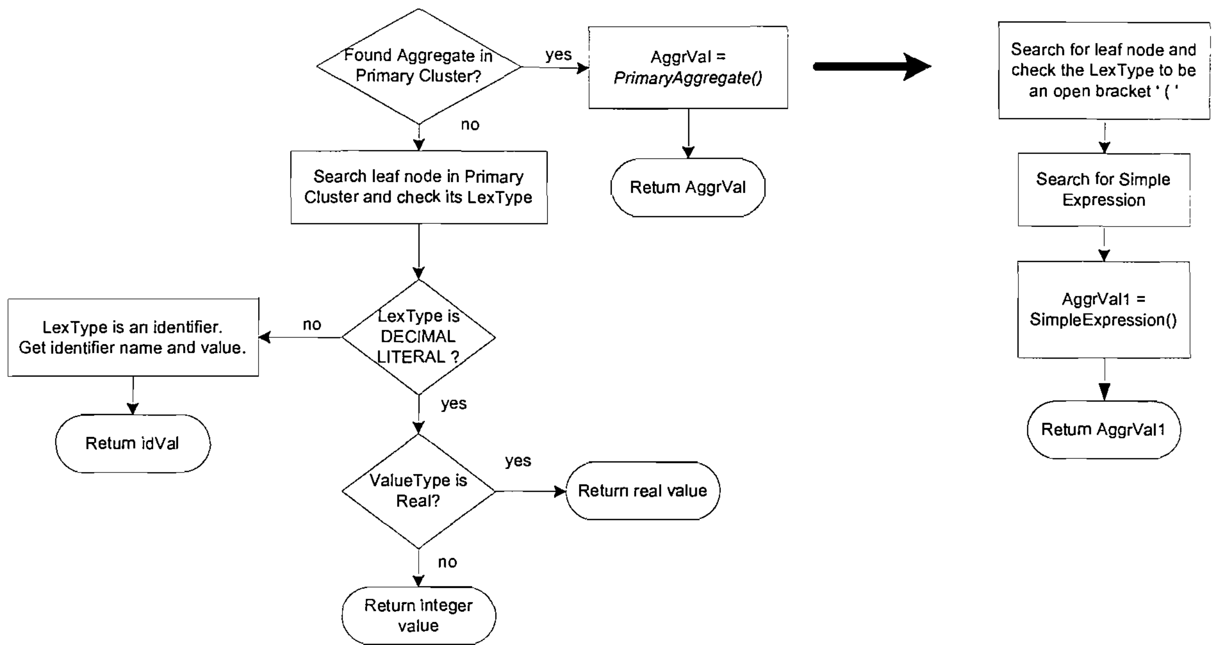


Figure 3.20 The algorithm implementing methods *Primary()* and *PrimaryAggregate()*.

3.6 Concluding remarks

This chapter firstly presents the two methods of describing behavioural analogue filter models in VHDL-AMS. The first method is to model a filter in the time-domain using differential-algebraic equations. The second method uses the VHDL-AMS LTF attribute to represent the filter's transfer function in the frequency domain. The behavioural synthesis process begins by analysing the parse tree structure of the VHDL-AMS description.

In Section 3.5, two software modules that extract filter information from the VHDL-AMS description of the analogue filter parse tree are explained. Both modules, the synthesis syntax checker and static calculator, are explained and illustrated with two case studies. The case studies respectively represent both the use of a DAE construct in the time-domain and an LTF construct in the frequency-domain. The synthesis syntax checker looks for patterns in the parse tree that represent either an LTF construct, or a valid DAE construct. Following the discovery of such patterns, the static calculator proceeds to find and evaluate the numerical values containing filter information that will be used by the subsequent stages of the synthesis process. The next chapter, Chapter 4, will explain the methodologies to attain analogue filter hardware structures from the filter information obtained from the methodologies described in this chapter.

The methodologies developed in this chapter are particularly useful in a hierarchical CAD environment for high-level synthesis. A designer may want to specify an abstract, behavioural filter model, as one of the components in a larger mixed-signal design. VHDL-AMS provides an integrated, standardised way to specify both digital and analogue systems on a variety of

abstraction levels. A filter specification may be written in terms of the gain, frequency or Q factor, or if necessary, at higher levels of abstraction, according to the design needs.

4 Behavioural Synthesis of Analogue Filters: from High-Level Filter Specification to Filter Structure

The previous chapter describes the VHDL-AMS behavioural modelling of analogue filter in both the time-domain and frequency-domain. Methods to extract analogue filter information from VHDL-AMS parse trees have also been explained. The next step in the synthesis process which proceeds from Chapter 3 is discussed in this chapter, where the techniques to obtain analogue filter structure from the extracted filter information is explained in detail. The filter structure, or topology, is taken from a filter cell library, which is a collection of various filter topologies suitable for ASIC applications. A very important phase in the synthesis process is the architectural optimisation stage, where filter topologies that can realise the required specification are firstly selected from the filter cell library. Then, each filter topology are parametrically optimised by adjusting and fine-tuning a set of circuit parameters, where finally the filter structure that has the best match to the specification is selected. The final output to the synthesis procedure is an optimised HSPICE netlist of the analogue filter topology.

There are two variations of the synthesis procedure that will be investigated earlier in this chapter, which are called Synthesis Procedure A and Synthesis Procedure B, which are described in Section 4.1. The main difference between both procedures lies in the processes preceding the architectural and parametric optimisation stage. The optimisation stage for both procedures is actually the same one. Synthesis Procedure A and Synthesis Procedure B are compared in Section 4.2 to choose the one that is most suitable for implementation.

Section 4.3 describes the architectural and parametric optimisation process in detail. The performance measure for analogue filters which defines the optimisation cost function is explained. In addition, the formulation and cvaluation of the cost function is shown and justified. However, the optimisation engine that implements the optimisation algorithm developed in the cause of this research, namely the three-tier algorithm, is not included in this chapter and will be the topic of discussion in Chapter 5.

4.1 Synthesis procedures

Figure 4.1 and Figure 4.2 show the synthesis flow in which the input is the filter information derived from VHDL-AMS parse tree and the output is the HSPICE netlist of an optimised filter topology. In Synthesis Procedure A of Figure 4.1, there are two types of filter information that results from a DAE or LTF construct, where each type needs to be handled in a different manner during the early part of the synthesis stage. Hence, the key synthesis processes are *root finding* followed by the *cell-mapping* process for the filter information derived from an LTF construct. Then, the process of *cell realisability check* receives either type of filter information (i.e. DAE or LTF) and if successful, the procedure proceeds to the following processes: *performance model construction*, *filter topologies selection* and finally, *architectural and parametric optimisation*. It is also shown in Figure 4.1 that for an LTF construct, there are two 'routes' to proceed from the *cell-mapping* process to *topologies selection* process. The differences between both 'routes' will be explained in Section 4.1.3

Figure 4.2 shows Synthesis Procedure B. This synthesis procedure differs from the other one in that it has fewer processes as it takes the filter information directly to produce the performance model. The processes involved in this synthesis procedure are *performance model construction*, *cell realisability check*, *filter topologies selection* and *architectural and parametric optimisation*.

The conceptual differences between both procedures will be explained in Section 4.2 after the related processes for both synthesis procedures are explained in subsections 4.1.1 to 4.1.5. The role for each process in Synthesis Procedures A and B is briefly explained as follows:

Root finding: A state-of-the-art polynomial root finding algorithm [109] has been implemented to calculate the poles and zeros of any VHDL-AMS transfer function written as an LTF construct.

Filter cell mapping: The main aim of this mapping stage is to identify the first- or second-order sections obtained in the root finding process, and to break down a high-order transfer function into first-order and/or second-order ones.

Cell realisability check: The aim of this stage is to extract the filter parameters from the DAE or LTF construct. As the synthesis methodology developed here is primarily intended for high-frequency applications, this stage determines whether the specification can be realised, as this will be constrained by the topologies available in the filter cell library.

Construction of performance model: The VHDL-AMS specification is translated into a performance model, which is used as the target or benchmark for the optimisation process.

Topology selection: This is a manual process, where the user will pick a selection of analogue filter candidates from the filter cell library. If necessary, this process could be automated.

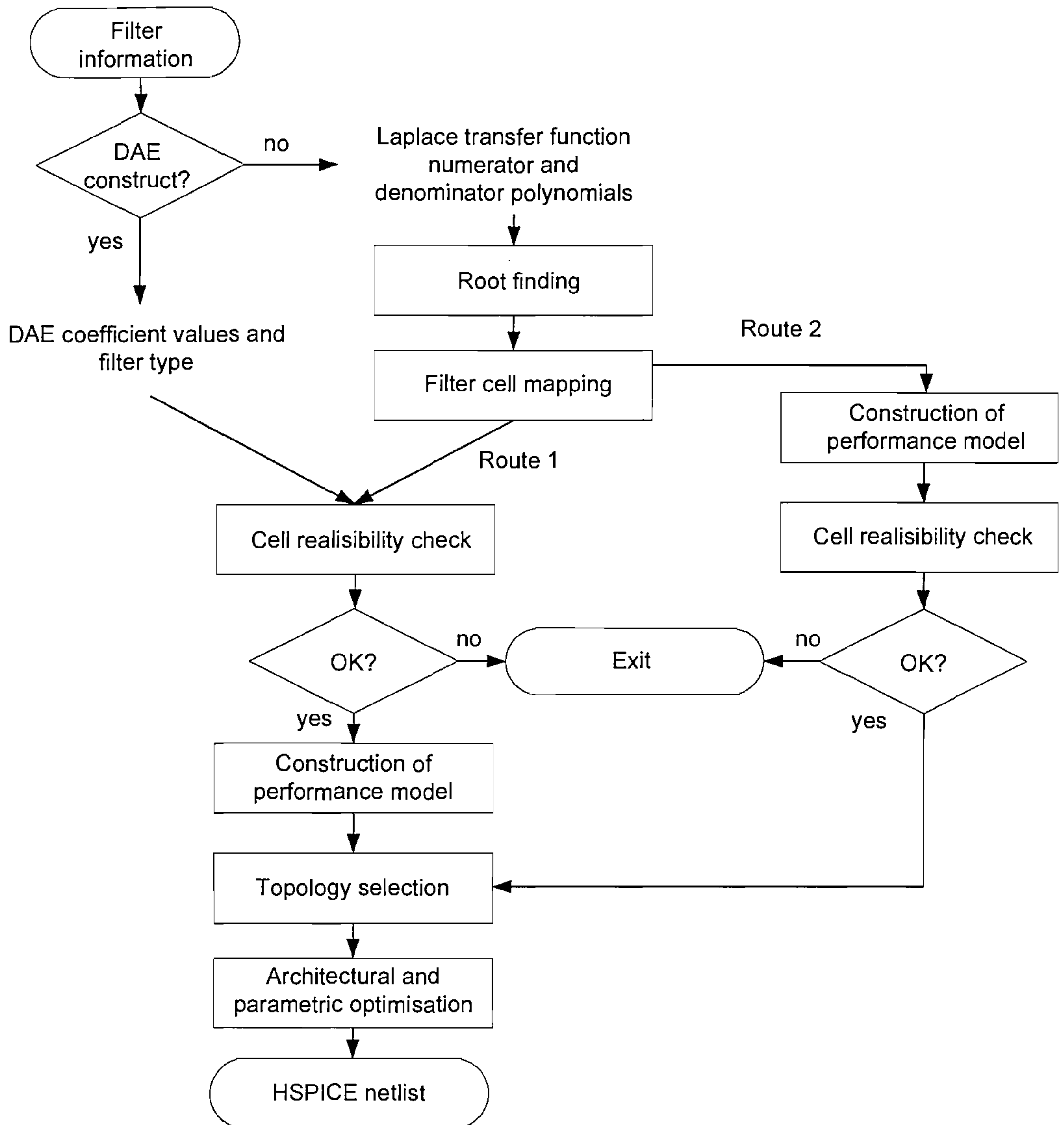


Figure 4.1 Synthesis Procedure A.

Architectural and parametrical optimisation: The candidate topologies selected by the user are each parametrically optimised using the three-tier algorithm developed as part of the synthesiser. The three-tier optimisation algorithm will be explained in Chapter 5. During optimisation, the evaluation of the performance of filter topologies is done by running an HSPICE AC analysis to compare the actual frequency response with the ideal response defined by the behavioural specification. Each filter cell in the library has a set of parameters that have been pre-selected for optimisation. The objective (or cost) function in the three-tier optimisation algorithm contains a weighted combination of the AC characteristic accuracy and power consumption. The optimised candidate topologies are then compared and the best one is selected. The optimisation methodology is independent of any technology and can be used for any type of filter whose frequency response can be obtained by full HSPICE simulation.

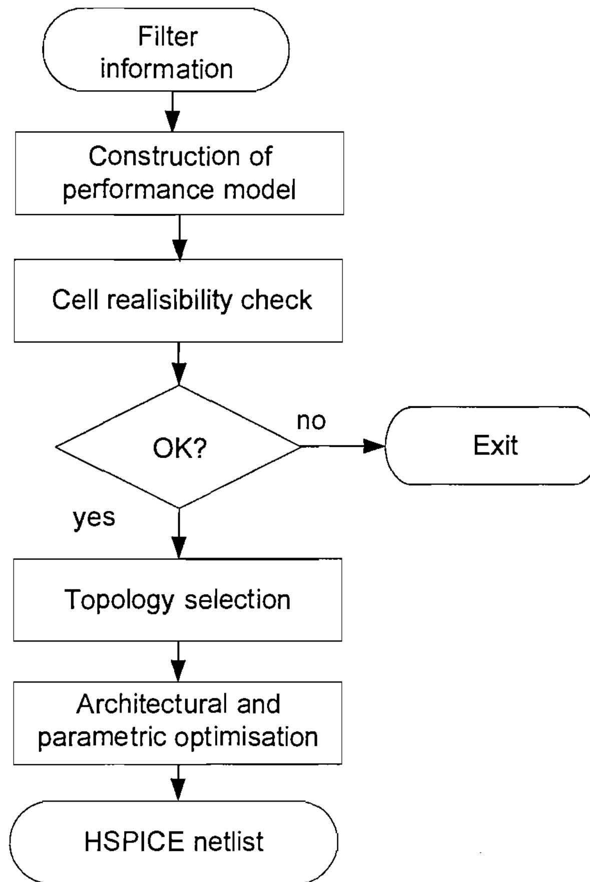


Figure 4.2 Synthesis Procedure B.

4.1.1 Root finding

The filter information derived from an LTF construct is in the form of polynomials for the numerator and denominator of the transfer function. Using classical analogue filter synthesis technique, synthesis proceeds by expressing the transfer function in terms of its poles and zeros, and to be able to do this, it is necessary to find the roots of the numerator and denominator polynomials. A standard, state-of-the art root-finding algorithm [109], that uses complex arithmetic, has been adapted for this purpose. The routine is based on Laguerre's method that guarantees convergence to a root from any starting point. In the implementation of the root-finder, the numerator and denominator coefficients are placed in a complex vector, and similarly the program will return the polynomial roots in another complex vector. After the polynomial roots have been found, the next stage in the synthesis procedure is the cell mapping process.

4.1.2 Filter cell mapping

The cell mapper identifies the pole and zero structure of the transfer function and translates high-order filter descriptions arising from LTF constructs to first and second-order (biquad) filter cells. This stage must be preceded by the root-finding process, as the mapping is based on the roots (the poles and zeros) of the filter transfer function. The cell mapper identifies four types of first-order and second-order filter structures as shown in Table 4.1, and will also map a high-order transfer function into first-order and second-order sections of the four types of filter cells.

Filter cell type	Transfer function H(s)
1st-order lowpass	$\frac{1}{s+a}$
2nd-order type 1 (bandpass)	$\frac{s+a}{s^2+bs+c}$
2nd-order type 2 (lowpass)	$\frac{1}{s^2+bs+c}$
Highpass	$s+a$

Table 4.1 First- and second-order constructs used by the cell mapper.

Although the functionality of the cell mapper can be expanded to include the automatic identification of other type of first- and second-order filter cells, it is sufficient for the purpose of demonstration to use only the four types listed in Table 4.1. The implementation of the cell mapper is shown in the flow chart of Figure 4.4.

Two examples of filter cell mapping are shown in Figure 4.3. In Figure 4.3 (a), which has a root in the numerator and a pair of complex conjugate roots in the denominator, the description is mapped to a biquad filter cell of type 1. Figure 4.3 (b) shows a section of the VHDL-AMS description that indicates that the filter has no root in the numerator, and two pairs of complex conjugate roots in the denominator. As shown in the flow chart in Figure 4.4, this structure will be mapped to two sections of biquad filter type 2.

High-order filters are commonly implemented as cascades of first-order and/or second-order sections. Therefore, to proceed with the synthesis of high-order filter such as that of Figure 4.3 (b), cascade design techniques [2], such as pole-zero pairing, section ordering and gain distribution must be applied, but such techniques have not been implemented in the synthesiser described here.

```
architecture transfer of filters is
      constant w:real:=6.283e9;
      constant q:real:=20.0;
      constant a:real:=w/q;
      constant b:real:=w*w;
      constant num:real_vector:=(0,a);
      constant den:real_vector:=(b,a,1.0);
begin
      vout == vin'LTF(num,den);
end architecture;
```

A 2nd order filter type 1:
(s-a)/(s*s + bs + c)

(a)

```
constant num:real_vector:=(a);
constant den:real_vector:=(b,c,d,e,1.0);
```

A 2nd order filter type 2:
1/(s*s + bs + c)

A 2nd order filter type 2:
1/(s*s + bs + c)

(b)

Figure 4.3 Examples of filter mapping for higher filter order.

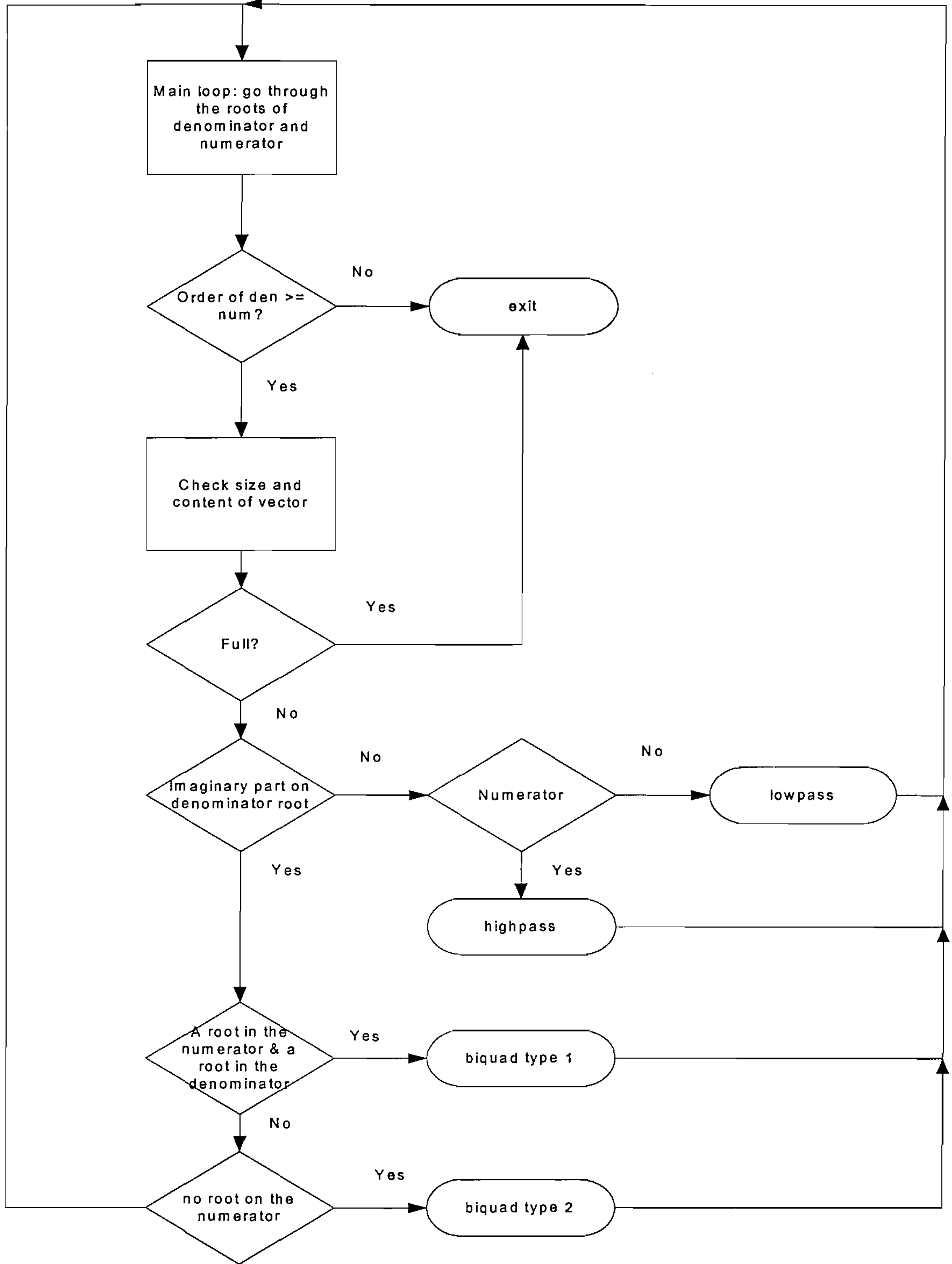


Figure 4.4 Flow chart of the filter cell mapper algorithm.

4.1.3 Cell realisability check

This stage is where filter information from both the DAE and LTF construct is received and evaluated. As mentioned, the main purpose of this stage is to determine whether the synthesis procedure can proceed to the next stage, by checking the filter information against the available topologies in the analogue filter cell library. The topologies in the cell library are characterised by the filter type (lowpass, bandpass etc) and filter order. The operational transconductance amplifier (OTA) cells which are the main building blocks of the analogue filer topologies have unity gain frequency in the range of several GHz. Hence the maximum frequency of operation for any analogue filter under synthesis is limited to 1 GHz. Following this, the main specifications that need to be derived from the DAE or LTF filter information is the filter type, order and frequency of interest.

For certain type of analogue filter specification, such as the second-order DAE constructs such as those described in Section 3.3.1, and second-order lowpass, bandpass and highpass LTF filter construct written as in Equation 4.1, cell parameters such as the gain, cut-off frequency and Q factor are derived to determine realisability.

$$H(s) = \frac{k_1 s^2 + k_2(\omega_0 / Q)s + k_3 \omega_0^2}{s^2 + (\omega_0 / Q)s + \omega_0^2} = \frac{N(s)}{D(s)} \tag{4.1}$$

where *k1*, *k2* and *k3* are constants, ω_0 is the natural frequency, and Q is the Q factor.

For other types of LTF construct not being written in the form of Equation 4.1, the filter order is known by inspecting the order of the denominator of the transfer function, while other specifications such as the filter response type and frequency of interest is known by constructing a magnitude frequency response curve of the filter. Having said this, the method of constructing the frequency response together with knowing the filter order can be used for any type of an LTF filter construct, including the one having the form of Equation 4.1. Referring to the diagram showing Synthesis Procedure A in Figure 4.1, the generalised LTF construct follows Route B, where cell realisibility check is done after the performance model is constructed. Route A is taken if the LTF construct is in the form of Equation 4.1. For Synthesis Procedure B of Figure 4.2, the performance model is constructed before the cell realisibility check is carried out, hence assuming the filter information in a generalised LTF form.

Figure 4.5 shows the realisability check for a DAE construct written in the form introduced in Section 3.3.1. Whereas the flow chart in Figure 4.6 illustrates a similar method for such LTF constructs.

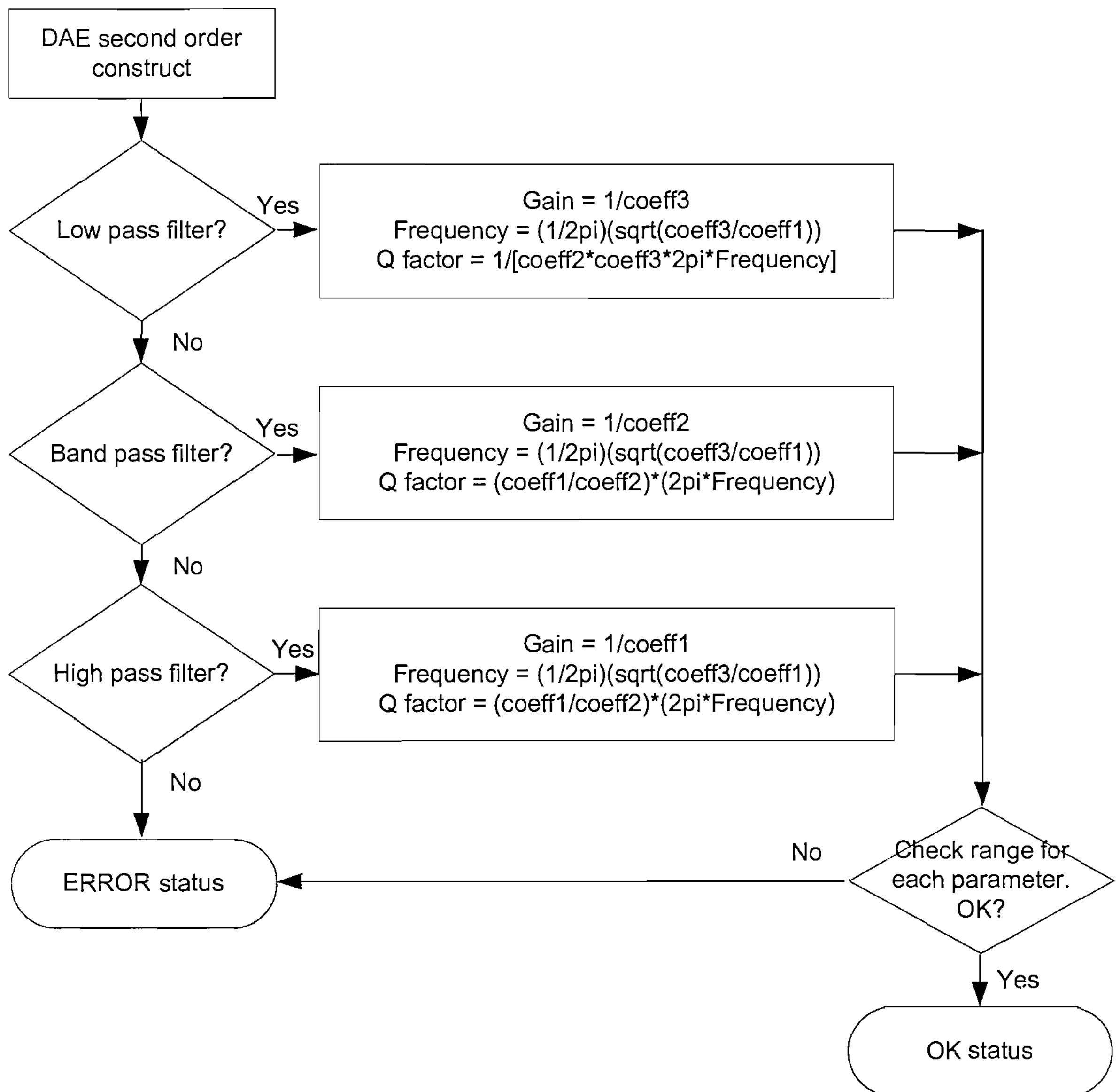


Figure 4.5 Filter parameters calculation and checking for a second-order DAE construct.

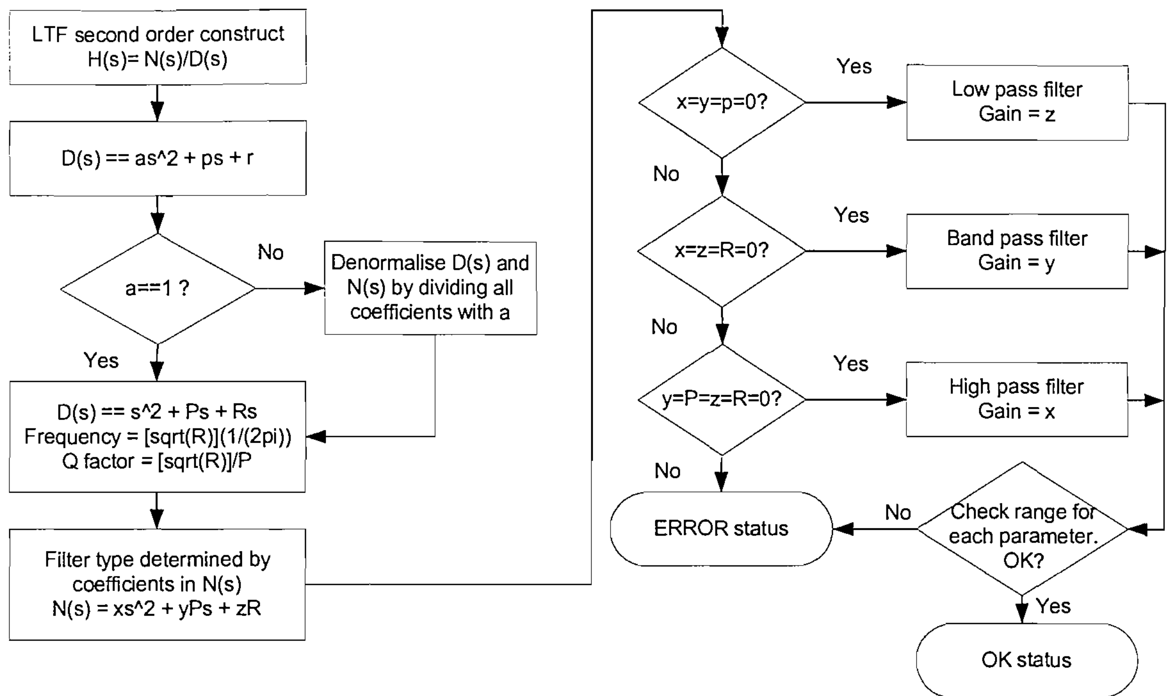


Figure 4.6 Filter parameters calculation and checking for a second-order LTF construct.

4.1.4 Construction of performance model

The aim of this step is to derive magnitude over frequency data for the analogue filter from the existing information. It is used in the optimisation stage, as it becomes the ideal or target performance to be achieved by the filter. The performance model can be represented in two forms:

- ***Measurement targets***, where the targets are used to inspect the output of the AC simulation results of HSPICE, and to calculate certain criteria that defines the specification, for example the Q factor of a bandpass response.
- ***Construction of ideal curve***, which is the filter's frequency response in terms of the magnitude versus frequency.

The information obtained from either the DAE or LTF construct is sufficient to build the performance model for optimisation. This is especially true for an LTF construct, where the ideal curve of the frequency response is derived directly from the transfer function. A set of frequency points that span over the range of the frequency of interest is selected, and the magnitude at each point is calculated.

4.1.5 Topology selection

In the present implementation of the synthesiser, the topology selection stage is done manually by the user, where the user selects up to 10 filter cells from a list suitable for the specification. As

mentioned before, the user is presented with the filter type and order. The user inputs the selected topologies to the cell optimisation stage. This is shown in Figure 4.7.

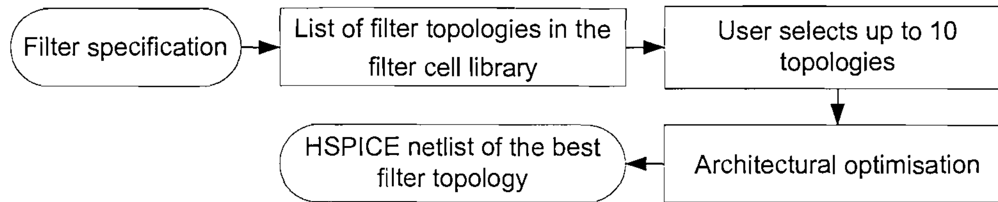


Figure 4.7 The process of selecting filters topologies for optimisation.

In this research, an analogue filter library has been developed for integrated second-order and fourth-order lowpass and bandpass filters for frequencies of up to 1 GHz. The topologies of the filters and its descriptions are listed in the file 'TopoList.txt', which the user will be asked to inspect. The topologies that the user selects must be saved in the file 'SelectedTopo.txt'. The program that does the architectural optimisation process will parametrically optimise each user-selected topology, and will finally select the best topology. The topic of architectural and parametric optimisation is discussed later in this chapter in Section 4.3.

4.2 Comparative study between synthesis procedures

Synthesis Procedure A and B are used to synthesise a fourth-order lowpass Butterworth filter that has a cut-off frequency at 1 GHz. The VHDL-AMS specification of the filter is given as follows:

```
entity filter is
      port (quantity Vin: real;
            quantity Vout: out real);
end entity filter;

architecture transfer of filter is
      constant a:real:=4.1589e-10;
      constant b:real:=8.6483e-20;
      constant c:real:=1.0535e-29;
      constant d:real:=6.4162e-40;
      constant num: real_vector:= (1);
      constant den: real_vector:= (1,a,b,c,d);
begin
      Vout == Vin'LTF(num,den);
end architecture;
```

where the original transfer function of Equation 4.2, taken from filter tables [108], is denormalised to 1 GHz.

$$H(s) = \frac{1}{1 + 2.6131s + 3.4142s^2 + 2.6131s^3 + s^4} \tag{4.2}$$

Following Synthesis Procedure A, this LTF construct is then taken through the root finding and then filter cell mapping process. The root finding process evaluate the LTF construct into two sets of pole-zero pairs in the form of complex conjugates (Equation 4.3), which are mapped into two second-order lowpass filter sections, $H_1(s)$ and $H_2(s)$ of Equations 4.4 and 4.5. The required fourth-order filter can then be obtained by cascading these second-order section of $H_1(s)$ and $H_2(s)$. The cascading order is $H_1(s)$ followed by $H_2(s)$ for the topology called Cascade H_1H_2, and $H_2(s)$ followed by $H_1(s)$ for topology Cascade H_2H_1.

$$s_1 = \alpha_1 \pm j\beta_1, s_2 = \alpha_2 \pm j\beta_2 \tag{4.3}$$

$$H_1(s) = \frac{\omega_0^{\ 2}}{s^2 + 2\alpha_1 s + (\alpha_1^{\ 2} + \beta_1^{\ 2})} \tag{4.4}$$

$$H_2(s) = \frac{\omega_0^{\ 2}}{s^2 + 2\alpha_2 s + (\alpha_2^{\ 2} + \beta_2^{\ 2})} \tag{4.5}$$

where $\omega_0 = 2.\pi.10^9$, $\alpha_1 = 2.404e10^9$, $\beta_1 = 5.805e10^9$, $\alpha_2 = 5.805e10^9$, and $\beta_2 = 2.404e10^9$.

After obtaining the transfer function for each second-order section, following Route B of Figure 4.1, the performance model in the form of an ideal curve is generated for each section. The topology selected to realise the second-order lowpass specification is an OTA-C filter, where the OTA cell is implemented using the wide-swing configuration. Then, both second-order sections are individually optimised using the three-tier parametric optimisation algorithm, where the performance models for each section is shown in Figure 4.8 and Figure 4.9. The details of the analogue filter topology will be explained in Chapter 6, while the detail on the three-tier algorithm is in Chapter 5.

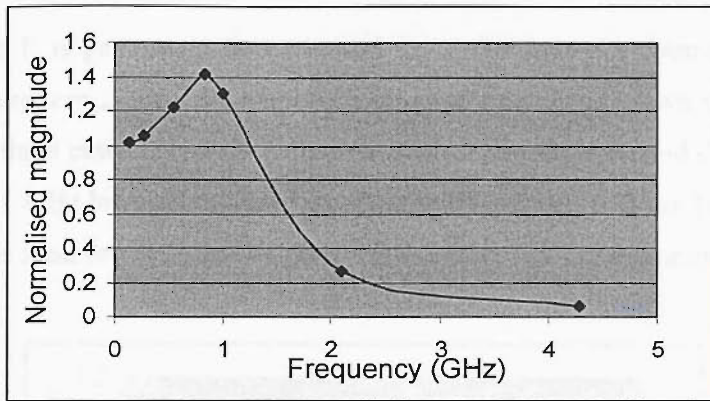


Figure 4.8 The performance model implementing the transfer function $H_1(s)$.

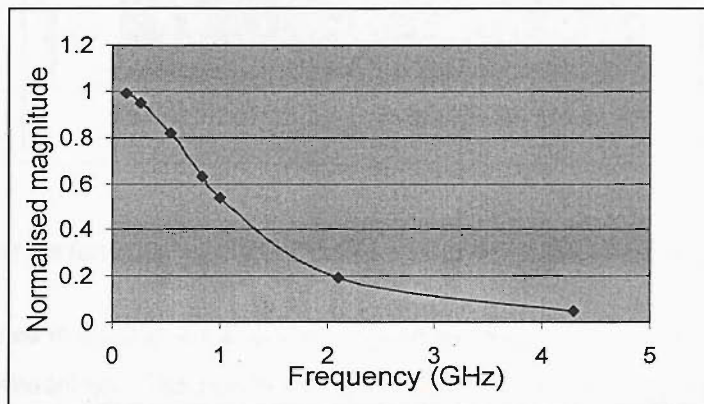


Figure 4.9 The performance model implementing the transfer function $H_2(s)$.

The three-tier optimisation returns an error figure, called err_{total}, which indicates how close the actual performance matches the ideal response. The lower the error figure, the better the result. After each second-order section is optimised, both sections are connected in cascade and the error figure for the resulting fourth-order lowpass filter is calculated. The two possible ways of combining the second-order filter sections into a fourth-order filter, as mentioned earlier, gives the following result in Table 4.2.

Topology	Error figure
Cascade H_1H_2	0.31461
Cascade H_2H_1	0.30603

Table 4.2 Error figures for the cascades of second-order sections of H_1 and H_2.

On the other hand, following Synthesis Procedure B, the VHDL-AMS filter description of the fourth-order lowpass filter implementing the denormalised transfer function of Equation 4.2 is taken to produce an ideal curve, as shown in Figure 4.10. Then, the topology of a fourth-order lowpass OTA-C filter using the wide-swing configuration is selected for implementation. The

topology, Cascade 1, is parametrically optimised using the three-tier optimisation, which finally gives an error figure, err_{total}, of 0.30260, which is lower than those shown in Table 4.2. The AC waveforms of the three cascade circuits - Cascade H_1H_2, Cascade H_2H_1 and Cascade1 - against the ideal fourth-order 1 GHz lowpass response are shown in Figure 4.11. It can be seen that Cascade 1 closely matches the ideal response and outperforms the other two configurations.

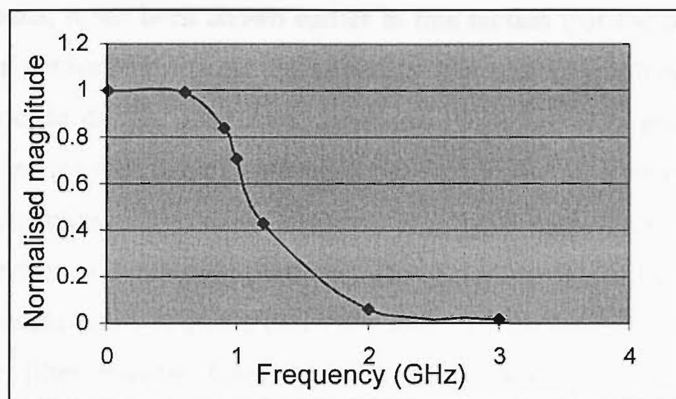


Figure 4.10 The performance model for the fourth-order 1 GHz lowpass Butterworth filter.

As mentioned in Section 4.1.2, cascade design techniques are used in the classical analogue filter synthesis methodology. The results of this synthesis example using Synthesis Procedure A shown in Table 4.2 and Figure 4.11 confirms one aspect of this technique, where the second-order sections should be arranged so that the section with the flattest magnitude of frequency, i.e. having the lowest Q factor should comes first. Using the information of the roots of the transfer function of Equation 4.3, the values of the Q factor can be derived using the following relationship between α and β, with ω_0 and Q:

$$\alpha = \frac{\omega_0}{2Q} \text{ and } \beta = \omega_0\sqrt{1-\frac{1}{4Q^2}} \qquad (4.6 \text{ (a) and (b)})$$

which gives Q_1=1.307 and Q_2=0.541. Thus, the synthesis results in this example confirms to the anticipated outcome of using this technique, that the topology Cascade H_2H_1 is better than Cascade H_1H_2.

From the presented results and the discussions in this section, it is obvious that Synthesis Procedure B produces <u>better</u> result using <u>less</u> computational effort than Synthesis Procedure A. In Synthesis Procedure A, the extra steps of finding the roots of the transfer function, mapping filter cells and individually optimising the second-order sections are not worth the effort as the results produced in such carefully defined steps are still inferior. In contrast, in Synthesis Procedure B

most of the computational burden is given to the parametric optimisation algorithm, which is a fully-automated process. Also, as will be seen in Case Study 3 in Chapter 6, there exist other type of topologies other than cascade circuits that can realise the required specification, thus, it is not necessary to limit the synthesis procedure to tailor towards one type of implementation only.

Synthesis Procedure B is more general, and although it might be argued that the root finding and cell mapping procedure can be used after the topology selection stage specifically for cascade type of circuits, it has been shown earlier in this section that the parametric optimisation process is a powerful yet simpler process that is able to find a better result rather than a specialised methodology for cascade design. Therefore, Synthesis Procedure B is implemented in the final version of the work and the root finding method (Synthesis Procedure A) was abandoned.

The implementation of Synthesis Procedure B in FIST recognises only LTF constructs in VHDL-AMS code, not time-domain descriptions. The reason for this is because, it is very rare to find time-domain specifications of analogue filters, and the numerous filter specification found in filter tables are for filter transfer functions. Also, any linear and time-invariant differential algebraic equations can be easily transformed to a transfer function using Laplace transformation method.

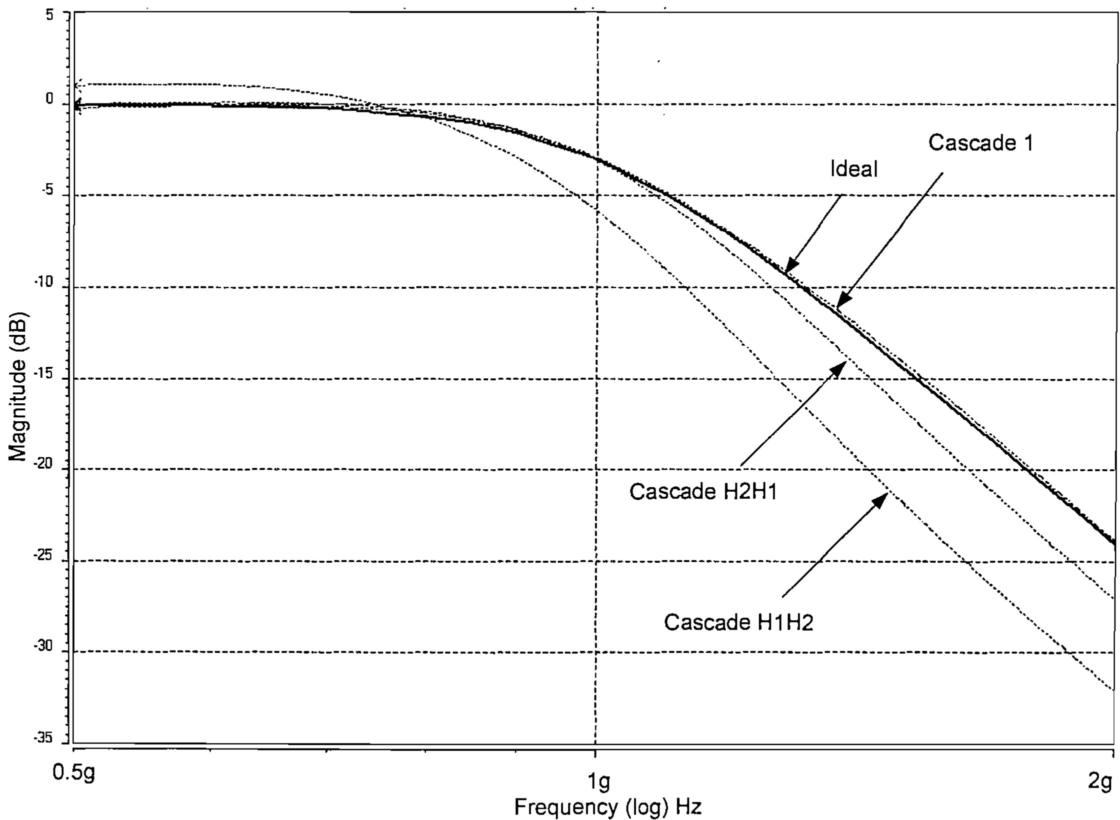


Figure 4.11 HSPICE AC results for the comparison between Synthesis Procedure A and Synthesis Procedure B.

4.3 Architectural and parametric optimisation

In the topology selection process of the synthesis methodology explained in the previous section, the user will select from the cell library, a set of filter circuits that can perform the specified functionality. During the last stage of the synthesis procedure, the optimisation algorithm will evaluate and optimise each candidate and will finally select the best circuit. This process is called architectural optimisation. This section outlines the architectural optimisation methodology, and other aspects concerning the parametric optimisation, which are the cost function formulation and evaluation.

4.3.1 Architectural optimisation

Architectural optimisation is performed by analysing various filter topologies, optimising them parametrically using the selected performance criteria and finally choosing the best one. There are three key aspects that must be addressed: 1) *cost function,* 2) *parametric optimisation engine*, and 3) *evaluation method.* In our case the cost function that models the required performance is based on a weighted combination of the frequency response accuracy and power consumption. The desired frequency response is computed from the filter specification as explained in Section 4.1.4. The optimisation engine is based on a three-tier optimiser that is described in detail in Chapter 5.

The evaluation of the cost function is performed for each candidate topology by running a full HSPICE AC analysis within each iteration of the optimisation loop. The results are then used by the optimisation engine to find a direction in the parameter space leading to a lower value of the cost function. This process is iterated for each topology until a termination condition is met, then another candidate is evaluated using the same method.

This algorithm is illustrated in Figure 4.12, where the performance model is the AC response data, which is used in the three-tier optimisation process. This data can either be passed directly to the parametric optimisation engine in the form of a data file, or can be embedded in the netlist of the analogue filter topologies. A set of user-selected filter topologies obtained from the filter library is passed to the architecture evaluator, which acts as a mediator or controller that gives the netlist of the filter topology to the optimisation engine to be parametrically optimised one after another. The three-tier optimiser returns the cost function value for each topology to the architecture evaluator which keeps track of this information. Finally, after each topology has been optimised, the architecture evaluator selects the topology with the lowest cost function. The pseudo code for the architecture evaluator is given in Figure 4.13.

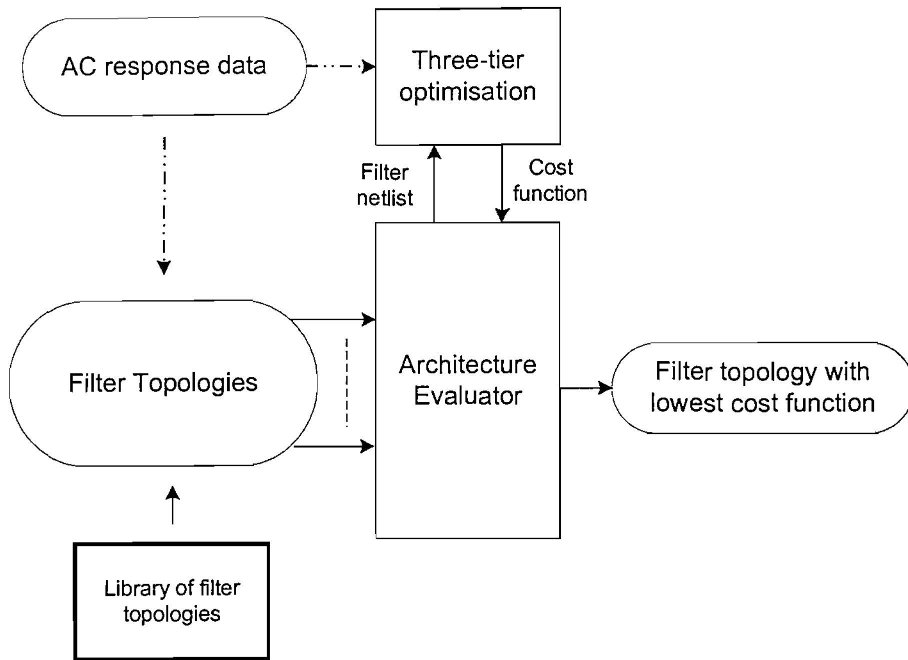


Figure 4.12. The process of architecture optimisation of analogue filter topologies.

Architecture evaluator

Input: Topology file netlist names.

1. For each topology:

Call three_tier_opt( )

Read the results file

Log the results that give the smallest cost function (CF)

2. Check the list of the best cost functions for each topology.

Output: the best topology and its CF.

Figure 4.13 Pseudo code for the architecture evaluator.

4.3.2 Parametric optimisation

In this section, Case Studies 1 and 2 that are introduced in Chapter 3 are used to illustrate practical details of the analogue filter synthesis process from behavioural specifications. As the specification is for high-frequency, and the implementation is intended for on-chip integration, the underlying

filter cell topologies are selected from suitable work reported in literature. This section demonstrates how the filter information extracted from the analogue filter's VHDL-AMS description is used to fine tune the response of analogue filter hardware. In Case Study 1, the direct calculation method for bandpass filters is presented, while Case Study 2 presents a curve-fitting-based optimisation. The parametric optimisation methodology is first applied to a second-order LC bandpass filter with a silicon inductor (Case Study 1), and a fourth-order vertical cascode Chebyshev lowpass filter (Case Study 2). Both filters are intended as filter cells that may be used as the building block of higher-order filter topologies.

Both methodologies are employed within a parametric optimisation process that includes an HSPICE simulation in the optimisation loop. It is imperative to include a full-HSPICE operating point and AC analysis in the evaluation strategy of the parametric optimisation especially because the analogue filter is intended for integrated high-frequency application. Firstly, integrated circuits are implemented by submicron devices where short channel effects can not be easily modelled. Therefore, transistor models that are based on BSIM 3v3 [110] that accurately models such behaviours must be used to obtain accurate simulation results for integrated circuits. Secondly, for high-frequency applications, the transistor models of the submicron devices should be able to predict the MOSFET behaviour at such frequencies. It is deduced from literature [111] that the BSIM 3v3-based transistor model is suitable for frequencies up to 1 GHz. Therefore by using BSIM3v3-based foundry-supplied transistor models for circuit netlists to be simulated by the industry-tested HSPICE simulator, the final result can be accepted with much confidence.

The parametric optimisation process is shown in Figure 4.14. Selection of filter circuit, circuit variables and the maximum and minimum range for the circuit variables are done by the user. The user also supplies the initial values for the circuit variables that must be within its predefined range. The large shaded box in Figure 4.14 represents the parametric optimisation process loop that runs an HSPICE simulation. The values of the circuit variables, or parameters of the components in the circuit are randomly assigned within the predefined boundary. The process is repeated until a satisfactory result is achieved.

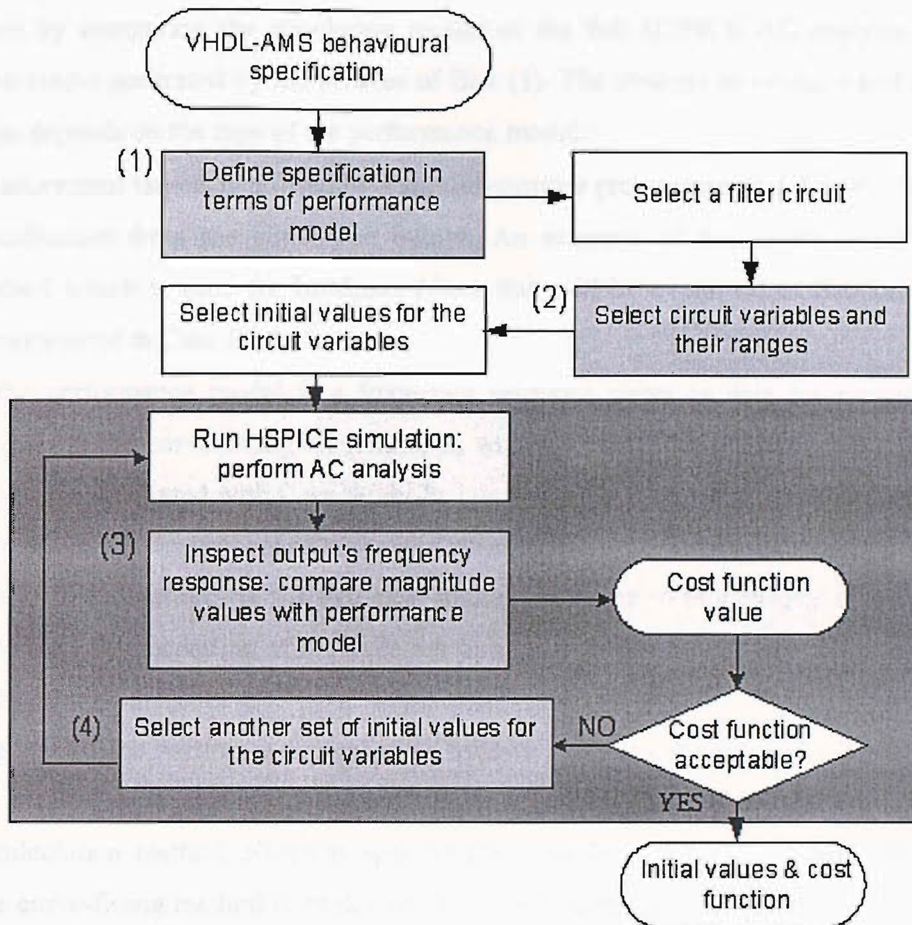


Figure 4.14 Parametric optimisation of an analogue filter circuit.

There are four dark boxes numbered (1) to (4) in Figure 4.14. These represent crucial processes that are demonstrated using the two filter topologies in the case studies, and are explained as follows:

Box (1) represents the process of translating the specification to a performance model, and has been previously explained in Section 4.1.4.

Box (2) represents the process of selecting the circuit variables in the filter circuit to be optimised by the optimisation procedure. This is done by the user. However, the number of parameters to be optimised, as well as the maximum and minimum range for each variable must be carefully selected.

- The selected circuit variables are the ones that have the most influence on the performance of the circuit.
- The upper and lower bounds for the parameters are realistically chosen for practical integrated circuit implementation. This is also important because the next step in the design cycle is to produce the circuit's layout and finally the fabrication of the circuit on chip.

Box (3) represents the process that finds the cost function value in each optimisation iteration. This is done by comparing the simulation results of the full HSPICE AC analysis against the performance model generated by the process of Box (1). The strategy to evaluate and produce the cost function depends on the type of the performance model:

- Measurement targets is utilised by a special-purpose programme that directly calculates the specification from the simulation output. An example of this is the direct calculation method which is used for bandpass filters that will be explained in Section 4.3.2.1 and demonstrated in Case Study 1.
- If the performance model is a frequency response curve or data for magnitude versus frequency, the curve-fitting algorithm, as will be detailed in section 4.3.2.2 can be used. This is demonstrated with Case Study 2.

Box (4) shows the process of automatic generation of the values for the circuit parameters during optimisation iteration. As the two case studies presented in this chapter demonstrate work done prior to the implementation of the three-tier optimisation, the circuit parameters are generated stochastically.

The rest of the section is organised as follows. Sections 4.3.2.1 and 4.3.2.2 and present Case Study 1 and Case Study 2 respectively, including the experimental results. For Case Study 1, the direct calculation method, which is specific for a bandpass function is presented. For Case Study 2, the curve-fitting method is explained. The conclusions of the investigations carried out in Sections 4.3.2.1 and 4.3.2.2 is offered in Section 4.3.2.3.

4.3.2.1 Case Study 1: direct calculation method

```
entity filter is
      port  (quantity vin: real;
               quantity vout: out real);
end entity filter;

architecture behavioural of filter is
      constant Q: real:= 50.0;
      constant frequency: real:= 1e9;
      constant w: real:= 2.0*3.142*frequency;
      constant coeff1: real:= Q/w;
      constant coeff2: real:= 1.0;
      constant coeff3: real:= Q*w;
begin
vin'dot == coeff1*vout'dot'dot + coeff2*vout'dot +
coeff3*vout;
end architecture;
```

The filter specification for Case Study 1 is a 1 GHz second-order bandpass filter with a Q factor of 50. The filter is described in VHDL-AMS as shown above. For a high-frequency bandpass filter suitable for integrated implementation on silicon, the filter topology that is selected for the specifications of Case Study 1 is based on an LC Colpitts oscillator. The circuit includes two capacitors, a spiral silicon inductor and an n-type MOS transistor.

Recall from Box (1) and Box (3) in Figure 4.14 regarding the performance specification and evaluation method for the parametric optimisation process. The performance model for this case study is expressed as measurement targets. As the type of the filter is a bandpass filter, and the required frequency and Q factor is known, this information are used in the direct calculation method, that will be described next.

The flow chart in Figure 4.15 shows the implementation of the strategy to evaluate the accuracy of the results from HSPICE's AC analysis, based on direct calculation. This direct calculation method is applicable for the evaluation of any bandpass filter, either to find the required Q factor at a given frequency, or to investigate the maximum Q factor achievable at the frequency of interest.

As shown in Figure 4.15, the first step is to produce an HSPICE netlist of the circuit under evaluation, and also specifying the frequency and Q factor of interest. Then the counter to count the number of iteration is set. The next step is to run an HSPICE AC analysis on the netlist of the filter circuit, and upon completion, the printout of the magnitude of the circuit's output in HSPICE's result listings file is inspected. The Q factor is calculated by scanning for the maximum amplitude, Amax, and the frequency of this maximum amplitude, Fmax, which is the centre or mid-band frequency for a bandpass filter. If Fmax lies within the range of frequency of interest, the following stage proceeds to find the –3dB frequencies. Now, the Q factor of the response can be calculated.

If the goal of the process is to find the highest possible Q factor, the calculated Q factor is logged if the value is larger than a prescribed value, BestQ. BestQ then takes the current Q factor value. The counter is incremented and another iteration is started. The process of generating a new set of parameters for the netlist, simulating and evaluating is repeated until the loop terminates. If the goal is to find the maximum Q factor, the number of HSPICE iterations is specified as MAX ITER in the flow chart. The termination condition is when the default maximum iteration value has been reached, and the log file contains the value of the best Q factor.

To find a specific Q factor, the iteration loops terminates either when the value found is deemed sufficiently close to specification or when the count has arrived at the default maximum iteration number.

This method is used in Case Study 1, where the specification is to get a bandpass response at 1 GHz with Q factor of 50. The maximum Q factor achievable by the filter circuit is also investigated. Next section introduces the circuit that is used to implement the filter specification of this case study.

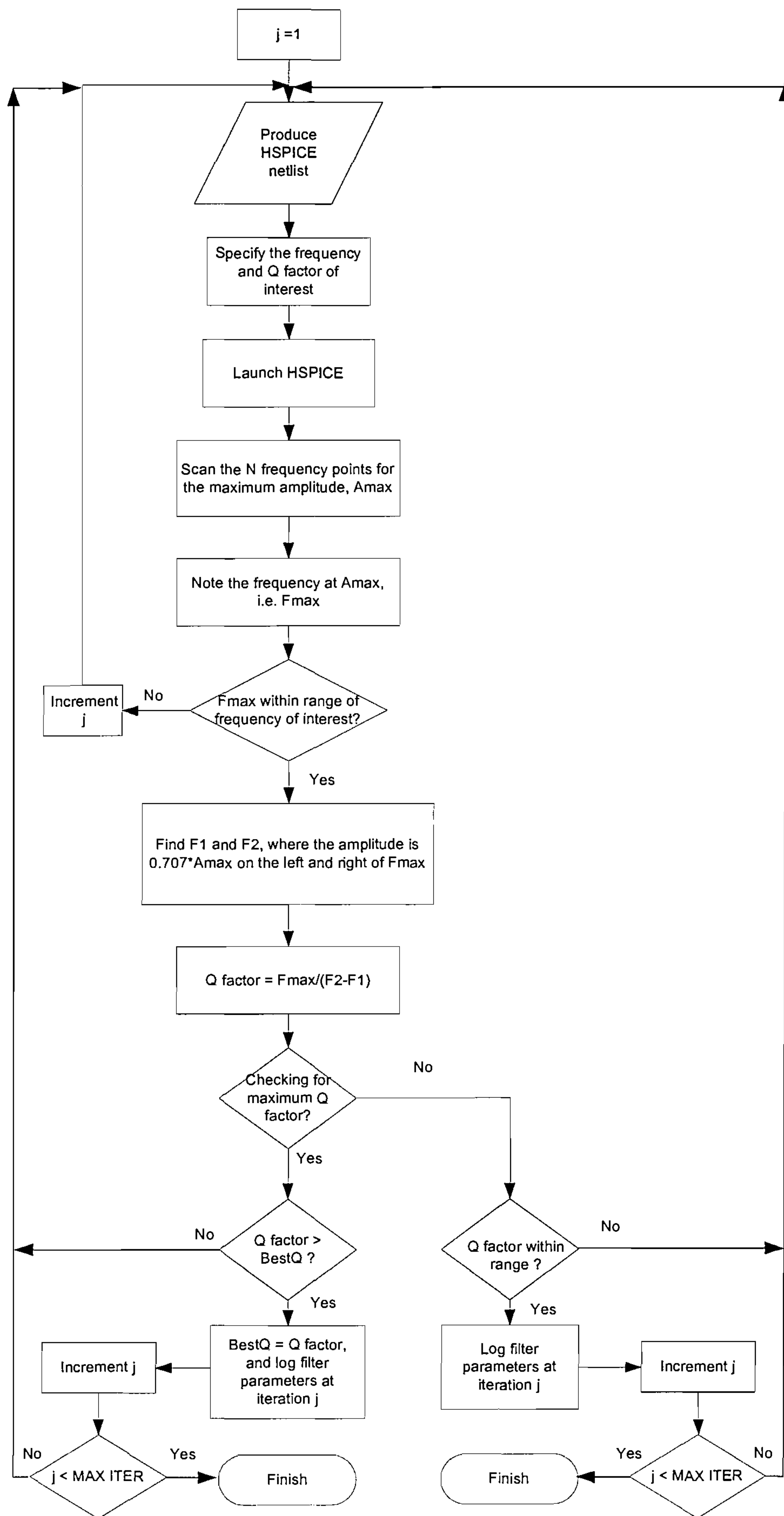


Figure 4.15 Flow chart for the direct calculation method.

4.3.2.1.1 Q-enhanced LC filter based on Colpitts oscillator

The use of active LC circuits for integrated high-frequency applications that uses spiral inductor on silicon has been proposed for its superiority on achieving larger dynamic range for wireless transceivers [112, 113]. Spiral inductors have significant losses and, for a bandpass filter, this results in a very low Q factor. Hence, a Q-enhancement method is used to cancel the effect of these losses by introducing a negative resistance [112, 113].

The second-order filter cell selected for this case study is based on a very effective, Colpitts-type LC oscillator [113] whose positive feedback provides a mechanism for Q-enhancement. The circuit is shown in Figure 4.16. Inductor L is a spiral inductor whose losses are modelled by the series resistor R_S. More accurate models have also been described in literature [112, 114-117]. Although it is possible to construct a higher-value spiral inductor, it is practical to choose a value of less than 10nH [117]. Therefore, for this specific case where the frequency of interest is at 1 GHz, the inductor is chosen to be 8 nH, and the corresponding loss resistance R_S is 5Ω [115].

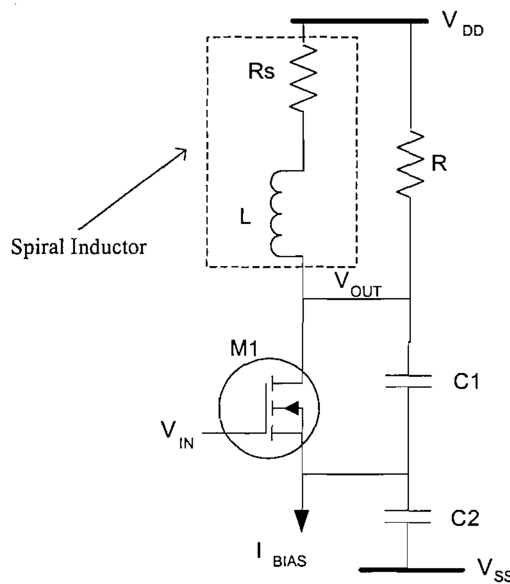


Figure 4.16 Colpitts circuit producing a Q-enhanced bandpass behaviour at output VOUT.

The centre frequency f_C is determined by the values of both capacitors in series, C_T, together with inductor L, as shown in Equations 4.7 and 4.8. Capacitors C1 and C2 act as a voltage divider, where the amount of feedback is controlled by the value of k (where k = C1/C2). The Q-enhancement is determined by the feedback provided by the capacitors, and the transconductance (g_m) provided by transistor M1. The transconductance value is controlled by the current flowing into the transistor from current source I_{BIAS}, and the width W of the transistor M1. This Q-

enhancement mechanism provides a negative resistance of (- 1/ g_m) which cancel out the losses of the inductor.

$$C_T = \frac{C1 \times C2}{C1 + C2} \tag{4.7}$$

$$f_C = \frac{1}{2\pi} \cdot \sqrt{\frac{1}{L \cdot C_T}} \tag{4.8}$$

The inductor Q factor is defined as follows:

$$Q = \frac{2\pi \cdot f_C L}{R} \tag{4.9}$$

With L = 8 nH and R = R_S = 5 Ω, the Q factor is about 10.1. If no Q-enhancement method is used, from simulation, the Q factor of a lossy LC circuit without Q-enhancement at f_C = 1 GHz is 10.2, as shown in Figure 4.17.

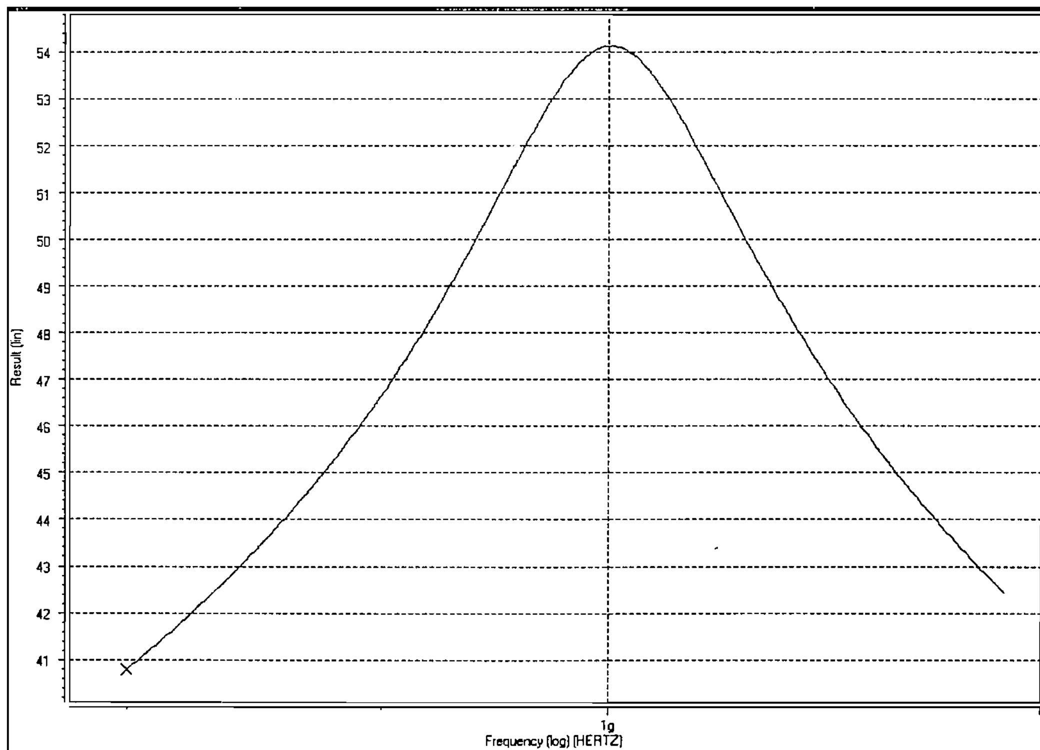


Figure 4.17 HSPICE simulation result for the lossy LC circuit without Q-enhancement; the graph covers the frequency range from 0.8GHz to 1.2GHz.

4.3.2.1.2 Parametric optimisation and experimental results for Case Study 1

From the simulation results in Figure 4.17, it has been shown that the Colpitts circuit need to be parametrically optimised. As explained, the circuit parameters that contributed to the Q factor and its enhancement in an LC circuit are the capacitor values, the bias current and the transistor width. At the same time, the values of the capacitors and the inductor determine the frequency of the bandpass response (Equation 4.8). There is also another optimisation constraint - the inductance value of the spiral inductor- which in order to minimise the losses and parasitic capacitances, should be kept to a minimum [117]. Therefore, in this case, the inductance was kept at a constant value of 8 nH. For the circuit to resonate at 1 GHz, the value of the series capacitance C_T is chosen to be 3.16 pF. The values of C1 and C2 are formulated as shown in Equation 4.10. Therefore, only the value of k is optimised.

$$C1 = C_T \cdot (1+k), \qquad C2 = C_T \cdot \frac{(1+k)}{k}, \qquad 0.01 \le k \le 0.99 \tag{4.10}$$

The objective for the parametric optimisation task is to firstly prove that the Q factor of the circuit can be improved, and therefore the best Q factor attainable within a specified number of iterations is required. Then, the optimisation aims to achieve the specified Q factor of 50. To be able to do this, the results of the AC analysis must be examined each time a new set of parameter values are assigned. The method of evaluating the accuracy of the AC response is based on direct calculation of the specifications from the HSPICE result file and has been described in the previous section. The circuit variables for the HSPICE netlists in this case study are randomly generated. The HSPICE netlist for the Colpitts circuit in Figure 4.16 is shown in Figure 4.18. The transistor model is provided by MIETEC for the 0.35μm technology. The allowed element value ranges are constrained as shown in Table 4.3.

```
C1     N2 N3F 5.024261e-12
C2     N3F N4  8.562172e-12
M1     N2 N5 N3F N4 nmos0553 L=0.35u W=W1
I1     N3F N4 DC IBIAS
VIN    N5 0 DC 0V AC 1V
L      N7 N2 8nH
Rs     N7 N1 5
R      N2 N1 10k
VDD    N1 0 3.3V
VSS    N4 0 -3.3V
```

Figure 4.18 HSPICE netlist of the Collpits oscillator of Figure 4.16.

Parameter	Minimum	Maximum
k	0.01	0.99
W1	0.5 μm	500 μm
I_{BIAS}	0.1 mA	10 mA

Table 4.3 Parameters selected for optimisation in Case Study 1.

Table 4.4shows the summary of the optimisation result for up to 118 iterations. Iterations beyond the 43rd one did not show a noticeable improvement of the Q factor. Therefore it can be concluded that for this particular Colpitts configuration, the largest Q factor that can be achieved is approximately 217, which the optimiser found at the 43rd iteration and took 74 seconds to find. Figure 4.19 shows the HSPICE simulation result for a bandpass response having a Q factor of 217. The experiment is done on a SPARC machine using SunOS version 5.8.

CPU Time (s)	No. of iterations	Q Factor
0	0	14.5
5	1	23.3
9	4	31.0
27	11	33.0
49	27	39.4
53	30	48.2
74	43	217.0
209	118	217.1

Table 4.4 Experimental results for Case Study 1, using a purely stochastic method in generating the filter parameters, and optimising to get the best Q factor.

For the case of optimising the circuit for a Q factor of 50, the results can also be seen in Table 4.4 as well. At the 30th iteration, the Q factor was calculated to be 48.2, which is within the acceptable range. Therefore the stochastic search method was able to find an optimal set of parameters within 53 seconds. Figure 4.20 shows the result for a bandpass response with Q factor of about 50.

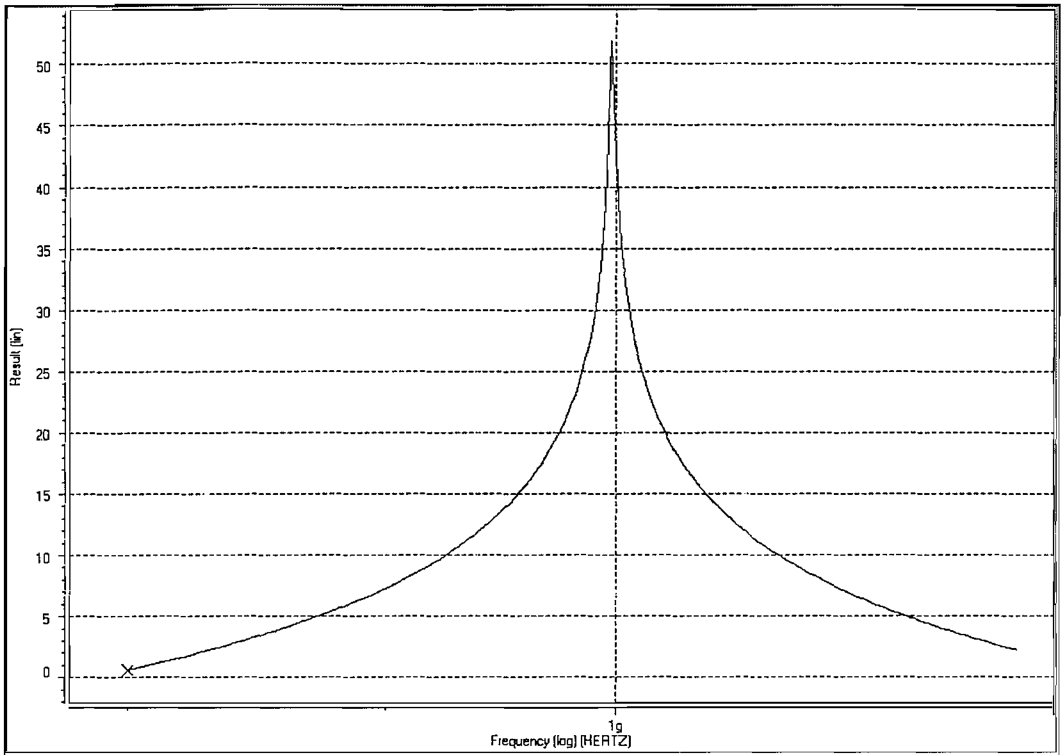


Figure 4.19 HSPICE simulation result of the filter optimised for the maximum Q factor (Q = 217); the frequency range is 0.8 GHz to 1.2 GHz.

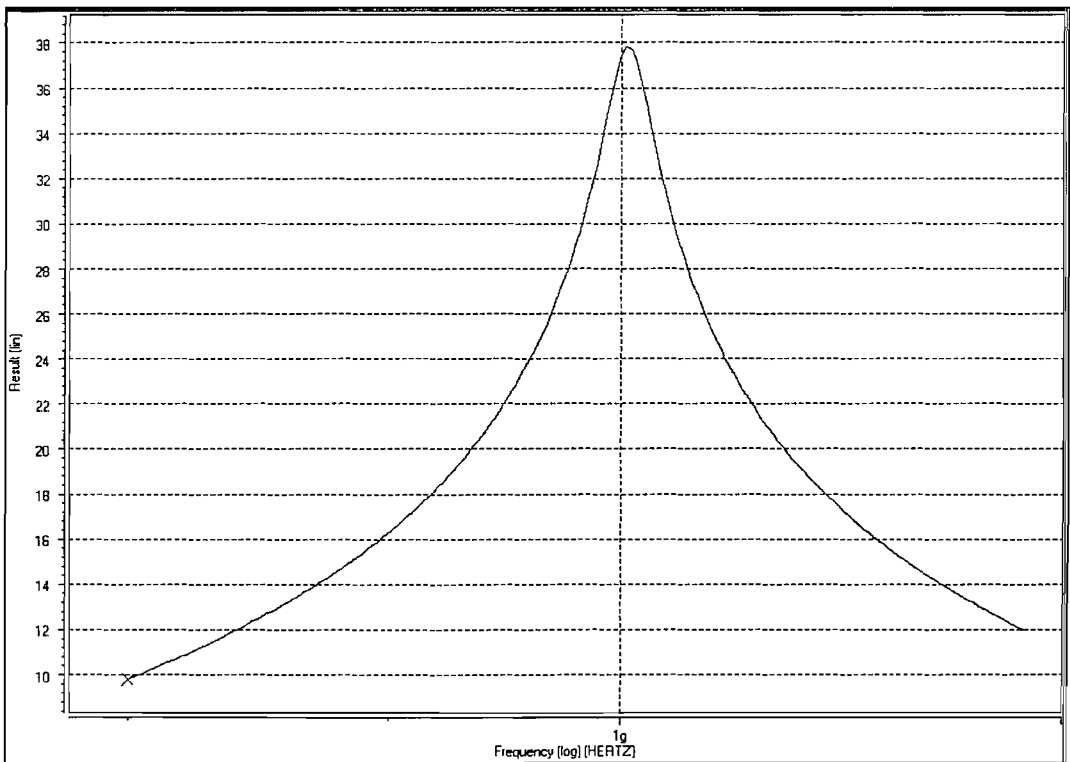


Figure 4.20 HSPICE simulation result of the filter optimised for the Q factor of 50; the frequency range shown is 0.8 GHz to 1.2 GHz.

4.3.2.2 Case Study 2: curve-fitting method

Case Study 2 is of a fourth-order lowpass Chebyshev filter with pass-band ripple of 0.5 dB. The coefficients that are taken from a filter table [108] are used to build the transfer function, which is implemented in VHDL-AMS as shown below. This filter is de-normalised to cut-off frequency of 1 GHz. The VHDL-AMS code is used to generate the transfer function's frequency response curve, as shown in Figure 4.21. This curve is used by the curve-fitting method that is described next.

```
entity filter is
      port (quantity Vin: real;
            quantity Vout: out real);
end entity;

architecture transfer of filter is
      constant frequency: real:=1e9;
      constant w: real:= 2.0*3.142*frequency;
      constant w2: real:= w*w;
      constant w3: real:= w2*w;
      constant w4: real:= w2*w2;
      constant a:real:=0.3579*w4;
      constant b:real:=0.3791*w4;
      constant c:real:=1.0255*w3;
      constant d:real:=1.7169*w2;
      constant e:real:=1.1974*w;
      constant num: real_vector:= (a);
      constant den: real_vector:= (b,c,d,e,1.0);
begin
      Vout == Vin'LTF(num,den);
end architecture;
```

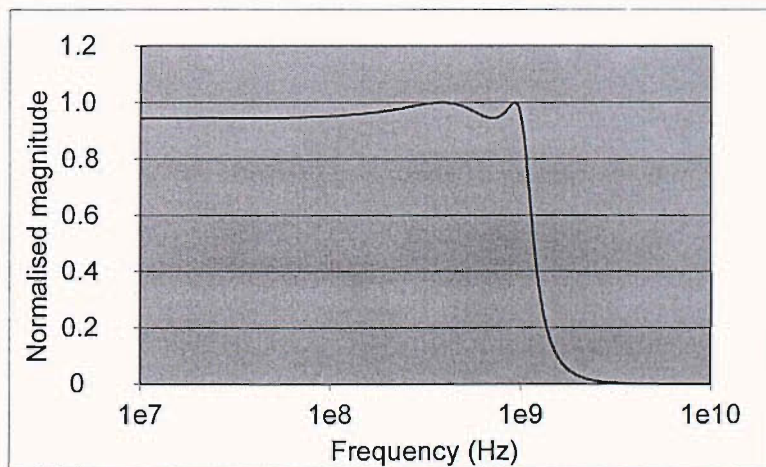


Figure 4.21 The ideal curve for Case Study 2.

The goal of an optimisation problem in general is to minimise an objective function or cost function. The most common measure for the cost function is the difference between some desired characteristic and its actual value. In this application, the cost function is defined as the least-squares error calculated from the ideal filter response and the HSPICE simulation result of the filter circuit. The error is expressed as follows:

$$f(C)=\sqrt{\frac{1}{N}\sum_{k=1}^{N}(ideal_k-result(C)_k)^2}, \qquad 1\leq k\leq N \tag{4.11}$$

where C is the set of optimised parameters, *f(C)* is the cost function, *N* is the number of frequency points within the frequency range of the simulation of the HSPICE AC analysis, $ideal_k$ and $result(C)_k$ are the magnitudes of the transfer function at each point *k*, obtained by calculation and HSPICE simulation, respectively.

Thus, the cost function aims to produce a netlist for which the HSPICE simulation most closely matches or fits the curve of the ideal response. The more the simulated curve fits the ideal curve, the smaller the difference between the two or error. The user may specify a tolerance value for *f(C)* below which the optimisation loop terminates as the error between the desired and simulated result is deemed small enough. It is implemented using the algorithm shown in Figure 4.22. A note on the method to produce the ideal filter AC response as shown in the flow chart on the right-hand side of Figure 4.22, is that this is applicable for a filter description using its transfer function only.

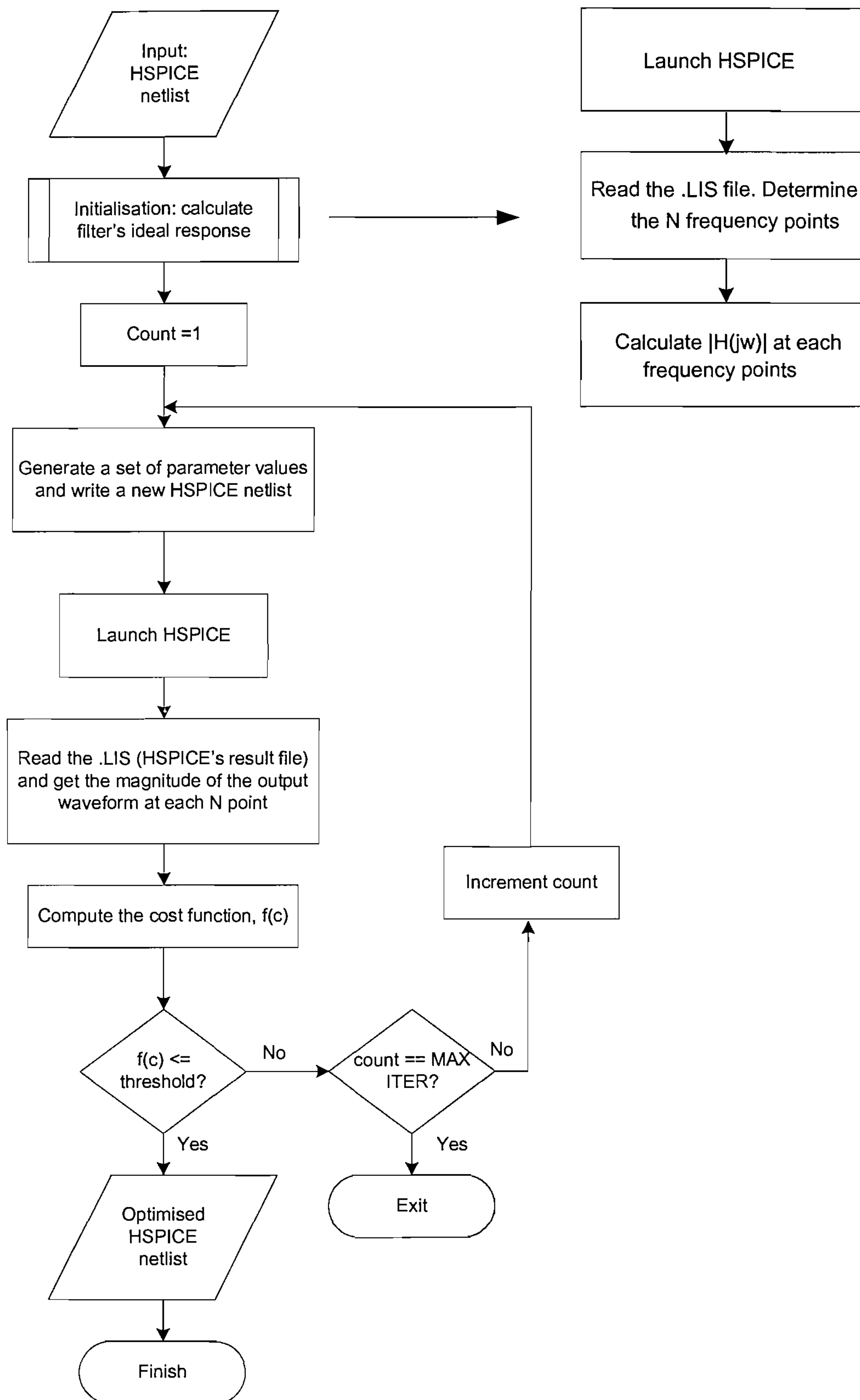


Figure 4.22 Algorithm of the curve-fitting parametric optimisation.

4.3.2.2.1 Vertically stacked current-mode biquadratic filter

The circuit chosen for Case Study 2 is a vertically stacked current-mode biquadratic filter implemented around the regulated cascode configuration [118]. It has been selected for its suitability for integrated high-frequency and low power properties as well as its design simplicity.

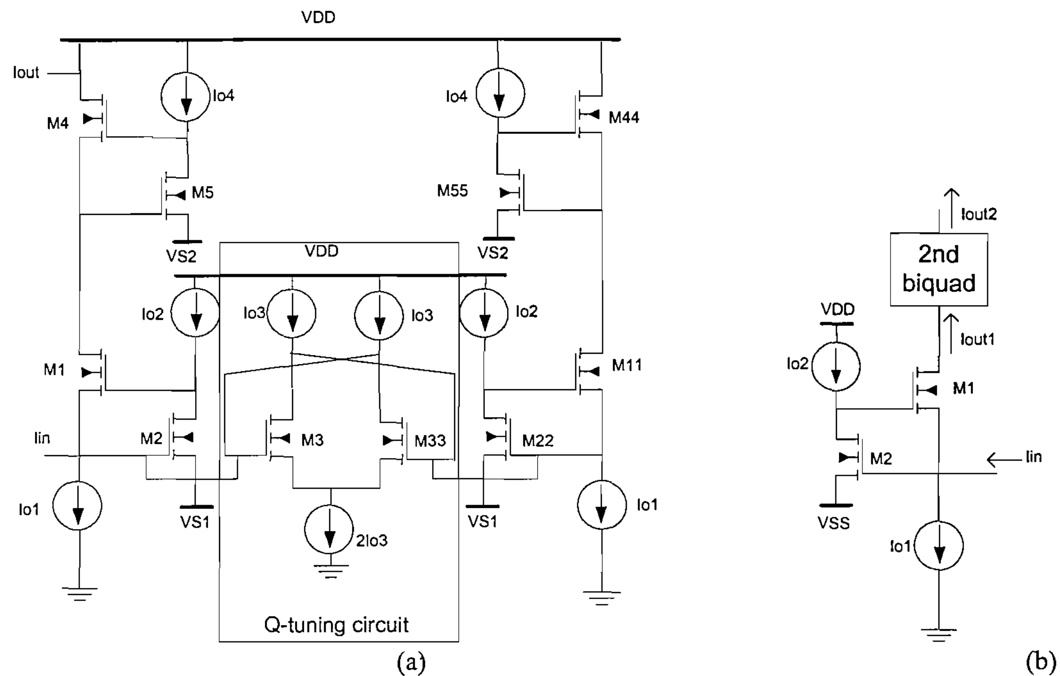


Figure 4.23 (a) A fourth-order lowpass Chebyshev filter, (b) Vertical stacking of a second-order regulated cascode.

The circuit in Figure 4.23 (a) is a fourth-order Chebyshev lowpass filter with 1 GHz cut-off frequency and 0.5dB ripple, and is realised by vertically stacking two fully-differential biquad circuits. Figure 4.23 (b) shows the regulated cascode biquad. In this diagram, the output currents Iout1 and Iout2 represent the lowpass output for the second-order and fourth-order filter. Bandpass response can be obtained by taking the output current from the drain of transistors M2. The regulated cascode biquad is actually lossy as the bandpass current is negatively fed back into the input terminal and thus gives result to a lower Q factor. To compensate for this loss, a Q-tuning circuit is implemented using the cross-coupled long-tail pair of transistors M3 and M33 as shown in Figure 4.23 (a). The Q factor of the circuit can be independently tuned by varying the transconductance of transistor M3 via Io3. Meanwhile, frequency tuning is done by varying the bias current Io1.

Firstly, an example of an un-optimised response will be shown. Figure 4.24 shows the HSPICE simulation result of the fourth-order lowpass filter using the original circuit parameters as in [118]. It can be seen that the circuit produces a high overshoot of almost 7 dB with a cut-off frequency exceeding 1 GHz.

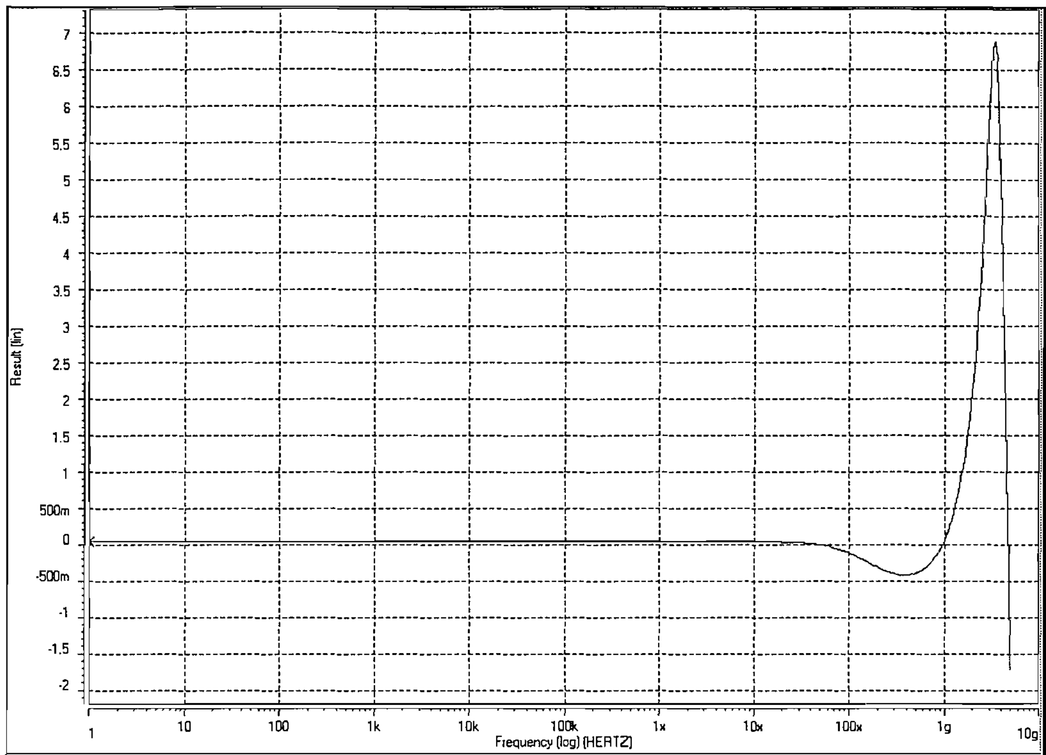


Figure 4.24 AC characteristic before optimisation using original circuit parameters as in the reference source[118].

4.3.2.2.2 Parametric optimisation and experimental results for Case Study 2

The curve-fitting methodology described previously is used in this case study. The HSPICE code containing the netlist is shown in Figure 4.25 and is simulated on HSPICE using the device specifications as quoted in the paper [118]. There are 7 parameters to be adjusted by the optimisation stage: all the 5 transistor widths and the bias currents (Io1 and Io3) for transistors M1 (and M11), and M3 (and M33). In the netlist, the five sets of transistor widths are marked as WM1-WM5, while the two currents are IO1 and IO3. Current IO33 has twice the value of IO3. The initial values and ranges for each parameter are defined in Table 4.5. In each iteration, the values of the parameters are randomly generated within the stated ranges.

```
m1        n6 n14 n7 0 nmos0553  l=0.35u  w=wm1
m11       n3 n10 n8 0 nmos0553  l=0.35u  w=wm1
m2        n14 n7 n11 0 nmos0553  l=0.35u  w=wm2
m22       n10 n8 n11 0 nmos0553  l=0.35u  w=wm2
m3        n8 n7 n13 0 nmos0553  l=0.35u  w=wm3
m33       n7 n8 n13 0 nmos0553  l=0.35u  w=wm3
m4        out1 n2 n6 0 nmos0553  l=0.35u  w=wm4
m44       out11 n5 n3 0 nmos0553  l=0.35u  w=wm4
m5        n2 n6 n16 0 nmos0553  l=0.35u   w=wm5
m55       n5 n3 n16 0 nmos0553  l=0.35u   w=wm5
io1       n7 0 dc io1
io11      n8 0 dc io1
io2       n1 n14 dc 20u
io22      n1 n10 dc 20u
io3       n1 n8 dc io3
io33      n1 n7 dc io3
io333     n13 0 dc io33
io4       n1 n2 dc 1u
io44      n1 n5 dc 1u
vs1       n11 0 0.3v
vs2       n16 0 0.65v
```

Figure 4.25 Netlist for fourth-order lowpass Chebyshev filter.

Parameter	Initial value	Minimum	Maximum
WM1	10e-6	0.5e-6	15e-6
WM2	10e-6	0.5e-6	15e-6
WM3	10e-6	5e-6	15e-6
WM4	82e-6	50e-6	120e-6
WM5	1e-6	0.5e-6	10e-6
IO1	40e-6	5e-6	80e-6
IO3	8e-6	1e-6	30e-6

Table 4.5 Parameter ranges for the parametric optimisation of Case Study 2.

The ideal curve which is used in the curve-fitting method consists of 55 points between the frequencies of 10 MHz and 5 GHz. This means that the value N in Equation 4.11 is 55. The termination criterion as shown in the flow chart of Figure 4.22 is either when f(c) is smaller than or equal to a threshold value of 0.05 or if the maximum iteration count, MAX ITER, of 500 is reached.

The experiment is done on a SPARC machine using SunOS version 5.8. The curve-fitting optimisation concludes after 500 iterations, with an error figure f(c) of 0.298 that was obtained on the 87th iteration.. HSPICE simulation waveforms for the optimised fourth-order lowpass Chebyshev filter circuit is shown in Figure 4.26.

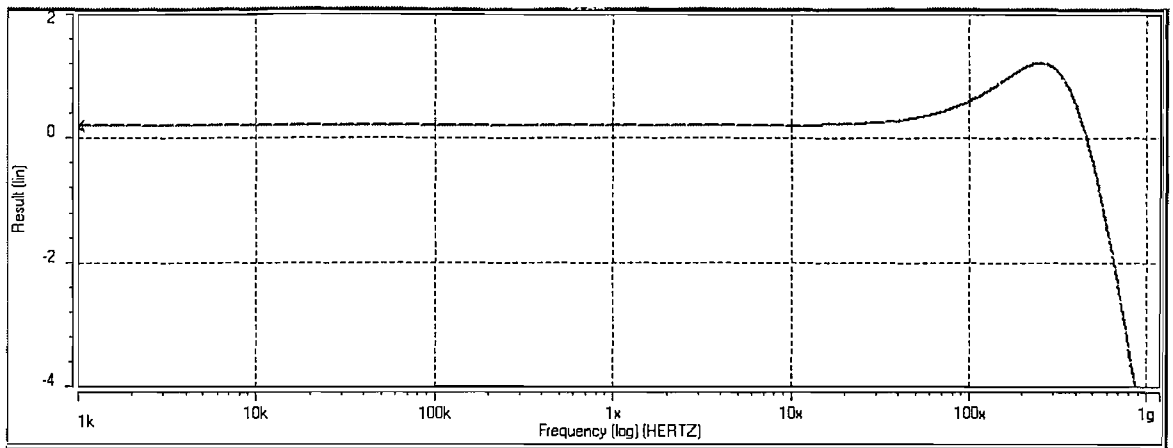


Figure 4.26 HSPICE simulation results for the parametric optimisation of Case Study 2.

4.3.2.3 Parametric optimisation strategy: curve-fitting vs. direct calculation

The previous sections present the parametric optimisation strategies for the synthesis of integrated high-frequency filters which are demonstrated with two case studies. The two case studies have shown that the optimisation strategies are well-suited for integrated high-frequency implementations. Important steps have been identified: 1) specification of performance model, 2) selection of circuit variables and their ranges, 3) evaluation of the accuracy for each candidate against the performance specification and 4) generation of a new set of circuit variables for the next optimisation run.

Case Studies 1 and 2 show how the filters' behavioural specifications in VHDL-AMS are used to evaluate the filters' AC frequency response. The first way to model the performance, as in Case Study 1, is suitable for bandpass filters where the requirement to check for several amplitude measurements at specified frequencies ensure the validity of the result. As demonstrated in Case Study 2, the method of generating the ideal curve of a filter based on its VHDL-AMS transfer function description is attractive in terms of its flexibility and generality, especially for filter approximation functions that has ripples in the pass-band and/or stop-band. Therefore, the three-tier optimisation algorithm relies on the curve-fitting methodology that can be implemented using one of HSPICE optimisation's features that will be described later in Chapter 5. The three-tier optimisation algorithm also employs a bandpass checking methodology, similar to the direct calculation method of Case Study 1 for the optimisation of bandpass filters.

Nevertheless, it follows that the evaluation of the performance of a circuit in the optimisation loop reveals the drawback of using the ideal curve and curve-fitting method in terms of both efficiency and accuracy especially if the ideal curve, or performance specification, is not properly chosen. This can be demonstrated with the following example on the choice of performance specification. Two different models are used to represent the specification of a fourth-

order, 1 GHz lowpass Chebyshev filter with 0.5 dB ripple, similar to the specification of Case Study 2. In each model, different data points in the magnitude vs. frequency curve are selected, as shown in Figure 4.27, Figure 4.28 and Figure 4.29.

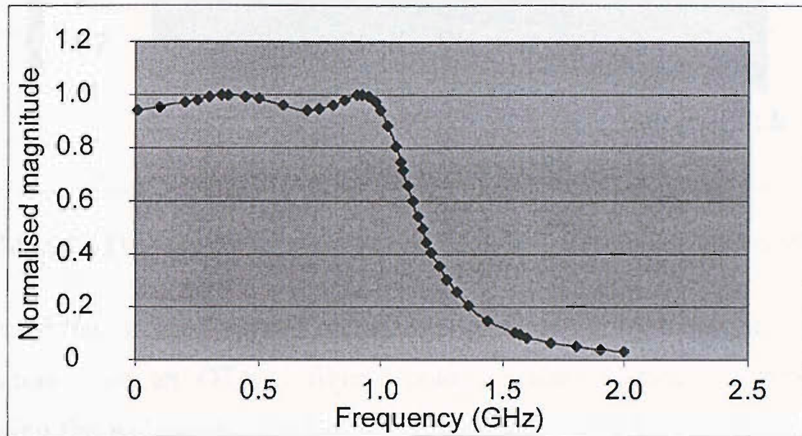


Figure 4.27 Model 1 (44 points between 1e7 to 5e9 Hz): points are evenly scattered in both passband and stopband.

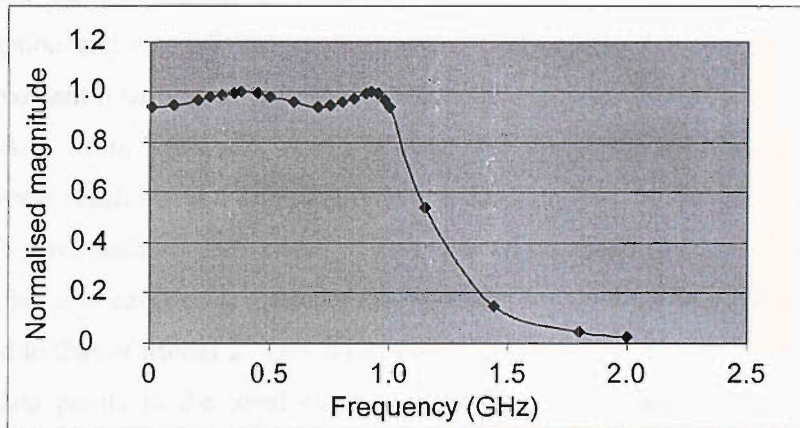


Figure 4.28 Model 2 (23 points between 1e7 to 2e9 Hz): more points in passband.

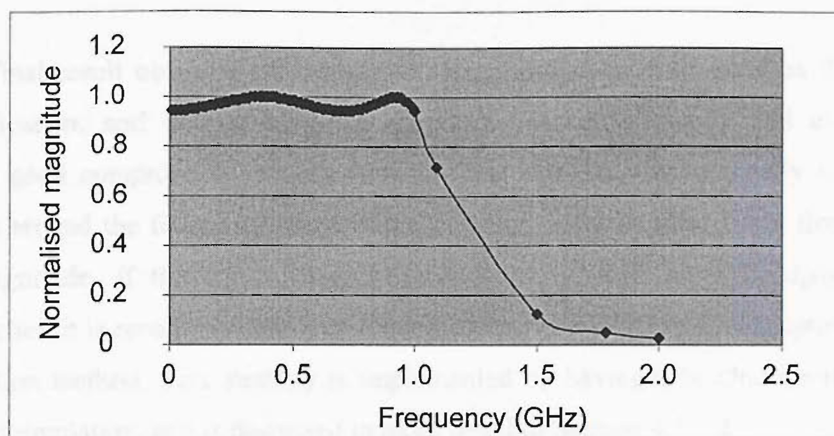


Figure 4.29 Model 3 (104 points between 1e7 and 2e9 Hz): even more points in the passband.

The three different 'ideal curves' are used in the three-tier optimisation – which is a curve-fitting optimisation - on an OTA-C filter topology, called Cascade 1, where the OTA is implemented using the wide-swing configuration. Cascade 1 will be explained in more detail in Chapter 6. The optimisation results are shown in Table 4.6, which includes the number of iteration that algorithm took to converge to a solution, the cost function value (*CF*) as well as the power consumption and curve-fitting error, err_{total}. The cost function formulation that consists of the power consumption and curve-fitting error value will be described in the next section, Section 4.3.3. For the moment it suffices to say that the lower the value of the curve-fit error, the closer the match to the ideal. From Table 4.6, it can be seen that the optimisation that employs Model 1 produces the worse result but at a comparatively quicker rate. The best result is obtained by using Model 3, which gives such accurate detail of the expected passband response that it not only gives the best result, but also causes the optimisation algorithm to converge to a solution at a faster rate when compared to that of Model 2. This faster convergence rate of Model 3 over Model 2 despite having more data points in the ideal curve is expected as the curve-fitting-based optimisation algorithm is more informed or guided in the challenging frequency region that has the passband ripples. However, to obtain such detailed information may not be possible in each filter synthesis / optimisation task, therefore it is more practical to specify the minimum number of data points in the performance model specification that is deemed sufficient for the three-tier optimisation algorithm to produce a correct and accurate result.

	Iteration No.	*CF*	Power (mW)	err_{total}
Model 1	206	0.202082	61.57	0.1405
Model 2	390	0.006624	36.22	0.0300
Model 3	291	0.004229	21.83	0.0218

Table 4.6 Experiment on performance model specification in a curve-fitting optimisation.

The final result obtained from curve-fitting optimisation is as good as the performance model specification, and will involve the trade-off between accuracy and efficiency. From experience, a good compromise between accuracy and efficiency is to specify several carefully chosen points around the frequency of interest instead of using an ideal curve than spans several orders of magnitude. If the curve-fitting method is to be used for a bandpass response, as mentioned earlier, it is recommended to combine both the use of curve-fitting optimisation as well as the inspection method. This strategy is implemented by having a bandpass error check in the cost function formulation, and is discussed in more detail in Section 4.3.3.2.

When a filter topology has been identified to realise the specification, the key process is then to select the appropriate circuit variables and the ranges. Having too many variables does not guarantee an optimal or even correct performance. The best way to select the suitable parameter and ranges is to have an understanding of the topology from analysis and documentation. Details regarding analogue filter topologies suitable for integrated high-frequency applications are given in Chapter 6.

4.3.3 Cost function formulation and evaluation

The HSPICE simulator plays a major role in evaluating the performance of the topology under consideration. It has several features that can be employed to aid the parametric optimisation of the filter topologies. Firstly, HSPICE has an optimisation feature that can be used to perform curve-fitting optimisation. This feature will be explained in more detail in Chapter 5. Its capabilities to perform various analyses, whose results can be presented in several ways, allow us to inspect several performance criteria which are of interest in the optimisation process. The performance measure or criteria used in the formulation of the cost function for each analogue filter candidate is accuracy and power consumption.

The accuracy measure is obtained from the error figure generated by the HSPICE curve-fitting algorithm, which gives a least-squares estimate of how much the simulated AC frequency response differs from the magnitude of the desired response. The least-squares error figure used by the HSPICE curve-fitting algorithm is given in Equation 4.12 [119].

$$err_{total} = [\frac{1}{n} \cdot \sum_{i=1}^{n} err_i^{\,2}]^{1/2} \tag{4.12}$$

where,

$$err_i = \frac{M_i - C_i}{\max(MINVAL, M_i)}, i = 1, n \tag{4.13}$$

where n is the number of points in the frequency and M and C are the measured and calculated values respectively. Function *max()* selects the higher between M and *MINVAL* at a particular point, where *MINVAL* is defaulted at 1e-12.

Apart from the accuracy, another optimisation requirement is that the power consumption of the topology under consideration to be as low as possible. The total power consumption required by the cost function can be obtained directly from the operating point data in the HSPICE results file. Both requirements on accuracy and power are expressed in the cost function *CF* as follows:

$$CF = err_{total} \cdot w + Power + err_{bp} \tag{4.14}$$

where w is a weighting factor to ensure that a penalty given to err_{total} is high. *Power* is the total power consumption in Watts, and err_{bp} is a penalty calculation specific to the bandpass filter only. The weighting factor w and bandpass penalty err_{bp} will be discussed in Sections 4.3.3.1 and 4.3.3.2 respectively.

3TOptFunc( )

Input: a set of parameter's initial values.

1. Construct a new netlist:

Prepare .PARAM statements with current parameter values.

Write newly constructed netlist to a file.

2. Launch HSPICE.

3. Read the results file and extract the following data:

Curve fitting optimisation results, i.e. the value of err_{total}

Power consumption

For a bandpass filter, the frequencies fpeak, f1 and f2

4. Assign weighting penalty.

5. For a bandpass filter, compute the value of err_{bp}.

6. Compute CF value.

7. Log results.

Output: CF.

Figure 4.30 Pseudo code of the basic cost function evaluator.

The cost function is implemented in the function called *3TOptFunc()*, which outputs the cost function value for a given set of parameter values. The pseudo code for the basic operation of the function *3TOptFunc()* is shown in Figure 4.30. The inputs to *3TOptFunc()* are the parameter starting points and the output value *CF* is evaluated according to Equation 4.14. A detailed version of *3TOptFunc()* being used in the three-tier algorithm will be presented in Chapter 5.

4.3.3.1 Cost function weighting assignments

Three experiments were carried out to compare the effect of different weightings on the cost function. The cost function is used by the three-tier optimisation algorithm, and its value influences the rate of convergence of the algorithm and the accuracy of the final result. It is desirable for the cost function to be able to cause the optimisation algorithm to converge quickly to a minimum, i.e. giving a result that has the lowest curve-fitting error figure and lowest power consumption.

The three different weighting assignments are used in the cost function for the three-tier optimisation on a synthesis example of a fourth-order lowpass filter at 1 GHz, where the topology selected for implementation is the cascaded OTA-C circuit using the wide-swing OTA cell.

Experiment 1: The corresponding values of the accuracy weighting w are assigned as listed in Table 4.7. The values of the weighting factor w are selected to give more prominence to the accuracy, and effectively ignore the power consumption, if the accuracy errors are excessively large. For curve-fitting error at the lower range of values, the weighting is relaxed so that the contribution from the power consumption in the cost function formulation is considered as well.

Curve-fitting error value	w
$err_{total} > 0.9$	10
$0.6 < err_{total} \leq 0.9$	5
$0.4 < err_{total} \leq 0.6$	4
$0.3 < err_{total} \leq 0.4$	3
$0.2 < err_{total} \leq 0.3$	2
$err_{total} \leq 0.2$	1

Table 4.7 The non-linear penalty values assigned to the curve-fitting error function in the cost function of Equation 4.14.

Experiment 2: In this experiment, the weightings are assigned linearly at ascending values of the curve-fitting error, as shown in Table 4.8.

Curve-fitting error value	w
$err_{total} > 0.9$	10
$0.8 < err_{total} \leq 0.9$	9
$0.7 < err_{total} \leq 0.8$	8
$0.6 < err_{total} \leq 0.7$	7
$0.5 < err_{total} \leq 0.6$	6
$0.4 < err_{total} \leq 0.5$	5
$0.3 < err_{total} \leq 0.4$	4
$0.2 < err_{total} \leq 0.3$	3
$0.1 < err_{total} \leq 0.2$	2
$err_{total} \leq 0.1$	1

Table 4.8 The linear penalty values assigned to the curve fitting error function in the cost function of Equation 4.14.

Experiment 3: Weight is 5 regardless of the value of err_{total}. Therefore the cost function is formulated as follows:

$$CF_{\exp 3} = 5.err_{total} + Power + err_{bp} \tag{4.15}$$

Experimental results

The summary of the results are presented in Table 4.9. The table shows the number of iteration that it takes for the three-tier optimisation algorithm to converge, as well as the value of the resulting cost function (together with the curve-fit error and power consumption values) that it finally converged to. Figure 4.31, Figure 4.32 and Figure 4.33 show the relationship between cost function against the iteration number for Experiments 1, 2 and 3 respectively.

Experiment	Iter. No.	*CF*	Power (mW)	err_{total}
1	74	0.229	93.7	0.136
2	90	0.565	294	0.136
3	83	0.998	319	0.136

Table 4.9 Experimental results for the weighting assignments values.

From these figures and Table 4.9, it can be seen that the weighting assignments of Experiment 1 gives the best result, as it causes the three-tier optimisation algorithm to converge at the fastest rate, and gives the lowest power consumption. The curve-fit error for the three experiments converged to the same value, although if the values are taken up to 5 decimal points, the curve-fit error for Experiment 1 turned out to be the lowest, followed by Experiment 2 and 3.

The effect of the cost function formulation can be seen from Figure 4.31,Figure 4.32 and Figure 4.33. The series of high points of the cost function values seen in the figures indicate that the optimisation algorithm is testing or perturbing a direction in the cost function surface in search for a minimum. The graph of Experiment 1 shows smoother transitions from one iteration to another, until it settles down and converges, whereas the behaviour shown in Experiments 2 and 3 is more erratic. Therefore, the weighting assignments of Table 4.7 of Experiment 1 is chosen in the formulation of the cost function *CF* of Equation 4.14.

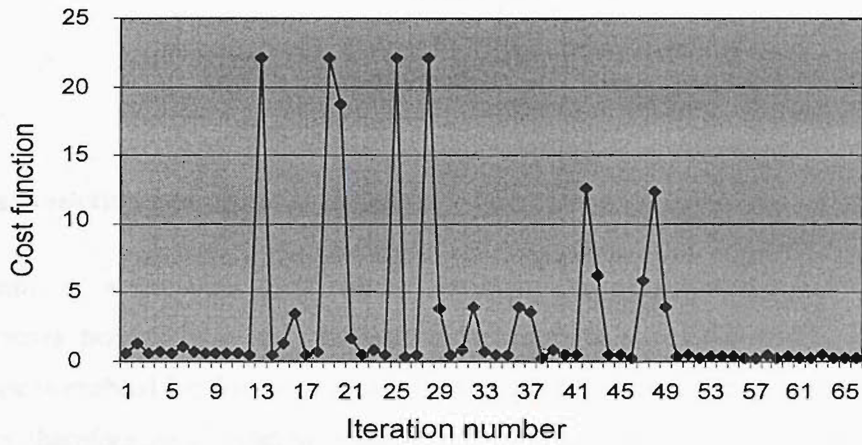


Figure 4.31 Results for Experiment 1.

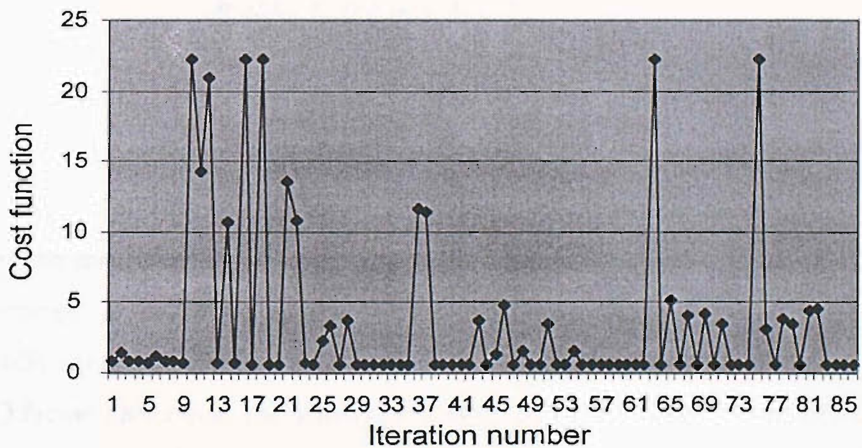


Figure 4.32 Results for Experiment 2.

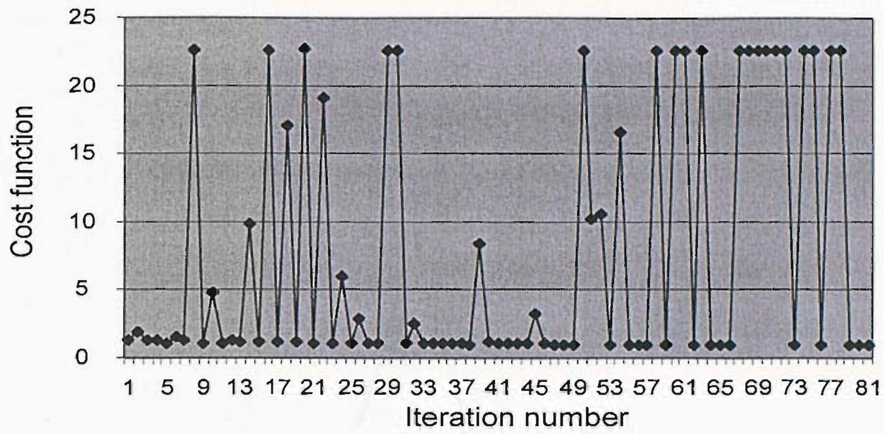


Figure 4.33 Results for Experiment 3.

4.3.3.2 Cost function bandpass error check

The optimisation of a bandpass filter requires an extra check to ensure a correct result, and therefore an extra penalty term err_{bp} is used in the formulation of *CF* (Equation 4.14). This bandpass check is enabled for the optimisation of bandpass filter topologies, and disabled for other types of filters, therefore, err_{bp} is set to zero for non-bandpass filtering. For a bandpass filter, err_{bp} is zero if the following conditions are fulfilled:

$$\begin{aligned} & fpeak_{lo} \leq fpeak \leq fpeak_{hi} \\ & f1 < f2 \\ & Q_{lo} \leq \frac{fpeak}{f2 - f1} \leq Q_{hi} \end{aligned} \tag{4.16}$$

where *fpeak* is the measured frequency at which the peak occurs, while $fpeak_{lo}$ and $fpeak_{hi}$ is the allowed *fpeak* range, *f1* and *f2* are the lower and upper frequencies at which the amplitude of *fpeak* drops to its -3dB value (*fpeak* , *f1* and *f2* is illustrated in Figure 4.34). Q_{lo} and Q_{hi} is the allowed range of the Q factor. Otherwise, the penalty err_{bp} is set to 10. Therefore, if the measured response fails the bandpass check, the cost function of Equation 4.14 will have a high value, and the optimisation algorithm will be driven away from the potentially 'bad' direction.

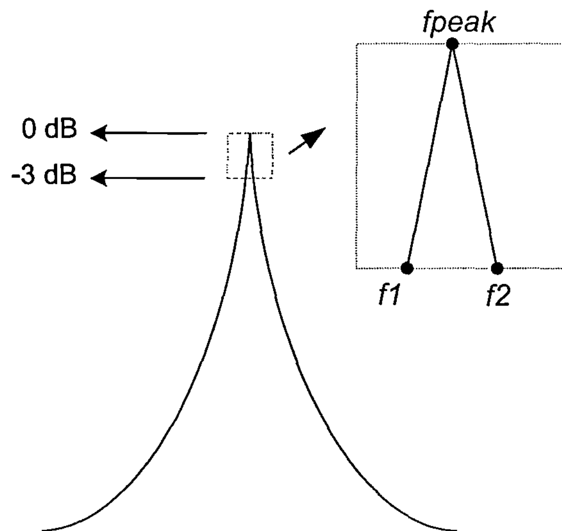


Figure 4.34 The position of *fpeak*, *f1* and *f2* on a bandpass response.

An example to justify the use of the bandpass check for the performance model specification having ideal points in the passband only as well as using points situated in both the passband and stopband is carried out. In this example, the required filter response is a 1 GHz fourth-order response with Q factor of 10. The topology selected to implement the specification is an OTA-C cascaded filter using the wide-swing OTA, which is named Cascade 4. This topology will be explained in more detail in Chapter 6. Four experiments are carried out by optimising Cascade 4 using the three-tier algorithm, where two different performance models are used, where in each case, the bandpass check is enabled and disabled. The performance models, Model 1 and Model 2 are given below:

Model 1: Eleven points in the passband are selected to represent the ideal curve of Figure 4.35. The frequency ranges of the points are 0.95 to 1.05 G Hz, which spans the magnitude between 0 to – 3 dB.

Model 2: The ideal specification is represented by fifteen points between 0.25 and 4 GHz, which include the points of Experiment 1 as well as four additional points as shown in Figure 4.36 below. The additional points are placed at -24 and -48 dB.

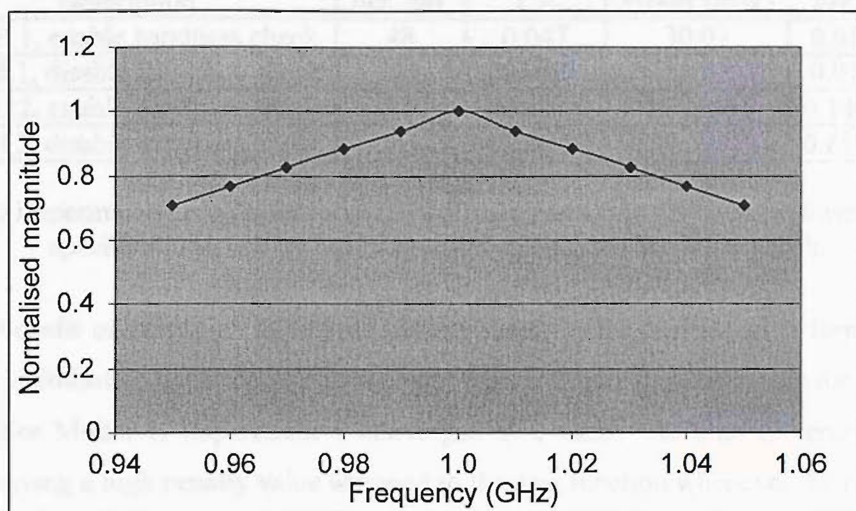


Figure 4.35 Model 1 (11 points in the passband) used in Experiments 1 and 2.

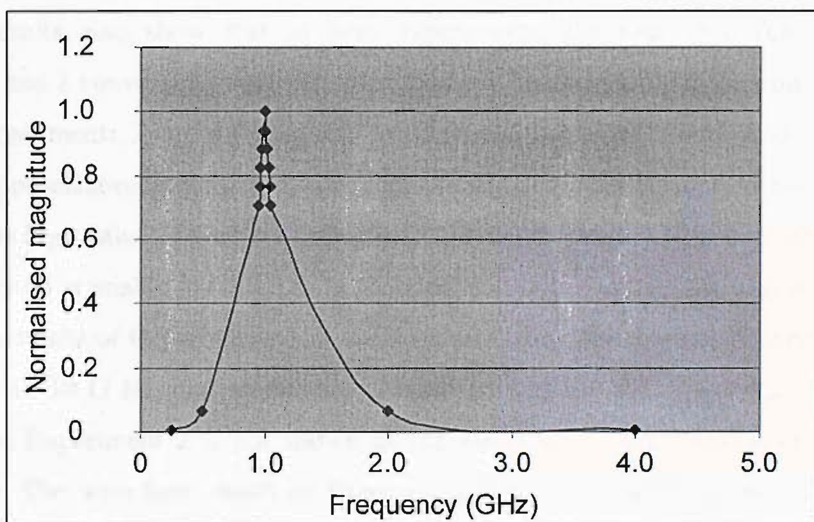


Figure 4.36 Model 2 (15 points in the passband and stopband) used in Experiment 3 and 4.

Model 1 is used in Experiments 1 and 2, while Model 2 is used in Experiments 3 and 4. The bandpass check is enabled in Experiments 1 and 3, and disabled in Experiments 2 and 4. The three-tier optimisation results for the experiments are shown in Table 4.10, where the results columns show the number of iteration that it took to converge, the final cost function value (CF) and its respective values for power and curve-fit error (err_{total}). The AC waveforms from the experiments are shown in Figure 4.37 to Figure 4.42, where the dark curve in each figure shows the waveform of the ideal curve.

Exp.	Description	Iter. No.	CF	Power (mW)	err_{total}
1	Model 1, enable bandpass check	48	0.047	30.07	0.0169
2	Model 1, disable bandpass check	152	0.052	34.72	0.0171
3	Model 2, enable bandpass check	87	0.19315	46.93881	0.14621
4	Model 2, disable bandpass check	135	4.138359	39.6095	0.81975

Table 4.10 Experimental results for three-tier optimisation using different performance model specifications, and by enabling and disabling the bandpass check.

The benefit of having an additional penalty factor in the cost function formulation of the curve-fitting optimisation of bandpass filters can be seen by observing the results for Experiments 1 to 4 above. For Model 1, Experiment 1 converged at a faster rate than Experiment 2. This is because, by having a high penalty value assigned to the cost function whenever the measured result fails the bandpass check, the three-tier optimisation algorithm is steered away from a 'wrong' direction. Therefore, this bandpass check helps the optimisation algorithm in deciding the direction to move much earlier, rather than the optimisation algorithm needing to proceed in the direction and finally discovering that it is indeed a 'bad' direction after a significant number of iterations.

The results also show that in both experiments, the final cost function values that Experiments 1 and 2 converged to are very near, and can be considered to be similar. Similarly, for the case of Experiments 3 and 4 employing performance model of Model 2, the bandpass check also aided the optimisation algorithm to converge quickly at a more accurate or correct solution. As indicated by the high value of the cost function and err_{total} in Table 4.10 and as observed in Figure 4.37, Experiment 4 is unable to converge to a correct solution. Figure 4.40, Figure 4.38 and Figure 4.37 shows the results of Experiments 1, 3 and 4 viewed using zoom factor A, where the frequency is between 0.1 to 10 G Hz, and magnitude is between 3 to -50 dB. (Note that the resulting AC waveform from Experiment 2 is not shown as the waveform is very similar to that obtained in Experiment 1). The waveform result of Experiment 4 in Figure 4.37 shows how the measured response completely misses the specified response in terms of its frequency and gain. Hence, it can be concluded that by having the bandpass check activated for the optimisation of bandpass filters, the optimisation algorithm converges to a <u>much better</u> or more importantly, <u>correct</u>, and in other cases, to a comparatively <u>similar</u> cost function value at a <u>faster rate</u>.

The results from this example also show the effect of using different performance models on the curve-fitting optimisation result. By concentrating the data points of the performance model in the frequency range of interest, as in Model 1, a more accurate result is obtained, especially within 0 to -3 dB attenuation range, compared to that of Model 2. From Table 4.10, it can be seen that the curve-fit error value, err_{total} for Experiments 1 and 2 of Model 1 are much lower than that of Experiments 3 and 4 of Model 2. This is confirmed by the AC waveform of the results. For a more specific comparison, this point is supported by the results of Experiment 1 and Experiment 3. It can be seen that for Experiments 1 and 3, the filter responses closely match the defined

performance model. However, the result from Experiment 1 is more accurate than that of Experiment 3. This fact can be seen clearer in Figure 4.41 and Figure 4.39 that shows the result of Experiments 1 and 3 using zoom factor B, where the frequency is between 0.5 and 2 G Hz, and magnitude is between 3 and – 20 dB. Figure 4.42 shows the response of Experiment 1 in more detail, where the frequency is between 0.9 and 1.3 G Hz, and the magnitude is between 1 and -5 dB.

Therefore, by concentrating the performance model specification around the passband as well as employing the bandpass checking mechanism, the optimisation algorithm is able to converge quickly at a solution that gives the most accuracy in the region of interest.

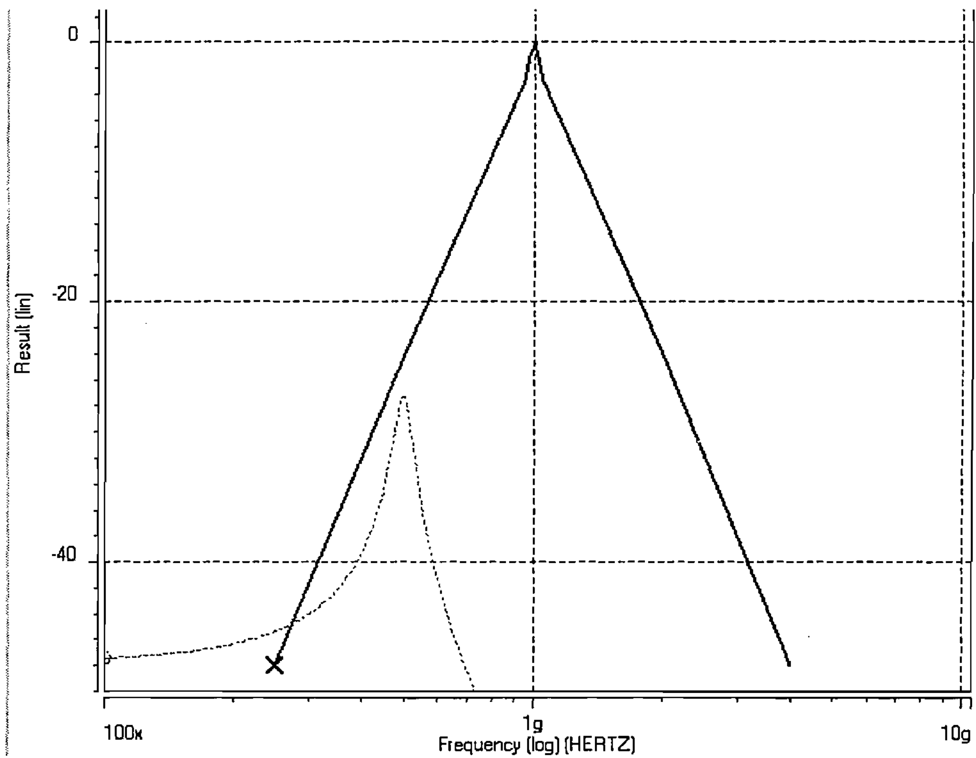


Figure 4.37 AC waveform result of Experiment 4 viewed using zoom factor A.

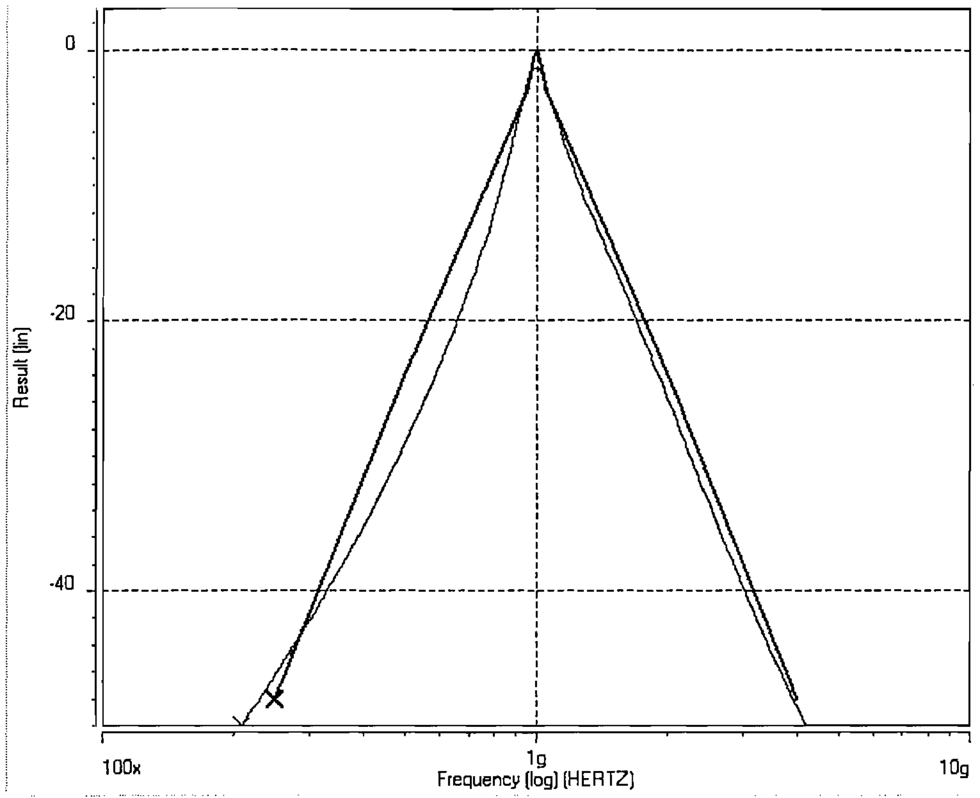


Figure 4.38 AC waveform result of Experiment 3 viewed using zoom factor A.

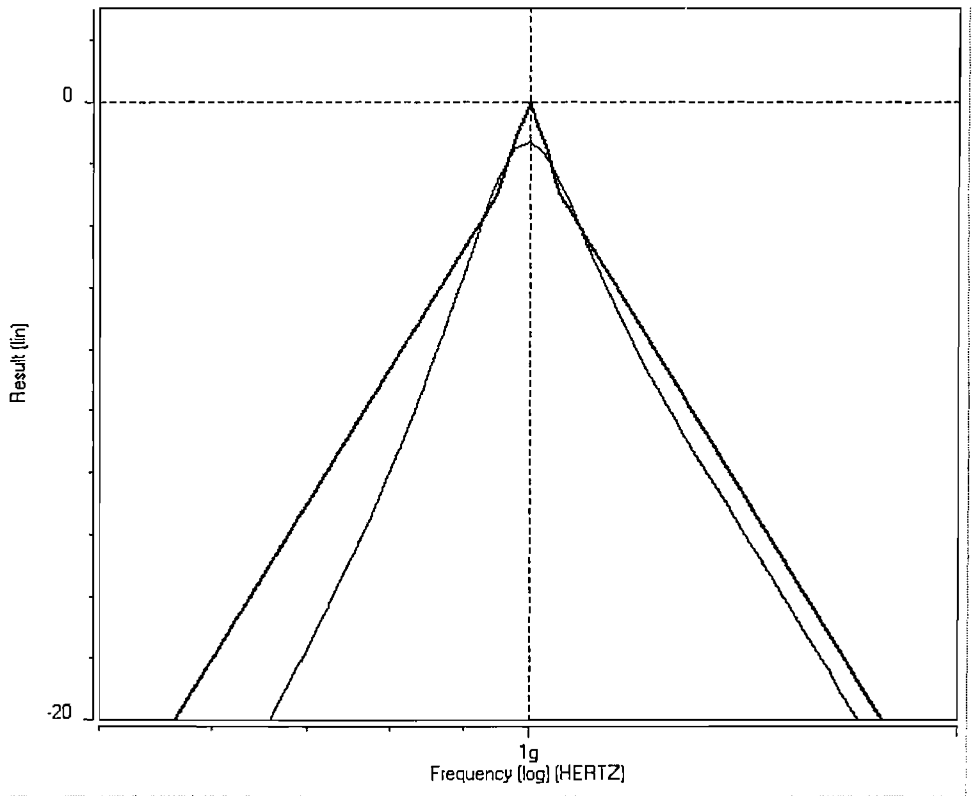


Figure 4.39 AC waveform result of Experiment 3 viewed using zoom factor B.

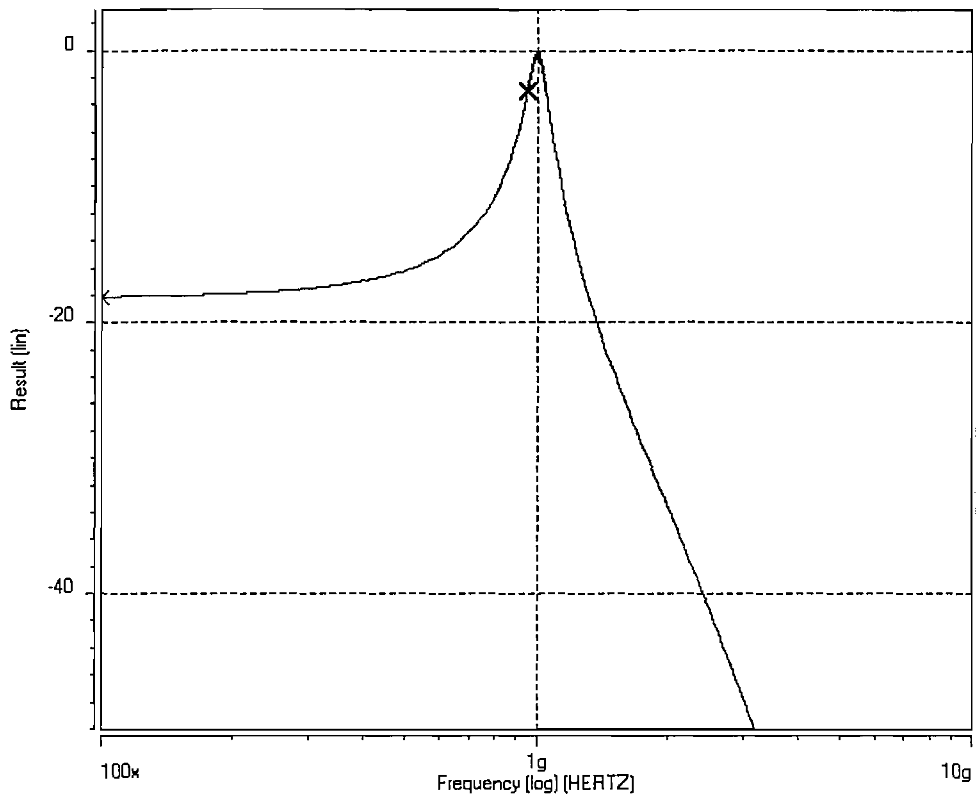


Figure 4.40 AC waveform result of Experiment 1 viewed using zoom factor A.

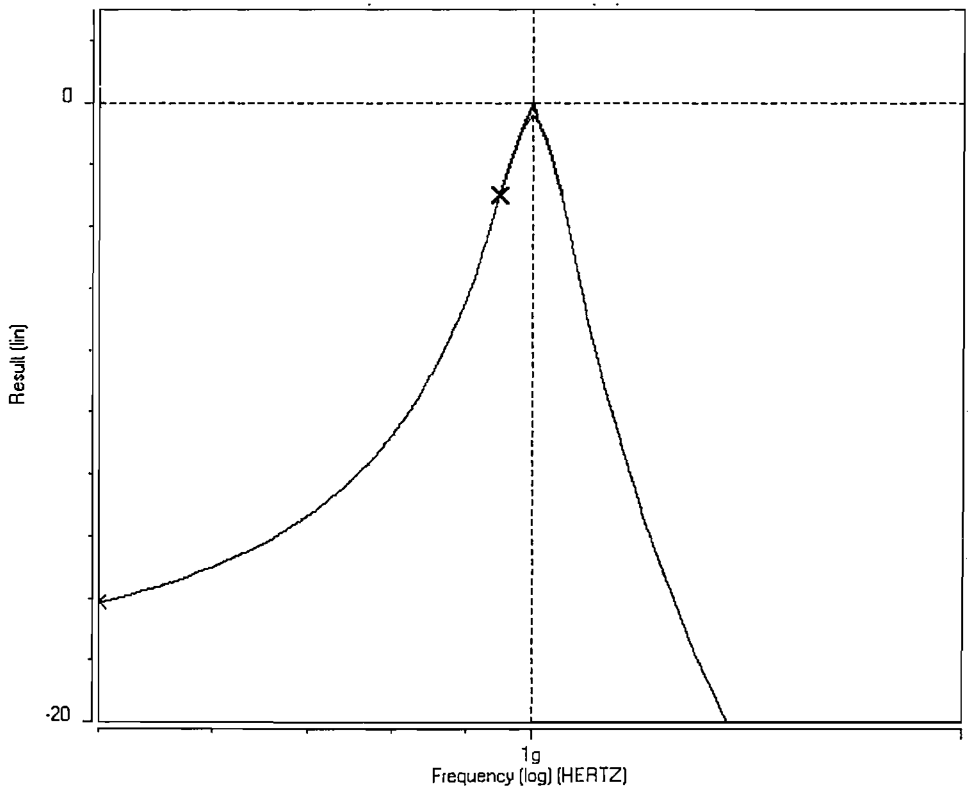


Figure 4.41 AC waveform result of Experiment 1 viewed using zoom factor B.

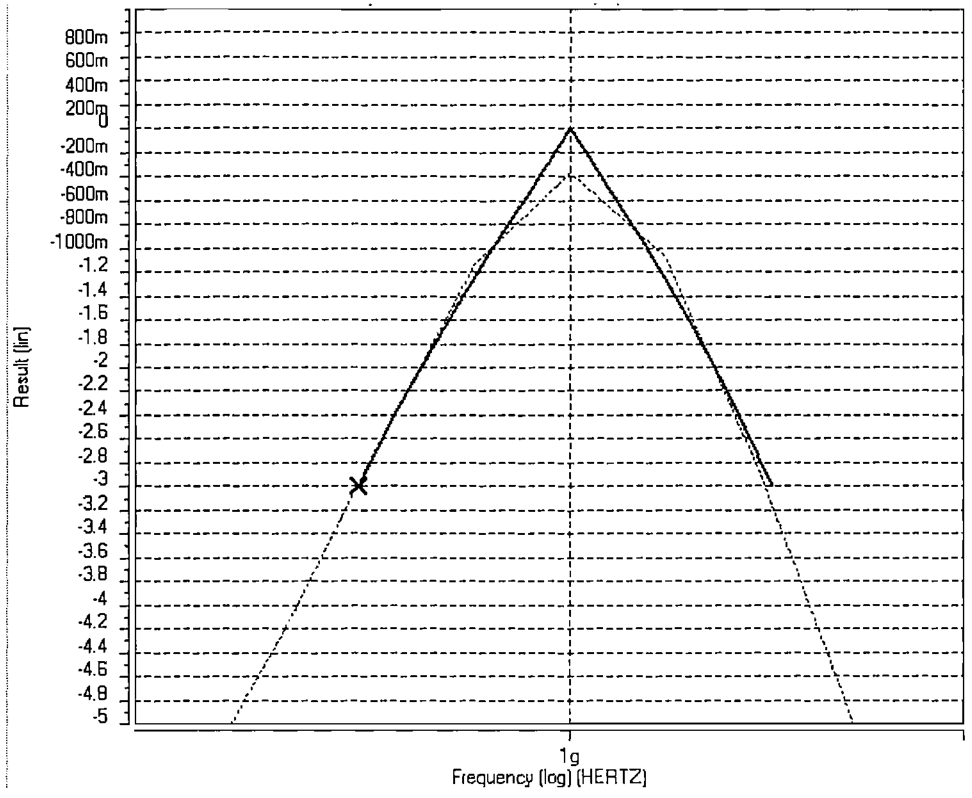


Figure 4.42 AC waveform result of Experiment 1 viewed using zoom factor C.

This bandpass error check is implemented by having HSPICE measurement statements, .MEASURE, in the filter netlist. Examples of such statements are shown in Figure 4.43. The output variable is the voltage of node VOUT4, and measurements are made for *fpeak*, *f1* and *f2* on the frequency response of the voltage of VOUT4, as indicated in the listing. The respective values for *fpeak*, *f1* and *f2* are then available in the HSPICE results file for processing by the optimiser.

```
.MEASURE AC MAXAMP MAX V(VOUT4) from=1 to=10G
.MEASURE AC FPEAK when V(VOUT4)='MAXAMP'
.MEASURE AC F1 when V(VOUT4)='MAXAMP*0.707945784' rise=1 fall=0
.MEASURE AC F2 when V(VOUT4)='MAXAMP*0.707945784' fall=1
```

Figure 4.43 Measurement statements to evaluate parameters required by the additional penalty check for bandpass filters.

4.4 Concluding remarks

The aim of this chapter is to explain the synthesis methodology developed in the course of this research to obtain optimised analogue filter netlists from VHDL-AMS behavioural specifications. The methodologies described in this chapter proceed from those outlined in the previous chapter,

Chapter 3, where in this chapter the filter information obtained from the analysis of VHDL-AMS parse trees are used the synthesis procedure.

After detailing the methodologies to obtain analogue filter hardware from filter information, the latter part of this chapter focuses on the topic of architectural and parametrical optimisation. Justifications for the optimisation's cost function formulation and discussion on the choice of performance model specification for the curve-fitting-based optimisation have been made. The inclusion of full HSPICE simulation to evaluate the performance of the candidate topologies makes this methodology particularly suitable for high frequency synthesis. The criteria used in the optimisation are based on the frequency response accuracy and power consumption.

The following chapter, Chapter 5, presents the optimisation engine which relies on the three-tier optimisation algorithm. Together with the cost function formulation and appropriate choice of performance specification, this algorithm is proven to produce accurate results as have already been shown in several examples in this chapter.

5 Three-tier Optimisation Algorithm

As explained in Chapter 4 above, the architecture evaluator calls the optimisation algorithm that performs parametric optimisation for each candidate filter topology. Candidate filter cells are obtained from a collection of HSPICE netlists of analogue filter topologies suitable for high-frequency integrated circuits. To increase the optimisation accuracy, and to ensure that a minimum is found, the parametric optimisation strategy is to employ a three-tier optimisation which is a combination of:

- Stochastic search.
- Downhill simplex algorithm [120].
- HSPICE's built-in optimisation engine.

Sections 5.1 to 5.3 explain the stochastic search, downhill simplex algorithm and HSPICE's optimiser in detail, and Section 5.4 describes the three-tier optimisation algorithm together with experimental results highlighting the advantage of the three-tier algorithm over its other constituent algorithms. Finally, the software implementation of the methods described in Chapters 3, 4 and 5 which are integrated into FIST (Filter Synthesis Tool) are presented in Section 5.5.

5.1 Stochastic search

The stochastic, or random search is an optimisation technique which is based upon the generation of a (set of) numerical circuit parameter value(s), within a predefined range(s). For example, the parametric optimisation of the Colpitts oscillator circuit in Case Study 1 (Section 4.3.2.1) involves the following parameters and their ranges (that are given in Table 4.3), reproduced below in Table 5.1. If, for example, it is specified for the parametric optimisation based on random search to have 3 independent iterations, or more accurately, 3 tries, such tries would produce 3 sets of parameters, such as those listed in the last three columns of Table 5.1.

Parameter	Minimum	Maximum	Set 1	Set 2	Set 3
k	0.01	0.99	0.56	0.08	0.64
W	0.5 μm	500 μm	67.88μm	4.56μm	102.44μm
I_{BIAS}	0.1 mA	10 mA	2.33mA	0.89mA	5.77mA

Table 5.1 The optimisation space for the parameters selected for optimisation in Case Study 1, as well as three sample parameter sets.

This is a very simple and straightforward optimisation method, as no intelligence is employed in determining the choice of the direction in the optimisation space. In the above table, Set 2 is independently generated without considering the outcome of the optimisation using parameters in Set 1, and the relationships between the subsequent sets are similarly independent. Consequently, the minimum of the optimisation cost function is found by chance and not by successive purposeful movement based on previous function evaluation or information gathered from evaluating previous points. It is also apparent that this is a costly procedure, in terms of its inefficiency as much of the unwanted or unnecessary information is rejected. However, this method has the ability to find the global minimum, albeit by accident, as opposed to the nearest local minimum given a sufficient number of maximum tries as well as an appropriate optimisation space. Also, this optimisation method is useful for the preliminary stages of parametric optimisation problems, and may be combined with other more powerful algorithms as will be discussed later.

5.2 Downhill simplex optimisation

The downhill simplex algorithm, or also known as the amoeba, is based around manipulation of a geometrical figure that forms a multi-dimensional simplex. This particular simplex method is claimed to be the best sequential optimisation method [121]. In an N-dimensional optimisation space where there are N parameters to be optimised, a simplex consists of N+1 vertices (points) including the interconnecting line segments and polygonal faces. The amoeba algorithm makes a series of steps called *reflection*, *expansion*, or *contraction*, aiming at moving the vertices downhill, until the simplex points converge to a minimum. For example, the basic movements for a tetrahedron, a four-node simplex (i.e. N = 3) are shown in Figure 5.1. The *reflection* aims to move the simplex away from the point that has the highest cost function, while for the direction where good progress is made, the *expansion* move is applied. *Contraction* is to pull the simplex towards the point where the cost function is the lowest, where the aim is to converge all the vertices of the simplex to a minimum.

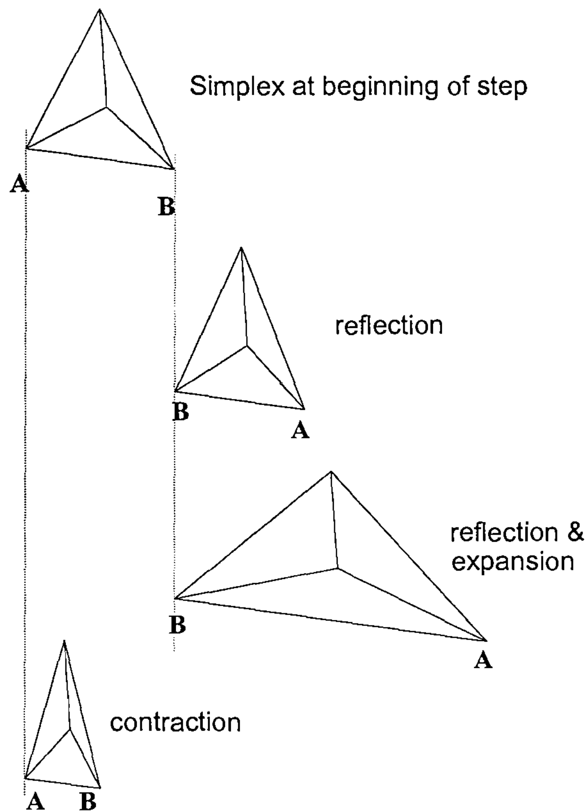


Figure 5.1 Basic movements of the simplex algorithm.

The application of the basic movements are indicated in the implemented function, *amoeba()*, as shown in the pseudo code in Figure 5.2 [109]. This function implements the downhill simplex algorithm being used by the three-tier algorithm, where it is called after the initialisation of the N+1 points has been made. The initialisation of the vertices of the simplex is made by randomly generating N+1 points, and then finding the cost function value of each vertex according to the cost function of Equation 4.14. This information is passed to the function *amoeba()*. The simplex is then moved according to the value of the cost function of its vertices.

When either of the termination criteria is met, the current position of the simplex and the respective cost functions as well as the number of iteration or function evaluation are returned. The *amoeba()* is terminated when the difference between the cost function of the vertices is smaller than the user-defined value, *ftol*. In this case, the algorithm has converged to a minimum. For the case that the downhill simplex is trapped in a meaningless direction, *amoeba()* terminates when the number of iterations or function evaluations exceeds a user-defined value.

amoeba()

Inputs:

1) number of variables: N,

2) N+1 parameter set: p_amb[N]

3) the corresponding CF: y_amb[N],

4) the tolerance to determine termination ftol,

5) a pointer to the evaluation function which is 3TOptFunc( ).

1. The points in the simplex are examined to determine the highest, y_amb[hi] and the lowest point, y_amb[lo].

2. The fractional range, rtol, from the highest to the lowest is computed.

3. If the termination criteria, or convergence test, is fulfilled, the outputs are listed and the procedure stops. Termination criteria:

(a) rtol < ftol, or

(b) maximum iteration value has been reached (each function evaluation is an iteration).

4. Else, the centroid is computed and a series of steps are taken to move the highest points through the opposite face of the simplex to the lowest point:

Reflection: extrapolate through the face of the simplex by factor ALPHA

If the result is better than the best point, i.e. lower than y_amb[lo]

An additional extrapolation by factor GAMMA is made.

If the reflected point is worse than the second-best point:

Contraction: by a factor of BETA, to look for an intermediate lower point

If the point is still worse or equal to the highest point:

Contract around the lowest (best) point

Repeat Steps 1 to 4 until terminate.

Three outputs:

1) new N+1 sets of converged p_amb[N]

2) converged y_amb[N]

3) number of function evaluations, nfunc.

Figure 5.2 Pseudo code for the downhill simplex algorithm, *amoeba()*, from [109].

The downhill simplex algorithm is noted for its robustness in its ability to converge to a minimum and being able to handle difficult objective function surfaces. Also, the optimisation of non-linear circuits involves complex multi-dimensional parameter spaces in which the geometrical naturalness of downhill simplex and the fact that it requires only function evaluations, not

derivatives, are particularly attractive. It is noted, however, that this simplicity is traded-off in terms of inefficiency of the number of function evaluation required [109], where the progress is slow especially when the algorithm converges, [121]. Nevertheless, the most common strategy is to employ this algorithm during the early stages of optimisation and then switching to gradient-based methods in the latter stages.

Another common issue with the downhill simplex algorithm is the starting points, where the user needs to specify N+1 starting points to start the algorithm. Nonetheless, in this research, the latter issue is viewed as an advantage of the algorithm, as the need to find a large number of starting points implies a relatively global search.

The effectiveness of the method, as well as the higher possibility for a convergence to a global minimum, makes this algorithm to be attractive in analogue circuit parametric optimisation. Similar methods have successfully been used in electrical circuit optimisation and many synthesis applications [4, 60, 122, 123].

5.3 HSPICE built-in optimiser

HSPICE has its own built-in optimisation engine that may be invoked to perform multi-parameter optimisation in AC simulations [119]. HSPICE uses a combination of the steepest descent method and Gauss-Newton to explore the search space. The optimisation algorithm used is Levenberg-Marquardt [109],[119] where the Marquardt scaling parameter is used to indicate the closeness of the solution to the optimum. The steepest descent algorithm is used when the current search is far from the solution, hence larger and rapid movements are made to find a better point. However, when the algorithm detects that the search is improving too slowly, the Gauss-Newton method is used. There are two types of object functions that can be used in HSPICE:

- *Curve-fitting*, where the optimiser aims to match the simulation results with user-defined data. The curve-fit error defining the difference between specification and actual measurement can be selected from several error functions defined by HSPICE.
- *Goal,* where goals for a particular electrical specification are defined in terms of HSPICE's .MEASURE statements, using a specific relational operator.

The HSPICE optimisation feature may be invoked by adding relevant command lines in the netlist. The key statements that must be present in the netlist are:

- .PARAM parameter=OPTxxx (init, min, max)
    - .PARAM statements specify the parameters to be optimised, their starting point values as well as minimum and maximum limits.
- .MODEL modename OPT…

- This statement defines the optimisation model which contains convergence and termination criteria, derivative methods and various accuracy control parameters.

- .DC | .AC | .TRAN <DATA=filename> SWEEP OPTIMIZE=OPTxxx RESULTS=ierr1 ... + ierrn MODEL=optmod
  - Analysis statement that includes the OPTIMIZE command for optimisation, as well as the name of the optimisation model (optmod), and the optimisation name (OPTxxx) that relates the parameters being optimised with the optimisation command.
- .DATA statement
  - This statement provides a list of numerical data for the desired curve used in curve-fitting optimisation.
- .MEASURE statements using the GOAL keyword.
  - .MEASURE statements are used in goal optimisation in which the .GOAL keyword defines the target electrical requirement to be measured and used in the evaluation of the cost function.

Figure 5.3 gives a sample of the HSPICE netlist for curve-fitting optimisation. For example in line (1), the parameter to be optimised is the transistor width of an OTA (operational transconductance amplifier). The initial value is 10μm, while its boundaries are set between 0.5μm and 500μm. The user may control the optimisation process by changing default control parameters defined in the optimisation model. In the example in Figure 5.3 line (2) shows that the optimisation model OPT1 has been selected with the maximum count of simulation runs 'itropt' set to 100. The default value is 20. Line (3) is the analysis statement for a curve fitting optimisation. The measurement to compare the simulated and ideal response is given by line (4) where the standard ERR1 error function has been selected. The desired curve data points are held in a text file (named cas.dat) and are read by the statements in the lines (3), (5) to (7), as shown in Figure 5.3.

```
.param WM1= OPT_OTA(10u, 0.5u, 500u) (1)
.MODEL OPT1 OPT itropt=100 (2)
.AC DATA=LOWPASS SWEEP OPTIMIZE=OPT_OTA RESULTS=COMP1 MODEL=OPT1 (3)
.MEAS AC COMP1 ERR1 par(magn) V(VOUT4) (4)
.DATA LOWPASS MERGE (5)
 + file=cas.dat freq=1 magn=2 (6)
.ENDDATA (7)
```

Figure 5.3 Example of HSPICE netlist for curve-fitting optimisation.

An example of goal-type optimisation is shown in Figure 5.4. In this case a lowpass filter circuit has seven parameters to be optimised; five are transistor widths (lines (1) to (5)) and two are

bias current sources (lines (6) and (7)). This is actually the vertically stacked regulated cascade topology introduced in Chapter 4 for Case Study 2. The goal of this optimisation is to obtain a lowpass cut-off frequency of 1 GHz, while at the same time limiting the overshoot to be lower than 3 dB. The midband gain is set to 1. This objective function is coded into the HSPICE netlist specification as shown in lines (8), (9) and (10).

```
.param WM1=OPTWI(10e-6, 0.5e-6, 15e-6) (1)
.param WM2=OPTWI(10e-6, 0.5e-6, 15e-6) (2)
.param WM3=OPTWI(10e-6, 5e-6, 15e-6) (3)
.param WM4=OPTWI(82e-6, 50e-6, 120e-6) (4)
.param WM5=OPTWI(1e-6, 0.5e-6, 10e-6) (5)
.param AMPIO1=OPTWI(40e-6, 5e-6, 80e-6) (6)
.param AMPIO3=OPTWI(8e-6, 1e-6, 30e-6) (7)
.measure ac midb_gain MAX I(Viout1) from=0Hz to=1e+6Hz goal=0.5A (8)
.measure ac threedb when I(Viout1)='0.707*midb_gain' goal=1GHz (9)
.measure ac ov_shoot MAX I(Viout1) from=100e+6Hz to 3e+9Hz
+goal='1.188*midb_gain' (10)
.model opt1 opt relin=1e-6 relout=1e-6 (11)
.ac dec 50 1 5G sweep optimize=OPTWI results=midb_gain,threedb,ov_shoot
+model=opt1 (12)
```

Figure 5.4 Section of HSPICE netlist implementing the built-in optimisation tool of HSPICE, for a GOAL-optimisation.

The results of HSPICE optimisation will be a summary stating the number of function evaluations, as well as the values of other parameters that indicate the accuracy of the optimisation process. A new set of parameter values, as well as the curve-fitting and measurement results are also given. For example, for the curve-fitting optimisation example of Figure 5.3, the value of 'comp1' indicating the curve-fit error value is returned. Whereas for the GOAL optimisation of Figure 5.4, the measurement values of 'midb_gain', 'threedb' and 'ov_shoot' are also returned by HSPICE.

HSPICE optimisation employs powerful gradient-based optimisation algorithm, which performs excellently when the initial values for the parameters are within the vicinity of a local minimum. However, for poorly chosen initial values, the optimisation fails to converge. Although there exist some optimisation options for HSPICE that can be used to control the optimisation, depending whether the starting points are near or far from a solution, such adjustment need to be done manually by the user. This dependency of HSPICE optimisation on the starting point values can easily be shown by analysing the earliest and the final stages of an optimisation example using the three-tier algorithm. This point will be demonstrated in the following section that describes the three-tier algorithm in detail.

5.4 The three-tier algorithm

The three-tier algorithm is the combination of the stochastic search, downhill simplex, and HSPICE curve-fitting algorithms, to make use of the best features of these algorithms. Although it is possible to use only HSPICE optimisation, or any of the described algorithms in the optimisation process, the combination of the three algorithms increases the accuracy as well as the efficiency of the optimisation procedure. Further in the text, the term 'one-tier' refers to any of the components that constitute the three-tier optimisation algorithm.

The operation of the three-tier algorithm can be summarised as follows. The stochastic search serves primarily as a tool to provide initial values for the downhill simplex algorithm. These initial points are passed to HSPICE optimisation, which will attempt to find the local minima for each set of points and returns the associated cost function value to the downhill simplex algorithm. Based on the cost function values, the downhill simplex will perform a series of movements aiming to converge the vertices of the simplex around the best point that has the lowest cost function value. By repeatedly restarting the downhill simplex, each time using a different set of initial values, the possibility of finding the global minimum is increased.

Thus, the strategy employed by the three-tier algorithm explores the local minima present on the hyper-surface of the analogue circuit optimisation cost function. The initial points generated stochastically are used by the simplex-based optimisation, which moves the initial points towards a solution that optimises the accuracy and power of the candidate design. The downhill simplex algorithm uses HSPICE's optimisation to evaluate the cost function, and hence, to obtain the set of parameter values that gives the closest fit to the specified frequency response curve.

The following subsections explain the three-tier algorithm in more detail. Section 5.4.1 firstly explains the use of HSPICE curve-fitting optimisation within the downhill simplex algorithm. After this has been clarified, the operation of the three-tier algorithm is demonstrated in Section 5.4.2, and finally, in Section 5.4.3, the performance of the three-tier algorithm against each one-tier algorithm is evaluated.

5.4.1 Downhill simplex function evaluation

The downhill simplex requires evaluations of the cost function, which is done by employing HSPICE optimisation. There are two distinctive occasions when an evaluation of the function is required. Firstly, an evaluation is required during the initialisation of the vertices of the simplex, before the downhill simplex algorithm begins its operation. As described previously in Section 5.2, this is done before the function *amoeba()* is called. The second occasion is during the operation of the downhill simplex algorithm itself, where the decision to perform the next operation depends on the cost function value of the vertex currently under consideration.

The basic operation of the cost function evaluator, *3TOptFunc()*, was shown in Figure 4.30 in Section 4.3.3. However, for the use of the function *3TOptFunc()* within the downhill simplex, a detailed implementation as given in the pseudo code of Figure 5.5 below is used. The two additional stages are shown as Step 1 and Step 5 of Figure 5.5.

3TOptFunc()

Input: a set of parameter's initial values.

1. Check the range of values for each parameter.

If OK, proceed to Step 2.

If NOT OK, assign a large value to CF, return this value and exit.

2. Construct a new netlist:

Prepare .PARAM statements with current parameter values.

Write newly constructed netlist to a file.

3. Launch HSPICE curve-fitting optimisation.

4. Read the results file and extract the following data:

Curve fitting optimisation results, i.e. the value of err_{total}

Power consumption

For a bandpass filter, the frequencies fpeak, f1 and f2

New values of the parameters

5. For downhill simplex initialisation, replace old parameter values with new ones.

6. Assign weighting penalty.

7. For a bandpass filter, compute the value of err_{bp}.

8, Compute CF value.

9. Log results.

Output: CF.

Figure 5.5 Pseudo code of the detailed cost function evaluator.

Step 1 is checking the parameter range. This process involves analysing the values of each parameter to ensure that the parameters passed to the cost function evaluator are within their predefined ranges. As the downhill simplex operations of reflection, expansion or contraction involve changing the values of the parameters, there is a possibility of the downhill simplex to calculate and thus assign values that are out of the specified range for any (or all) of the parameters. If this action is not contained, and the downhill simplex algorithm is further allowed to proceed into 'forbidden territories', there are two possible consequences. The first is due to the fact that the parameter values got out of range of the practical realisable values, and the second is due to the fact

that the downhill simplex sometimes uses a negative-valued parameter, which may happen when calculating a reflection or an expansion point.

For the first case, the cost function value that is returned may be acceptably small, and finally the downhill simplex will settle around and converge around this value, but the optimised parameters will not be acceptable as they are out of the predefined range. For the second case, the downhill simplex algorithm may collapse by running until completing the specified maximum iteration value and returning completely erratic cost function and parameter values. This is because unpractical parameter values of circuit netlists which are used to run HSPICE analyses may cause HSPICE to return unpredictable results.

For example, an OTA-C filter having 4 parameters under optimisation is used for the synthesis of a fourth-order lowpass filter at 1 GHz (see Case Study 3 in Chapter 6). The topology is optimised using the three-tier algorithm with one restart where the downhill simplex algorithm of the three-tier is not subjected to the 'sign test'. It is discovered that the algorithm got trapped at a 'wrong' point where one of the parameters have a negative value. The cost function evaluator keeps on returning an inconsistent value, causing the algorithm to terminate only when the maximum number of iteration of 1000 is reached. The point which the downhill simplex algorithm got trapped and converged to is shown in Table 5.2, where it can be seen that parameter 4 is the culprit, and the erratic values of the cost function of the vertices can be observed in the last column.

Vertex no.	Param. 1	Param. 2	Param. 3	Param.4	CF
1	8.724e-05	2.423e-06	8.999e-13	-3.750e-15	3.709e-03
2	8.724e-05	2.423e-06	8.999e-13	-3.750e-15	5.752
3	8.724e-05	2.423e-06	8.999e-13	-3.750e-15	3.418
4	8.724e-05	2.423e-06	8.999e-13	-3.7504e-15	5.412
5	8.724e-05	2.423e-06	8.999e-13	-3.750e-15	4.933

Table 5.2 The three-tier optimisation result for the case that out-of-range parameters goes unchecked.

Therefore, to safeguard against these effects, the very first step in the cost function evaluation process involves checking that the parameters are within the allowable range. If any of the parameters are detected to be out of the allowable range, a large value is assigned to the cost function, which is returned to the downhill simplex algorithm, and then the function *3TOptFunc( )* exits without completing the further steps.

Step 5 involves replacing the set of parameters that is passed to function *3TOptFunc()* with those that are newly found by HSPICE optimisation. This step will only affect the parameters that are selected during the initialisation of the downhill simplex, as will be explained in Section 5.4.2. For cost function evaluations during the operation of reflection, expansion or contraction, the new parameter values do not replace the original ones that are passed to *3TOptFunc()*. Obviously, this

step can be enabled or disabled, depending whether it is desirable to change the old parameters with new ones or not. The investigation to justify Step 5 will be explained in the following section, Section 5.4.2.

As a final point, there is another subtle implication of the formulation of the cost function CF (Equation 4.14) that includes both the curve-fit error as well as power consumption, which requires the cost function evaluator to be implemented *within* the downhill simplex algorithm. Note that this multi-objective optimisation for accuracy and power cannot be done by HSPICE optimisation alone, as this will involve different types of HSPICE analyses – AC analysis for frequency response, and operating point or transient analysis for power - using the same set of parameters. HSPICE can independently optimise the circuit for either accuracy or power, but not both at the same time. Although HSPICE optimisation is a reasonably powerful gradient-based optimisation algorithm, it is very dependant on the starting points of the parameters, also, it is unable to handle such formulation of the cost function as explained above. By embedding HSPICE optimisation within the downhill simplex algorithm, the downhill simplex provides the starting points for HSPICE optimisation that are influenced according to the cost function that includes both the accuracy and power. While HSPICE optimisation directly optimises the circuit to maximise the curve-fitting accuracy, information regarding power consumption of the candidate can be obtained from HSPICE optimisation results. Hence, using such arrangement between HSPICE optimisation and the downhill simplex as well as the formulation of the cost function as given in Equation 4.14, it is possible to find a set of solutions which are both accurate and exhibits a reasonably low power consumption for a particular topology.

5.4.2 Three-tier description

Figure 5.6 illustrates the block diagram of the architectural optimisation strategy that employs the three-tier parametric optimisation algorithm. The three-tier algorithm is implemented in the software module called *Three_tier_opt()*, indicated by the dark rectangular box in the diagram. The shaded right-hand part shows that the algorithm starts by initialising the stochastic search count to zero. Then, the N+1 sets of initial points for the downhill simplex is prepared by firstly randomly generating the parameters and then finding the cost function value for each N+1 sets of parameters. The left-hand-side of the shaded block shows that the downhill simplex, or the amoeba algorithm, is called repeatedly by the stochastic search, until the count reaches a predefined value, *max_stoch_count*. This means that the amocba algorithm is restarted using different initial values for *max_stoch_count* times.

By restarting the amoeba algorithm several times using different starting points, the probability of finding the global minimum, or at least a better local one, is increased, rather than those found by the one-tier optimisation algorithm, especially for HSPICE optimisation whose

performance is very sensitive to the starting point. Cost functions calculated from simulation results of analogue circuits tend to have many local minima, therefore to ensure that a good solution is found, the optimisation process needs to be restarted with different initial values generated by a random number generator of the stochastic search.

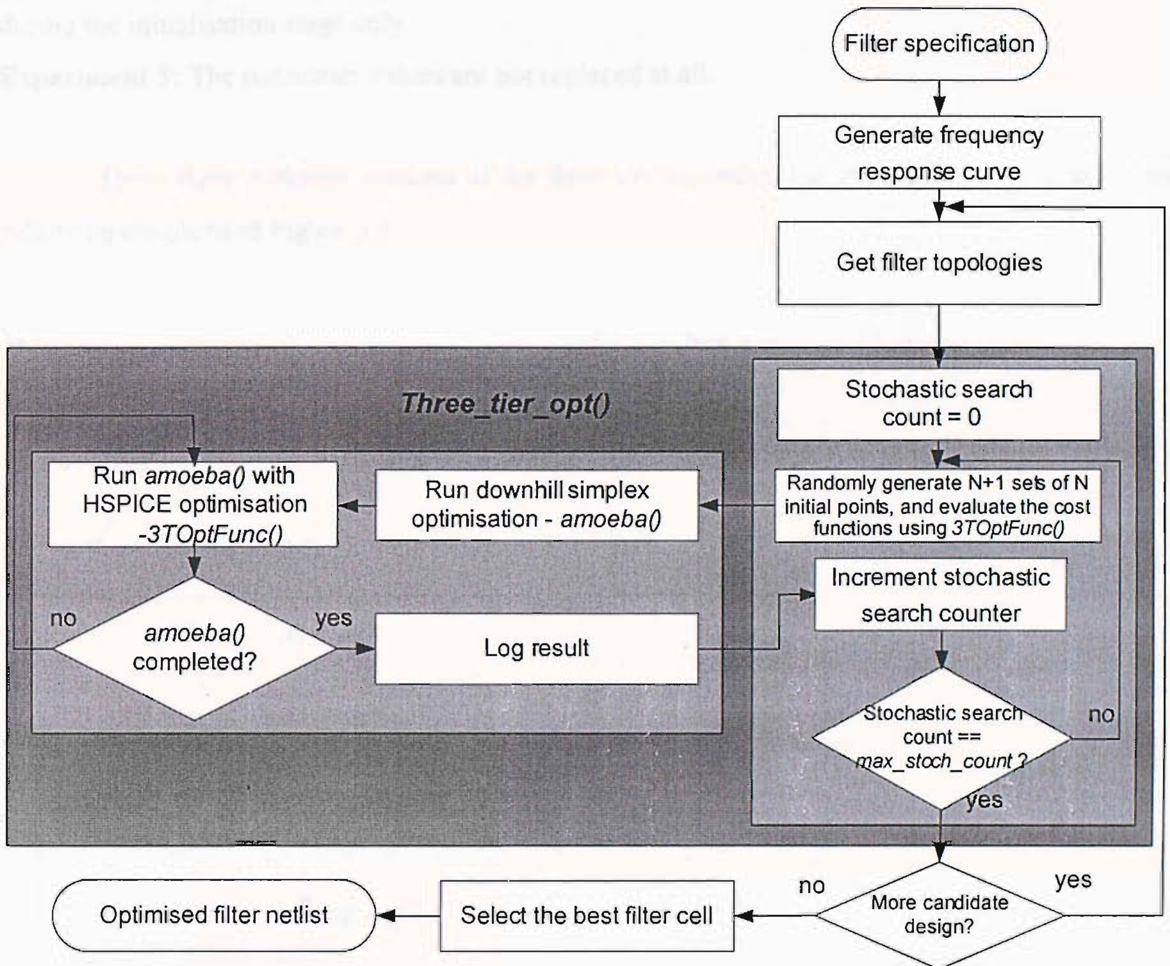


Figure 5.6 Three-tier parametric optimisation strategy.

The cost function evaluation is done by the function called *3TOptFunc()* as described before in Figure 5.5, which returns the value of the cost function CF (Equation 4.14). As previously described, the need to evaluate the cost function arises twice, first is for the preparation for the starting points for the amoeba algorithm itself. Secondly, is in the amoeba algorithm as shown in the shaded left-hand-side block of Figure 5.6, where in this case, the cost function evaluation which involves calling the curve-fitting algorithm of HSPICE is done repeatedly until the downhill simplex algorithm converges or terminates. As mentioned in Section 5.4.1, the cost function evaluator is able to replace the parameters passed to it with the new one found by HSPICE optimisation, indicated by Step 5 of the pseudo-code in Figure 5.5. The investigation to justify Step 5 is done with three experiments as follows:

Experiment 1: All parameter values are replaced during the initialisation step, as well as during the subsequent function evaluation calls during the simplex operation of reflection, expansion or contraction.

Experiment 2: Parameter values are changed to the new ones found by HSPICE optimisation during the initialisation stage only.

Experiment 3: The parameter values are not replaced at all.

These three different versions of the three-tier algorithm are explained with the aid of the following diagrams in Figure 5.7.

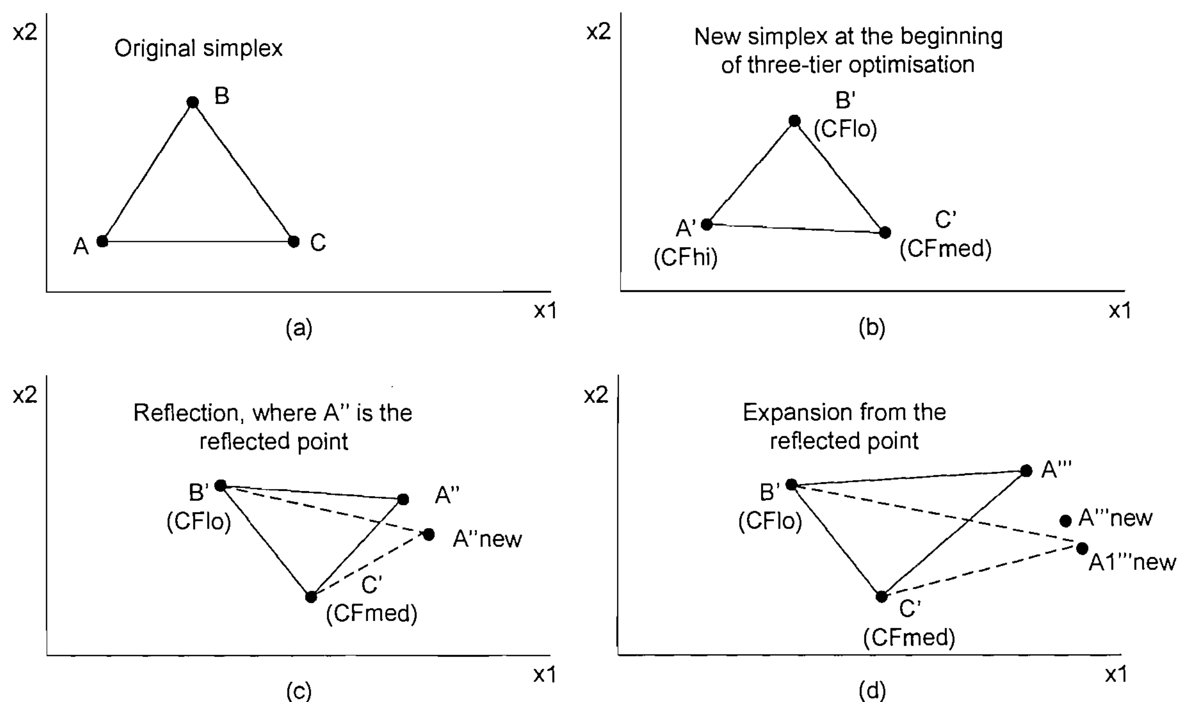


Figure 5.7 The operation of the different versions of the three-tier using a two-variable example.

Figure 5.7 shows an example of the operation of the three-tier algorithm on a two-parameter optimisation task, where N = 2. The stochastic search generates three nodes, marked as A, B and C in Figure 5.7(a), that form a triangle for the two-dimensional argument space. The cost function evaluator, *3TOptFunc()* is used to find the cost function for the three nodes, where it is found that node A has the highest cost function of CF_{hi}, while node B has the lowest cost function of CF_{lo}. The cost function value of node C is CF_{med}. The three cost function values are the local minima of nodes A, B and C. The cost function evaluator also returns new node values A', B' and C', corresponding to each local minima that is found from the points A, B and C respectively. Thus, in Experiment 1 and 2, the shape of the simplex is altered by the new nodes A', B' and C' that represents the local minima for nodes A, B and C. The new triangle is shown in Figure 5.7(b).

The reflection, expansion and contraction that operate on nodes A', B' and C' use the same procedure for the downhill simplex algorithm. However, the effect of changing the node to a new one, as is done in Experiment 1, or not changing it, as in Experiment 2, can be explained using Figure 5.7(c). In this diagram, it is shown how the simplex is reflected from node A' that has the highest cost function, to a new point, A''. The cost function value for node A'' is evaluated, where *3TOptFunc()* returns the local minima associated with A'', as well as a new position, A''_{new}. For Experiment 1, the reflected point is now changed from A'' to A''_{new}, while in Experiment 2, it remains at A''.

If good progress is being made, where the cost function of the reflected point A'' is better than the lowest, CF_{lo}, for Experiment 2, the simplex is expanded to node A''' as shown in Figure 5.7(d), and the cost function of the expanded node is found. For Experiment 1, this expansion is done from the new reflection point of A''_{new} to point A'''_{new}, however, after the cost function at A'''_{new} is found, the expanded point is changed to that of $A1'''_{new}$. For the case of Experiment 3 where no node replacement are made, the simplex begins at its original position of A, B and C of Figure 5.7(a) and proceed with reflection, expansion or contraction as in Experiment 2.

The three experiments are used to optimise an analogue filter topology called Cascade 1 (which will be detailed in Chapter 6) to realise a fourth-order lowpass response at 1 GHz. The results for the three-tier algorithm under the three experimental conditions are given below in Table 5.3.

Experiment	Iteration	CF	Power (mW)	Err_{total}
1	111	0.223	149.81	0.0736
2	274	0.128	93.072	0.0347
3	401	0.061	23.46	0.0377

Table 5.3 Experimental results for the enabling and disabling of parameter changes during the three-tier optimisation.

From the experimental results, it can be seen that Experiment 1 converged faster than the rest, however, its result is the worse. Although Experiment 3 is the slowest, it found the best result with the lowest value of the cost function. These three different versions of the three-tier implementation actually describe different ways to explore the surface of the optimisation cost function. In Experiment 1, where each iteration effectively changes the shape of the simplex, the reflection and expansion movement of the simplex is not truly being 'reflected' or 'expanded' as the centroid from which the movements are based upon keeps on changing. However, the process of contraction is speeded up, as the point of local minimum found by HSPICE optimisation is passed back to the downhill simplex algorithm during each iteration. Therefore, in this case, the three-tier is more influenced by the gradient-based optimisation algorithm of HSPICE, causing it to be easily trapped in a local minimum, as indicated by the results of Experiment 1 above.

On the other hand, in the case of Experiments 2 and 3, only the starting shape of the simplex differs. The subsequent movements of the simplex for both experiments are governed by the downhill simplex algorithm. Therefore, it is expected that the results for both Experiments 2 and 3 to be different because the initial simplex in both cases are different, therefore, the simplex in the two experiment explored different parts of the function surface. It is effectively like restarting the downhill simplex algorithm using different starting points. In this particular example, it so happen that the simplex in Experiment 3 converged to a better value than in Experiment 2, and took more iteration, although this is not necessary the case in other situation.

To conclude, the three-tier version of Experiment 1 is not a favourable implementation as the robustness of the downhill simplex algorithm is reduced. As both versions implemented in Experiments 2 and 3 are principally similar, it is decided to use the implementation of Experiment 2 where the initial simplex is changed.

Three_tier_opt( )

Input: an HSPICE netlist file

1. Open netlist file to read and check the parameter declarations in the file:

.PARAM paramname = optimname (initial, minimum, maximum)

2. Save paramname and optimname for each .PARAM statement.

3. The number of .PARAM statement is counted as N.

For all N parameters: each value for initial is computed randomly

3TOptFunc() is called to get the cost function CF for this set of N parameters

The initial value of the N parameters are replaced.

4. Step 3 is repeated N+1 times.

5. The downhill simplex algorithm, amoeba( ) is called.

6. The results from amoeba( ) is logged and analysed:

Amoeba( ) is restarted using a new set of initial points, where steps 3 to 6 will be repeated.

7. The program stops when amoeba has been restarted max_stoch_count number of times.

Output: max_stoch_count number of results files.

Figure 5.8 Pseudo code for the three-tier optimisation algorithm.

The pseudo code of the three-tier algorithm is shown in Figure 5.8. Steps 1 and 2 are carried out only once, and are driven by the type of analogue filter topologies that are predefined in the optimisation process. Steps 3 and 4 pre-compute initial points necessary to start *amoeba()* and, together with Steps 5 to 7, are repeated *max_stoch_count* times. The input to the program is a netlist file, while the output is a collection of results files. The results files contain the final values of the optimised parameters, simulation results obtained from HSPICE optimisations, as well as values of the respective cost functions.

An example that details the operation of the three-tier algorithm on a real optimisation example is presented as follows. The topology selected to implement a fourth-order lowpass Butterworth filter with cutoff frequency at 1 GHz has 8 parameters for optimisation. It is called Cascade 1, and will be detailed in Chapter 6. The stochastic search is used to generate the 9 initial points of the simplex. HSPICE optimisation is used within the function *3TOptfunc()* of Figure 5.5, which finally returns the following cost function values of Table 5.4 for each 9 points. It can be seen that the cost function values varies from 0.571 to 9.26.

Vertex no.	CF	Power (mW)	err_{total}
1	4.360	160	0.840
2	1.860	107	0.439
3	1.720	32.7	0.423
4	0.571	424	0.147
5	2.910	566	0.585
6	0.592	492	0.099
7	1.170	163	0.334
8	2.410	616	0.447
9	9.260	5.31	0.926

Table 5.4 The initialisation of the simplex for an 8-variable three-tier optimisation.

The three-tier optimisation converges after 283 iterations, where the algorithm finally converges to the cost function of 0.1278, with power consumption of 93.07 mW and curve-fit error value of 0.0347. The details of the last twenty iterations are given in Table 5.5.

The behaviour of the algorithm during the early stages and the final stages can be observed from Figure 5.9 and Figure 5.10. In Figure 5.9, the cost function value of the initial simplex that is given in Table 5.4 is plotted, while the cost function of the last 20 iterations are shown in Figure 5.10. During the initial stage, the cost function values are vastly different to each other, as they are due to the stochastic search and each vertex are generated independent to each other. However, when the downhill simplex is converging in the last twenty iterations, all vertices are near to each other, so that their cost function values are not ranging too far. It can be seen from Table 5.5 that the curve-fit error value is almost constant at 0.035.

The dependency of HSPICE optimisation on the parameter values supplied to it is also apparent, where in the very first stage of the three-tier optimisation, the parameter values are obtained from the stochastic search. During this initial stage, HSPICE optimisation returns large cost function values due to their random positions on the function surface. Whereas during the final stage, the parameter values passed to HSPICE for optimisation are due to the downhill simplex algorithm, and are positioned near the local minima of the surface of the function. Thus, in the end, the three-tier algorithm managed to find a cost function value which is about one-fifth of the lowest cost function of the initial point.

Iter.	CF	Power (mW)	err_{total}	Iter.	CF	Power (mW)	err_{total}
264	0.1319	97.10	0.0348	274	0.1294	94.59	0.0348
265	0.1321	97.37	0.0348	275	0.1294	94.68	0.0348
266	0.1130	95.32	0.0348	276	0.1289	94.15	0.0348
267	0.1132	97.03	0.0348	277	0.1280	93.27	0.0347
268	0.1288	94.05	0.0348	278	0.1278	93.07	0.0347
269	0.1290	94.18	0.0348	279	0.1335	98.69	0.0348
270	0.1285	93.73	0.0347	280	0.1349	100.11	0.0348
271	0.1368	101.92	0.0348	281	0.1292	94.41	0.0348
272	0.1297	94.96	0.0348	282	0.1331	98.27	0.0349
273	0.1305	95.76	0.0348	283	0.1283	93.52	0.0348

Table 5.5 The results of the last 20 iterations of the three-tier optimisation.

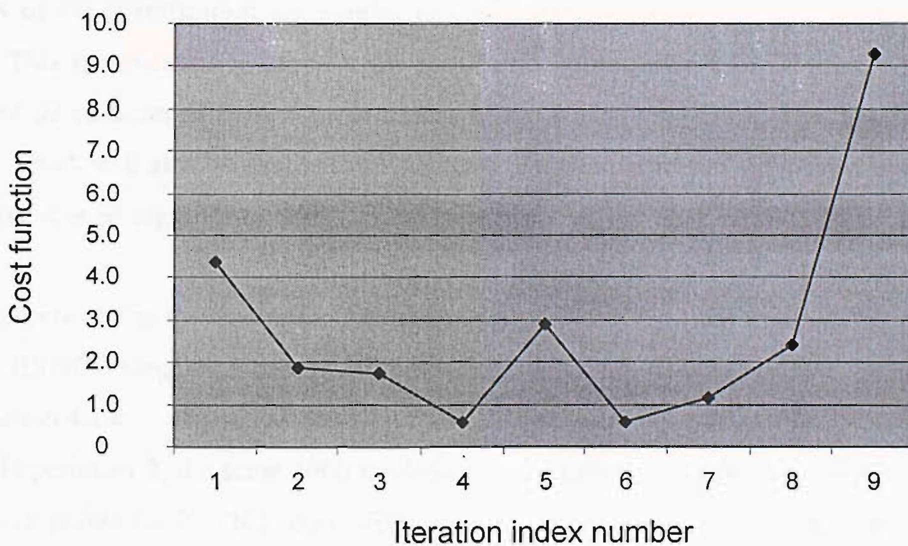


Figure 5.9 The cost function values of the vertices of the initial simplex.

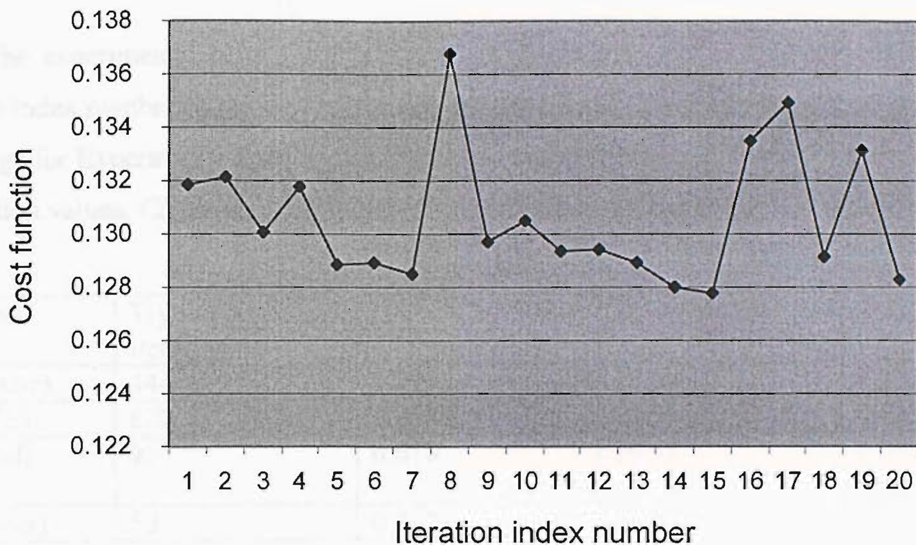


Figure 5.10 The cost function values of the last twenty iterations before the algorithm converges.

5.4.3 Three-tier optimisation versus one-tier algorithms

This section presents an example to compare the performance of the three-tier algorithm against the single usage of the stochastic search, downhill simplex algorithm and the built-in HSPICE optimisation. The analogue filter synthesis specification is for a fourth-order 1 GHz lowpass filter. The details of the specification are similar to Case Study 3 which will be further explained in Chapter 6. This specification is used for the parametric optimisation of an OTA-C filter topology implemented as cascades of second-order blocks of the wide-swing OTA. This topology is called Cascade 1, which will also be detailed in Chapter 6. Four experiments are carried out to compare the above-mentioned algorithms, where in each experiment, the cost function of Equation 4.14 is used.

Experiment 1 is the stochastic optimisation which runs for 1000 independent tries. In each try, normal HSPICE simulation and measurement commands are used to evaluate the cost function, CF, of Equation 4.14.

In Experiment 2, the same 1000 randomly-generated points used in Experiment 1 are used as the starting points for HSPICE curve-fitting optimisation. Similar to Experiment 1, each 1000 tries in this experiment are independent to each other, which mean that the result from one try does not affect the optimisation of the following one.

Experiment 3 uses the downhill simplex algorithm. The cost function of Equation 4.1.4 is evaluated using normal HSPICE simulation. Finally, Experiment 4 uses the three-tier algorithm. Both Experiments 3 and 4 are run once (i.e. not restarted) using the same random number generator, therefore the shape of the initial simplex in both experiments is the same.

The experimental results are presented in Table 5.6. The second column of Table 5.6 shows the index number of the best try for Experiments 1 and 2, and the number of iterations taken to converge for Experiments 3 and 4. The other remaining columns in Table 5.6 give the respective cost function values, *CF*, as well as the power consumption and curve-fit error figure, err_{total}.

Experiment	Try index no. / Iteration no.	*CF*	Power (mW)	err_{total}
1 (Stochastic)	444	1.143	27.64	0.372
2 (HSPICE)	820	0.223	86.75	0.137
3 (Downhill simplex)	95	0.610	20.18	0.295
4 (Three-tier)	52	0.227	90.88	0.136

Table 5.6 Experimental results comparing the performance of the three-tier algorithm with each one-tier algorithm.

From the experimental results, it is obvious that the three-tier algorithm is superior in terms of convergence rate compared to the other three algorithms. Also, the final cost function value that it converged to is very near to the one found using HSPICE optimisation, and can considered to be the same value, i.e. 0.2.

The performance shown in the above table is as expected, where it is expected that optimisation based on stochastic search will give the poorest cost function value. It is also expected that the HSPICE optimisation, which if given sufficient sets of starting points, may be able to finally give an excellent result, which is possibly a global minimum. However, this is at the costly expense of computational effort and time, as each try involves lengthy HSPICE optimisation, which alone, will take many iteration to finally converge to a local minima. Such waste is avoided by using improved algorithm such as the downhill simplex. As proven, the three-tier algorithm that combines the robustness of the downhill simplex algorithm with the powerful gradient-based HSPICE optimisation have the ability to quickly converge to a possibly global minimum. The probability of obtaining a global minimum of the cost function is increased by having several number of restarts using different starting points for the downhill simplex within the three-tier algorithm, as explained in Section 5.4.2.

5.5 FIST – Filter synthesis tool

This section presents the software implementation of the work done in the research. The methodologies presented in this chapter, as well as Chapters 3 and 4 have been implemented as an integrated filter synthesis tool, named FIST (Filter Synthesis Tool). In Chapter 3, the analysis of VHDL-AMS parse trees by the synthesis syntax checker and the static calculator to produce

analogue filter information have been presented. Then, in Chapter 4, the synthesis methodology that uses the analogue filter information together with a set of filter topologies to obtain an optimised circuit netlist of a filter have been detailed. Finally, in this chapter, the optimisation strategy is presented and discussed in depth. The integration and interaction of the methodologies in terms of software modules are further detailed in this section, and in short, is referred to as FIST.

FIST has been developed to run on two platforms: Windows and Unix. The Borland C++ Builder compiler was used in the development of the Windows version, while the Unix implementation relies on the SunOS g++ compiler. The software modules are created such that they can be easily ported between both platforms and compilers. The only exception is the Windows GUI which was developed specifically to drive the system on Windows machines. During the latter stage of the research that involved long runs of multiple HSPICE simulations, mainly the Unix version was used. The latest version of HSPICE being used in the research is version 2003.03.

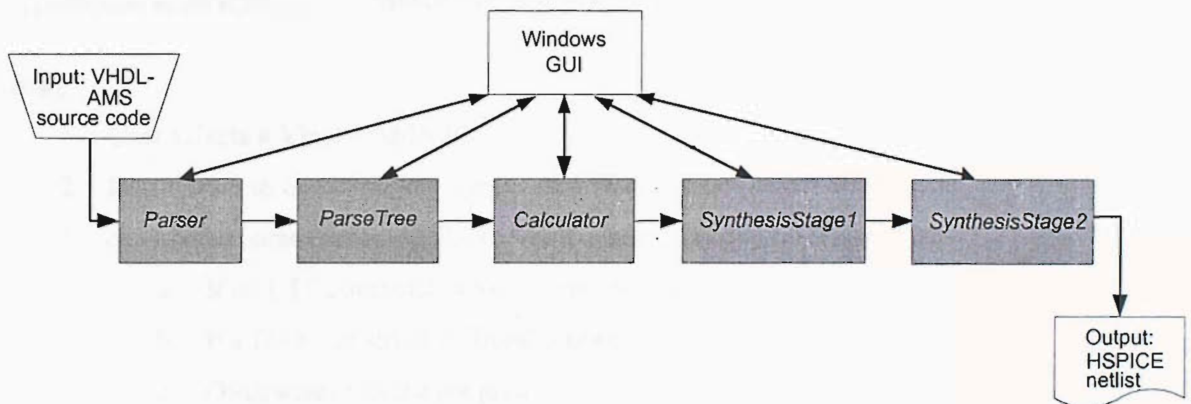


Figure 5.11 FIST structure.

Figure 5.11 shows the structure of the software modules that implement the methodologies presented in this chapter and the previous chapter. The shaded boxes show the five main modules. The basic functions of the modules can be described as follows:

Parser: VHDL-AMS compiler which produces a parse tree structure (Chapter 3).

ParseTree : Parse tree analyser which looks for synthesisable filter constructs (Chapter 3).

Calculator: Recursive static calculator which scans the parse tree to calculate coefficients for filter characteristics (Chapter 3).

SynthesisStage1: Initial synthesis stage which maps filter specifications specification to available filter cells (Chapter 4).

SynthesisStage2: Main synthesis stage which performs the architectural synthesis and three-tier parametric optimisation on the filter candidates (Chapters 4 and 5).

5.5.1 Windows GUI environment

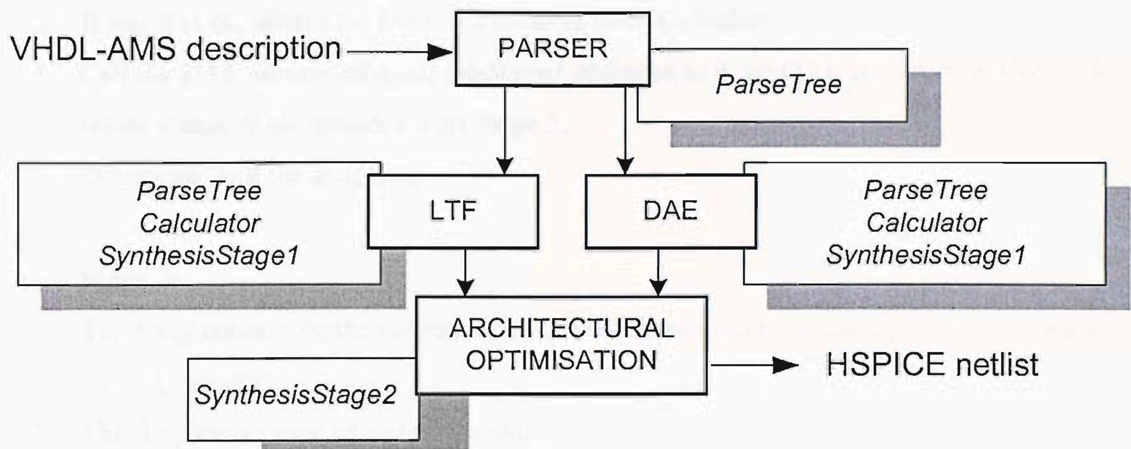


Figure 5.12 The interaction between the C++ modules in the GUI environment.

The structure of the main menu driven windows GUI module is shown in Figure 5.12. The typical operation scenario consists of the following 6 stages:

Stage 1:

1. User selects a VHDL-AMS file.
2. Run *Parser* to check for any syntax errors and create the parse tree.
3. Search the parse tree using *ParseTree* for synthesisable filter constructs.
    a. If an LTF construct is found, proceed to Stage 2a.
    b. If a DAE construct is found proceed to Stage 2b.
    c. Otherwise, exit the program.

Stage 2a:

1. Call the LTF version of *Calculator* and check the return status.
2. Exit if status is not ok.
3. If status is ok:
    a. Get the numerator polynomial coefficients and place them in an array.
    b. Find and store the numerator roots.
    c. Get the denominator polynomial coefficients and place them in an array.
    d. Find and store the denominator roots.
4. Call the LTF version of *SynthesisStage1* and check the return status. If ok, proceed with Stage 3.
5. Otherwise, exit the program.

Stage 2b:

1. Call the DAE version of *Calculator* and check the return status.

2. Exit if status is not ok.
3. If status is ok, obtain the DAE coefficients from *Calculator*.
4. Call the DAE version of *SynthesisStage1* and pass to it the DAE coefficients. Check the return status. If ok, proceed with Stage 5.
5. Otherwise, exit the program.

Stage 3:

1. The array containing the numerator and denominator coefficients is passed to the ideal curve generator.
2. The frequency range of interest is obtained.
3. The ideal curve is generated by calculating the magnitude of the transfer function at each frequency point.
4. The frequency and magnitude data is passed to the user.

Stage 4:

1. User selects the data points within passband as well as stopband.
2. The data points where column 1 is for the frequency and column 2 is for the magnitude are saved in a file with extension .DAT.
3. The performance model is passed to *SynthesisStage2*

Stage 5:

1. User inspects the file containing the list of available topologies and then saves the identifier of the selected topologies in the selected topologies file.
2. The selected topologies are passed to *SynthesisStage2*.

Stage 6:

1. In *SynthesisStage2*, the architectural and parametric optimiser is called.
2. Three-tier optimisation is set up by defining the number of restarts, i.e. *max_stoch_count*, as described in Chapter 4.
3. The three-tier optimisation algorithm is performed on each topology.
4. The architecture evaluator analyses the results file for each topology.
5. The topology with the lowest cost factor is selected, and its details are returned to the user.

5.6 Concluding remarks

A novel algorithm which features the best of the existing algorithms of stochastic search, downhill simplex and HSPICE optimisation has been detailed in this chapter. It has been shown that the performance of the three-tier algorithm is superior than each one-tier optimisation, in which it is able to converge to a possibly global minimum with minimal computational effort and time. The three-tier algorithm is also easy to implement and use, as the user does not need to tweak parameters in the algorithm, as is the case with some other heuristic-type algorithms. Even when the default settings for HSPICE optimisation are used, the three-tier algorithm is able to perform excellently. As long as the performance model of the ideal curve, as well as the analogue filter topology and its optimised parameters are appropriately chosen, the three-tier optimisation will be able to give a satisfactory solution.

The implementation of the three-tier algorithm employs a new and effective approach to the downhill simplex algorithm. It is usual to combine the downhill simplex with other gradient-based optimisation algorithm where the downhill simplex is used during the earlier stages and the gradient-based is use during the final stages of optimisation. However, in this case, the gradient-based methods of the HSPICE optimisation is used for function evaluation within the operation of the downhill simplex algorithm. This proved to be very effective as the HSPICE optimisation keeps on returning the local minima to the downhill simplex, so that, effectively, the optimisation cost function surface is thoroughly explored by the three-tier algorithm. Hence, the probability of encountering the global minimum is very much increased.

The last part of this chapter also presents the platform in which all the methodologies are integrated. From the description of the procedures involved in the high-level synthesis process, it is apparent that FIST is very easy to use and requires minimal interaction and knowledge from the user.

6 Practical Experiments with Two Case Studies

The process of architectural synthesis from VHDL-AMS parse trees and the three-tier parametric optimisation algorithm have been explained in Chapter 4 and Chapter 5. The aim of this chapter is to demonstrate the practical application of the presented synthesis methodology to two case studies. Case Study 3 is a fourth-order 1 GHz lowpass filter, while Case Study 4 is a fourth-order 1 GHz bandpass filter with Q factor of 10. Both filters are behaviourally specified using their transfer functions. Filter topologies that are suitable for integrated high-frequency applications are introduced in each case study, where the parametric optimisation issues of the topologies are also detailed. Figure 6.1 shows the steps taken in the architectural synthesis process. The dark boxes in the shaded area represent the repetitive process which is performed until all the filter topologies in the selected list have been parametrically optimised.

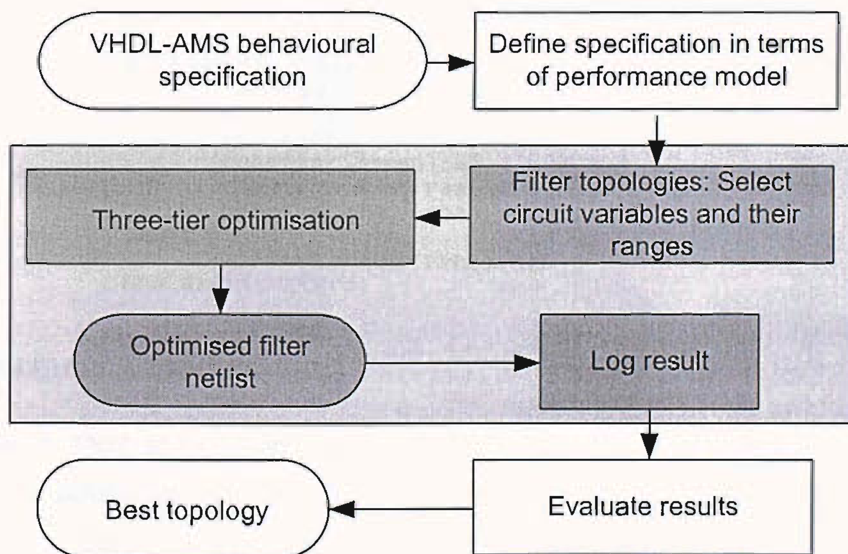


Figure 6.1 The architectural synthesis flow.

For both Case Studies 3 and 4, the material in this chapter is organised as follows:

- The performance model.

- The filter cell candidates - each candidate's circuit parameters to be optimised, their ranges and other constraints.
- The set-up for the three-tier parametric optimisation algorithm.
- Detailed results for the parametric optimisation of each topology.
- The final result of the architectural optimisation process.

Case Study 3 is detailed in Section 6.1, while Case Study 4 is in Section 6.2. Then, in Section 6.3, three application examples of one of the topology introduced in Case Study 3 are presented.

6.1 Case Study 3: Synthesis of an analogue fourth-order 1GHz lowpass filter

This example presents the synthesis of a fourth-order lowpass Butterworth filter with a cut-off frequency of 1GHz. The VHDL-AMS code for this filter is shown below. The desired transfer function is obtained by denormalising the polynomial coefficients for a fourth-order normalised filter obtained from filter tables (Table 8.1 in [108]).

```
entity filter is
      port (quantity Vin: real;
              quantity Vout: out real);
end entity filter;

architecture transfer of filter is
      constant a:real:=4.1589e-10;
      constant b:real:=8.6483e-20;
      constant c:real:=1.0535e-29;
      constant d:real:=6.4162e-40;
      constant num: real_vector:= (1);
      constant den: real_vector:= (1,a,b,c,d);
begin
      Vout == Vin'LTF(num,den);
end architecture;
```

6.1.1 Performance models

For Case Study 3, three models were initially investigated to represent the ideal curve in the architectural optimisation process. These are named as Model 1, Model 2 and Model 3, and are detailed below. The main differences between them concern the number of points used and hence the accuracy of representation. The purpose of the investigation was to select a model that can represent the ideal curve accurately, while consuming relatively low CPU time. This investigation is necessary to reduce the overall CPU time because, as detailed in Chapter 5, the three-tier optimisation will be restarted several times and also, while the downhill simplex algorithm is generally regarded to be a very robust method of multi-variable minimisation, it tends to converge slowly.

6.1.1.1 Model 1

Model 1 represents the ideal curve with 6 points obtained by calculation. The 6 points are located in the critical frequency region between 0.13GHz to 4.3GHz, and is plotted in Figure 6.2. The normalised magnitude axis represents the output voltage or current.

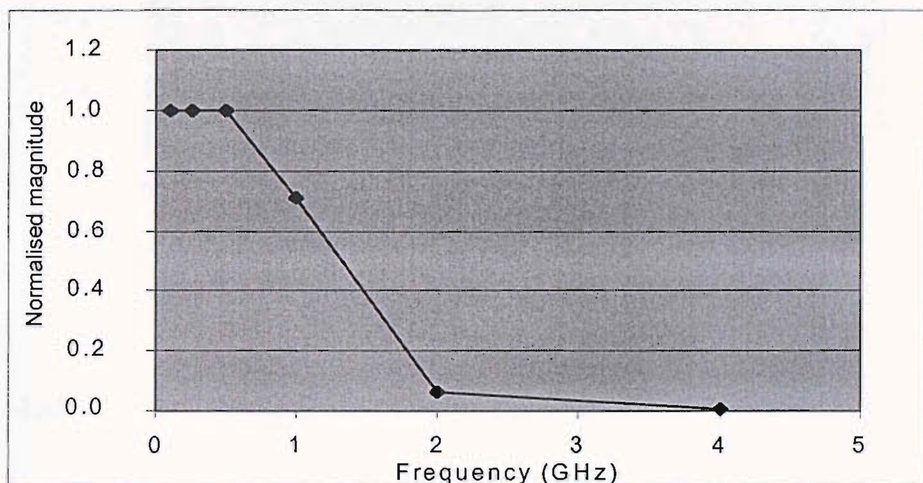


Figure 6.2 Performance model, Model 1, for Case Study 3.

6.1.1.2 Model 2

Model 2 represents 133 points that are located in the critical frequency region between 0.5GHz to 4GHz., and are shown in Figure 6.3. The magnitude span is between approximately 0 to –48 dB. These data are obtained by automated calculation of the magnitude of the transfer function defined by the VHDL-AMS code for Case Study 3. This method of building the performance model's ideal curve from the given transfer function in VHDL-AMS was explained in Chapter 4.

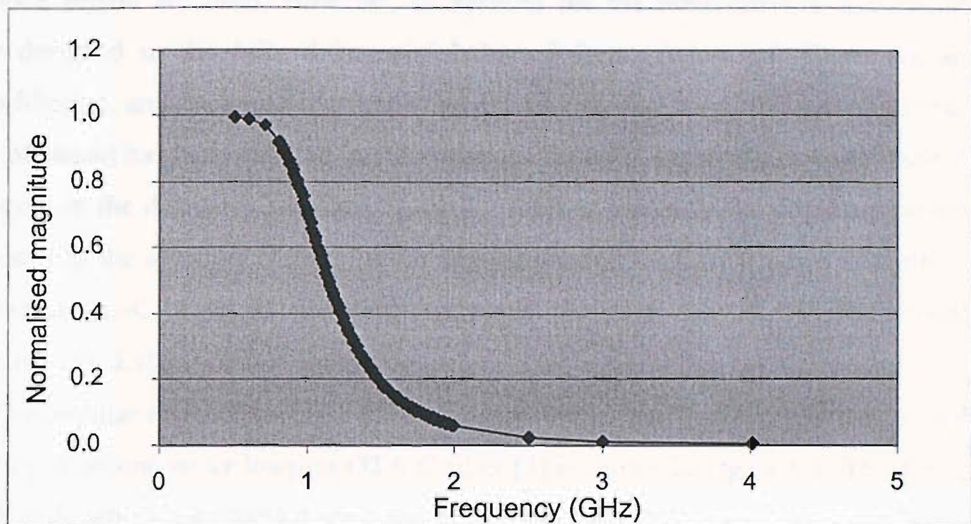


Figure 6.3 Performance model, Model 2, for Case Study 3.

6.1.1.3 Model 3

In Model 3, 36 points were selected from the 133 data points that are used in Model 2, and the corresponding curve is shown below in Figure 6.4.

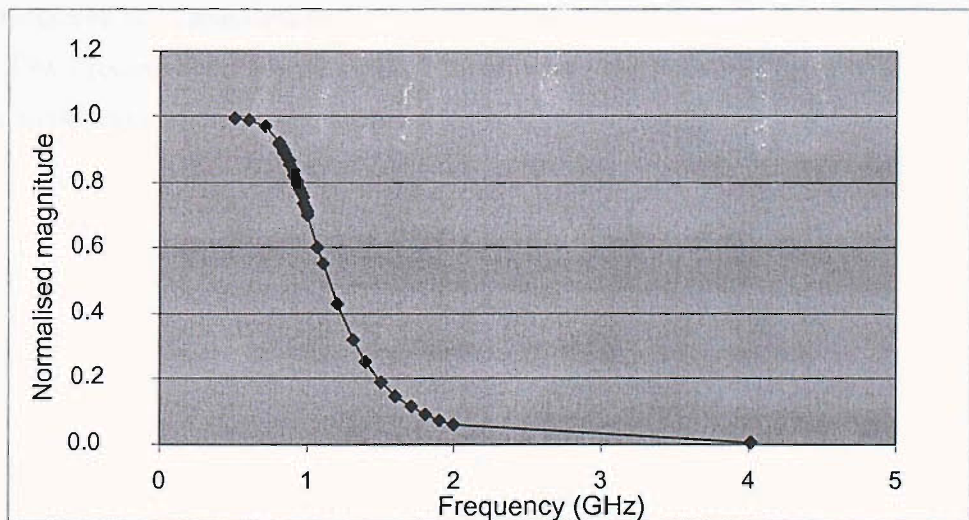


Figure 6.4 Performance model, Model 3, for Case Study 3.

6.1.2 Fourth-order lowpass analogue filter candidates

There exist several types of filters that are suitable for continuous-time, integrated and high frequency applications, the most popular being the OTA-C (Operational Transconductance Amplifier-Capacitor), MOSFET-C, and active LC filter [124]. MOSFET-C filters are very similar to active RC circuits, where the use of resistors is avoided and instead MOSFETs biased in the non-

linear triode region are used. However, to account for the nonlinearities, the circuit must be carefully designed in the fully differential balanced form. Active LC filters are specific for bandpass filtering, and the inductor is implemented on-chip as a spiral inductor. The discussion for this type of circuit has been detailed in the previous chapter (Chapter 4) in Case Study 1, and will be continued in the discourse for Case Study 4. As it is currently the dominant circuit for such applications [2], the topology chosen for the implementation for Case Study 3 is of the OTA-C (or also known as g_m-C) type. In this case study and the next, the OTA-C filter topologies are implemented in a 0.35μm CMOS technology.

The popular method to obtain a fourth-order design is to cascade two second-order biquad sections [2]. A second-order lowpass OTA-C filter [3] is shown in Figure 6.5. The circuit contains two OTA cells which are labelled gm1 and gm2. The OTA cell is a building block for the filter topology that may be implemented using various circuits. A simple OTA cell is shown in Figure 6.6, which is a single-ended OTA consisting of 5 transistors.

The transfer function $H_{LP}(s)$ of the biquad filter in Figure 6.5 is given by (6.1), where the transconductance value for both OTA are equal (in Figure 6.5, gm1 = gm2 = g_m), and the capacitances are labelled as C1 and C2. With this assumption, the Q factor, Q, and the natural frequency, ω_0, of the filter are given by (6.2) and (6.3) respectively. It can be seen that both the requirements for frequency and Q factor can be achieved by adjusting the values of the transconductances and capacitances in the biquad lowpass filter circuit. Figure 6.7 (a) shows the cascade of two second-order lowpass filter blocks, where the output of the first block is directly connected to the input of the second block.

$$H_{LP}(s) = \frac{g_m^{\ 2}}{s^2 C_1 C_2 + s C_1 g_m + g_m^{\ 2}} \tag{6.1}$$

$$Q = \sqrt{\frac{C_2}{C_1}} \tag{6.2}$$

$$\omega_0 = \frac{g_m}{\sqrt{C_1 C_2}} \tag{6.3}$$

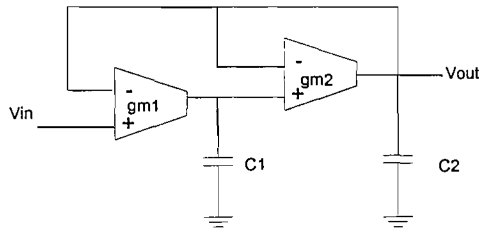


Figure 6.5 A lowpass biquad filter cell implemented using two OTAs and capacitors.

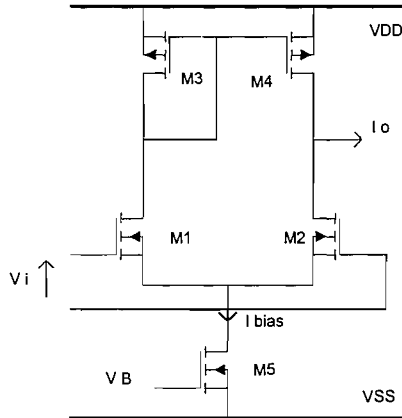


Figure 6.6 A single-ended OTA.

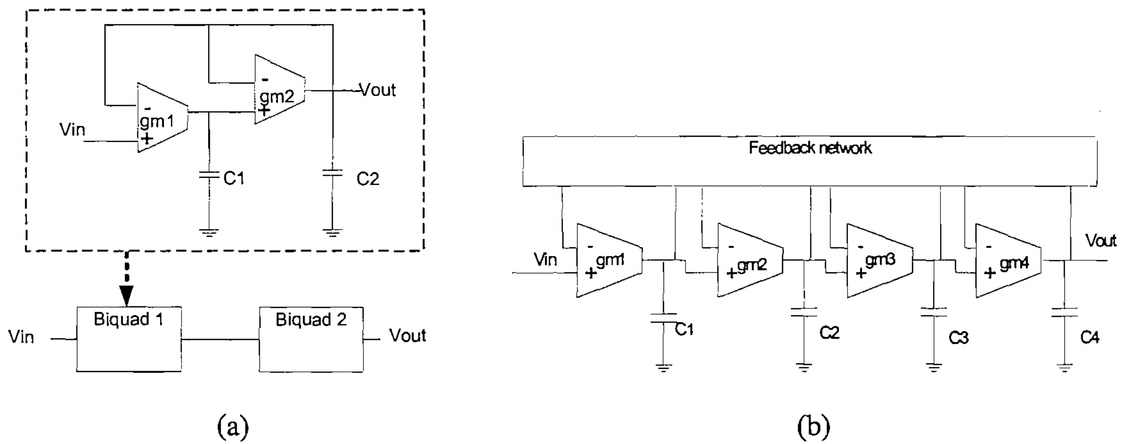


Figure 6.7 Implementation of a fourth-order lowpass filter. (a) Using two cascaded sections of lowpass biquads of Figure 6.5. (b) Using an all-pole multiple loop feedback structure.

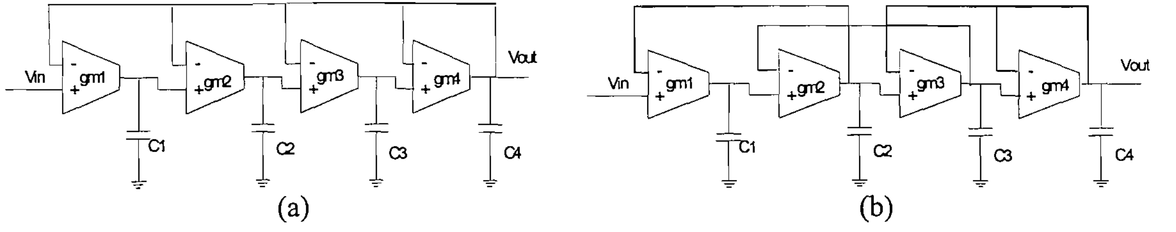


Figure 6.8 Fourth-order all-pole filter (a) IFLF (b) LF.

Apart from cascade, another method to implement a fourth-order lowpass OTA-C filter is to use a multiple loop feedback (all-pole) structure [124] as shown in Figure 6.7 (b). The number of the OTA-C integrator connected in the feedback network gives the filter order. There are two main feedback configurations that can be applied in the feedback network; the first one is called inverse follow-the-leader feedback (IFLF) and the other is the leapfrog (LF) configuration. Both are shown respectively in Figure 6.8 (a) and (b). The values of the capacitors C1 to C4 in the diagrams are obtained by comparing the respective transfer functions for each configuration ((6.5) and (6.6)) against a unity-gain fourth-order all-pole lowpass transfer function (6.4). Since the time constant τ of the integrator is related to the capacitance C and transconductance g_m as $\tau = C/g_m$, the capacitances C1 to C4 can be obtained by solving the formula in (6.7) and (6.8) for τ_1 to τ_4 for the values of IFLF and LF, respectively.

$$H(s) = \frac{1}{B_4 s^4 + B_3 s^3 + B_2 s^2 + B_1 s + 1} \tag{6.4}$$

$$H(s)_{IFLF} = \frac{1}{\tau_1\tau_2\tau_3\tau_4 s^4 + \tau_1\tau_2\tau_3 s^3 + \tau_1\tau_2 s^2 + \tau_1 s + 1} \tag{6.5}$$

$$H(s)_{LF} = \frac{1}{\tau_1\tau_2\tau_3\tau_4 s^4 + \tau_1\tau_2\tau_3 s^3 + (\tau_1\tau_2 + \tau_1\tau_4 + \tau_3\tau_4) s^2 + (\tau_1 + \tau_3) s + 1} \tag{6.6}$$

$$\tau_1 = B_1, \tau_2 = \frac{B_2}{B_1}, \tau_3 = \frac{B_3}{B_2}, \tau_4 = \frac{B_4}{B_3} \tag{6.7}$$

$$\tau_1 = B_1 - \frac{B_3}{B}, \tau_2 = \frac{B}{B_1 - \frac{B_3}{B}}, \tau_3 = \frac{B_3}{B}, \tau_4 = \frac{B_4}{B_3}, B = B_2 - \frac{B_1 B_4}{B_3} \tag{6.8}$$

Based on the previous explanation, for the synthesis of the specifications of Case Study 3, three different architectures of the OTA-C filter will be used:

- Cascade of second-order lowpass OTA-C filter blocks.

- All-pole multiple loop feedback in the inverse follow-the-leader feedback (IFLF) configuration.
- All-pole multiple loop feedback leapfrog (LF) structure.

It is obvious that the main building block of the three configurations of OTA-C filter is the OTA cell itself. The OTA or operational transconductance amplifier may be implemented using several configurations, which are widely available in books and publications such as [3, 125, 126]. In this synthesis design, three different OTA circuits [3] were used to implement the OTA or 'gm block' shown in Figure 6.5. The three OTA circuits are:

- The wide-swing OTA of Figure 6.9.
- The folded-cascode OTA of Figure 6.10.
- The wide-swing folded-cascode OTA of Figure 6.11.

Figure 6.9 Wide-swing OTA.

Figure 6.10 Folded-cascode OTA.

Figure 6.11 Wide-swing folded-cascode OTA.

Thus there are nine different circuits that implement the fourth-order lowpass filter, where each will be parametrically optimised using the three-tier algorithm. The details of each circuit are summarised in Table 6.1.

Name	Description	Circuit variables	Size
Cascade 1	Wide-swing OTA-C cascade	8 (all transistor widths, bias current, capacitor C1 and C2 for biquads 1 and 2)	48 MOSFETs, 4 capacitors
Cascade 2	Folded-cascode OTA-C cascade	8 (all transistor widths, bias current, capacitor C1 and C2 for biquads 1 and 2)	48 MOSFETs, 4 capacitors
Cascade 3	Wide-swing folded-cascode OTA-C cascade	8 (all transistor widths, bias current, capacitor C1 and C2 for biquads 1 and 2)	74 MOSFETs, 4 capacitors
LF 1	Wide-swing OTA-C leapfrog	6 (all transistor widths, bias current, capacitors C1 to C4)	48 MOSFETs, 4 capacitors
LF 2	Folded-cascode OTA-C leapfrog	6 (all transistor widths, bias current, capacitors C1 to C4)	48 MOSFETs, 4 capacitors
LF 3	Wide-swing folded-cascode OTA-C leapfrog	6 (all transistor widths, bias current, capacitors C1 to C4)	74 MOSFETs, 4 capacitors
IFLF 1	Wide-swing OTA-C inverse follow-the-leader feedback	6 (all transistor widths, bias current, capacitors C1 to C4)	48 MOSFETs, 4 capacitors
IFLF 2	Folded-cascode OTA-C inverse follow-the-leader feedback	6 (all transistor widths, bias current, capacitors C1 to C4)	48 MOSFETs, 4 capacitors
IFLF 3	Wide-swing folded-cascode OTA-C inverse follow-the-leader feedback	6 (all transistor widths, bias current, capacitors C1 to C4)	74 MOSFETs, 4 capacitors

Table 6.1 The summary of the topologies used in the architectural synthesis of Case Study 3.

6.1.3 Architectural optimisation for Case Study 3

The nine topologies introduced in the previous section consist of four OTA cells so that a fourth-order response can be realised. All four OTA cells are required to produce a unified transonductance value. This is achieved by having equal widths for all the transistors, and by biasing each OTA cell with the same amount of current. The transconductance g_m is related to the bias current and transistor sizes as follows:

$$g_m = \sqrt{2 \cdot \beta \cdot I_{bias}} \tag{6.9}$$

where I_{bias} is the bias current and β is the transconductance parameter. The transconductance parameter which relates the transistor dimensional size - width W and length L - and the variables due to different process technology is given by:

$$\beta = KP \cdot \frac{W}{L} \tag{6.10}$$

where KP represents the mobility and oxide capacitance factor of the transistor. For the process technology used in this research, the value for KP was calculated as 271 μA/V^2 and 62 μA/V^2 for

the nmos and pmos transistors respectively. For a high-frequency design, all the transistor lengths L are kept to the minimum, i.e. 0.35 μm, so that parasitic effects are reduced. The nmos and pmos transistors in the OTA circuit are matched so that the transconductance parameters for all transistors are made equal:

$$\beta_n = \beta_p = \beta \tag{6.11}$$

where β_n and β_p are the transconductance parameters for the nmos and pmos transistor respectively. Consequently, for all nine topologies in Case Study 3, the width of all pmos transistors were calculated to be four times the width of the nmos ones to achieve the correct transconductance matching between the devices. The transconductance value for the OTAs is tuned by selecting the bias current and the nmos transistor widths for optimisation.

As stated above, for the multiple loop feedback all-pole filter the time constant for both the leapfrog and inverse-follow-the-leader feedback can be derived from the all-pole transfer function and the relationships given by (6.7) and (6.8). The fourth-order transfer function is similar as coded in the VHDL-AMS description for this case study given in Section 6.1. The following time constant values are obtained for the two multiple loop feedback configurations:

	IFLF	LF
τ1	4.16-10	2.43e-10
τ2	2.08e-10	2.51e-10
τ3	1.22e-10	1.72e-10
τ4	6.09e-11	6.09e-11

Table 6.2 The time constant values in seconds for the fourth-order all-pole lowpass multiple loop feedback IFLF and LF OTA-C filters.

As the four OTA cells have the same transconductance, the information above can be used to express the capacitor values as ratios. For example, by normalising to C1, the following values for C2 to C4 in Table 6.3 are obtained. Hence, only capacitor C1 is directly optimised, similar to the strategy used for Cascade 1 to 3.

	IFLF	LF
C2	0.5*C1	1.03*C1
C3	0.29*C1	0.71*C1
C4	0.15*C1	0.25*C1

Table 6.3 The capacitor values for the IFLF and LF topologies expressed in terms of capacitor C1, where the ratio coefficients are based on the time constants for a fourth-order lowpass filter.

However, the initial investigation to use the expression in Table 6.3 shows that a satisfactory result can not be achieved. Therefore, the strategy used by the parametric optimiser is not to constraint the search for capacitance values with the above relationship. By specifying only the maximum and minimum boundary for the capacitances, the optimisation algorithm is given the freedom to select the capacitances within this range. Therefore for all the six multiple loop feedback topologies, each capacitors C1 to C4 are independently optimised.

6.1.4 Results

The first task is to determine which of the models presented in Section 6.1.1 that will be used as the performance model for this case study. The method to evaluate this is by running an HSPICE curve fitting optimisation once on topology Cascade 1. The results are shown in the following table:

Name	Curve-fit error (err_{total})	Time (sec)	Number of iterations
Model 1 (6 points)	0.26	26.0	18
Model 2 (133 points)	0.14	1851	50
Model 3 (36 points)	0.2	150.58	50

Table 6.4 Results for experiment to select the performance model for Case Study 3.

From Table 6.4, it can be seen that Model 1 takes the least time to converge. The time taken for a <u>single</u> HSPICE optimisation to converge is very important for the efficiency of the three-tier algorithm. As will be shown, the three-tier optimisation for the stochastic search had to be restarted several times, the total number of HSPICE optimisation runs amounted up to thousands. Therefore, the architectural optimisation uses Model 1 as the performance model for Case Study 3.

The number of restarts for the three-tier algorithm in this experiment is set to three. For optimisation of circuits containing MOSFET models, the simulation accuracy of HSPICE was set for tighter convergence options than the default values in order to obtain more accurate results. The summary of the three-tier optimisation results for the nine candidates is presented in Table 6.5, where the second column shows the total number of iteration to produce three sets of results, the following columns show the index number of the iteration that produces the best result, and which restart number it belongs to. Finally the last three columns of Table 6.5 give the details of the optimisation result for each topology.

Name	Total iter.	Best iter.	Restart no.	Cost function	Power (mW)	err_{total}
Cascade 1	663	149	2	0.060	25.28	0.035
Cascade 2	977	185	1	0.086	38.89	0.048
Cascade 3	802	692	3	0.145	48.19	0.096
LF 1	521	192	1	0.071	22.49	0.048
LF 2	734	259	1	0.040	16.22	0.024
LF 3	532	525	3	0.143	45.06	0.098
IFLF 1	735	636	3	0.089	32.58	0.056
IFLF 2	870	465	2	0.050	31.76	0.018
IFLF 3	568	122	1	0.066	39.16	0.027

Table 6.5 Architectural optimisation results for Case Study 3.

The topologies arranged from the best to the worst in terms of the cost function, curve-fit error and power consumption are shown in Figure 6.12, Figure 6.13 and Figure 6.14 respectively. From Figure 6.12 and Table 6.5, it can be observed that the best topology with the lowest cost function is topology LF 2, which is the leapfrog topology implemented using the folded-cascode OTA. The best three topologies are the LF 2, followed by IFLF 2 and Cascade 1. Figure 6.15 shows the AC response of LF 2.

However, from Figure 6.13 it can be seen that the most accurate topology is IFLF 2, the inverse follow-the-leader feedback - also being implemented using the folded-cascode OTA. The best overall topology, LF 2 has a slightly larger curve-fit error value than IFLF 2. The topology that consumes least power, about 16 mW, is also LF 2, the best overall topology. The topology with the worse performance is Cascade 3, followed by LF 3. Both topologies have the highest curve-fit error (LF 3 followed by Cascade 3) as well power consumption (Cascade 3 followed by LF 3). It is observed that all three topologies implemented using the wide-swing folded-cascode OTA consumes the most power, this would be as expected because this OTA configuration has more MOSFETs than the other two (Table 6.1).

The optimisation results of Case Study 3 are in agreement with published information [124] that states that the LF configuration gives the best overall performance in comparison to the IFLF and cascade structure .

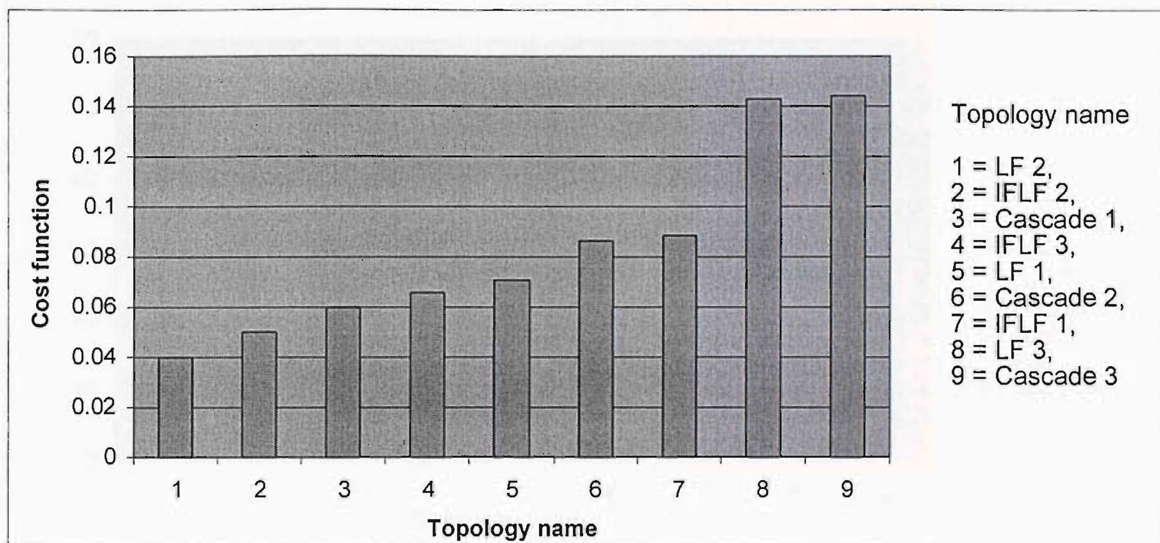


Figure 6.12 Results for Case Study 3 sorted by cost function (ascending).

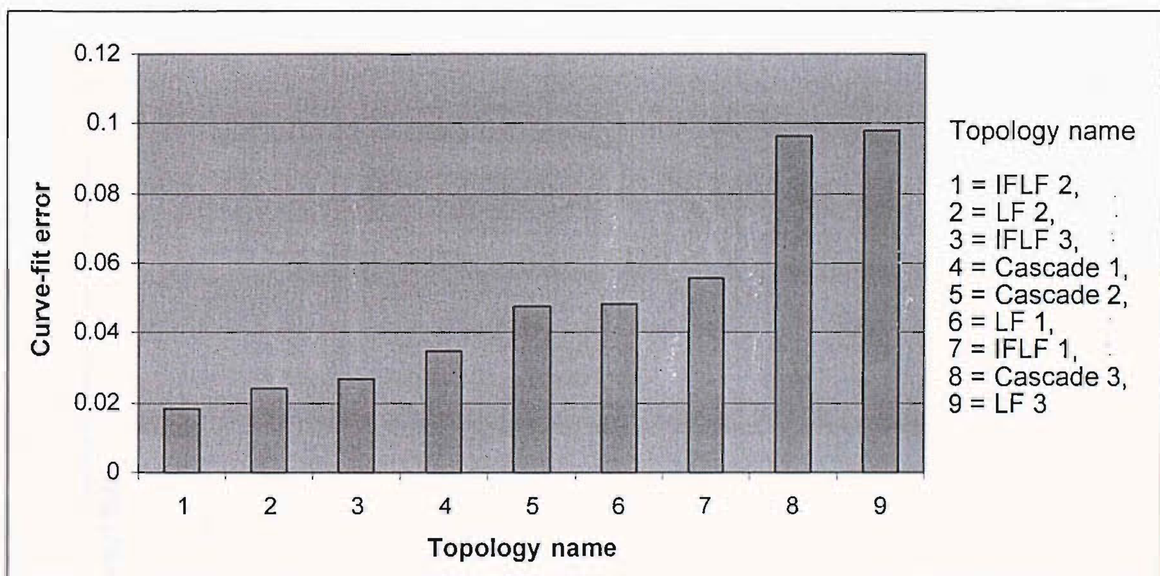


Figure 6.13 Results for Case Study 3 sorted by curve-fit error (ascending).

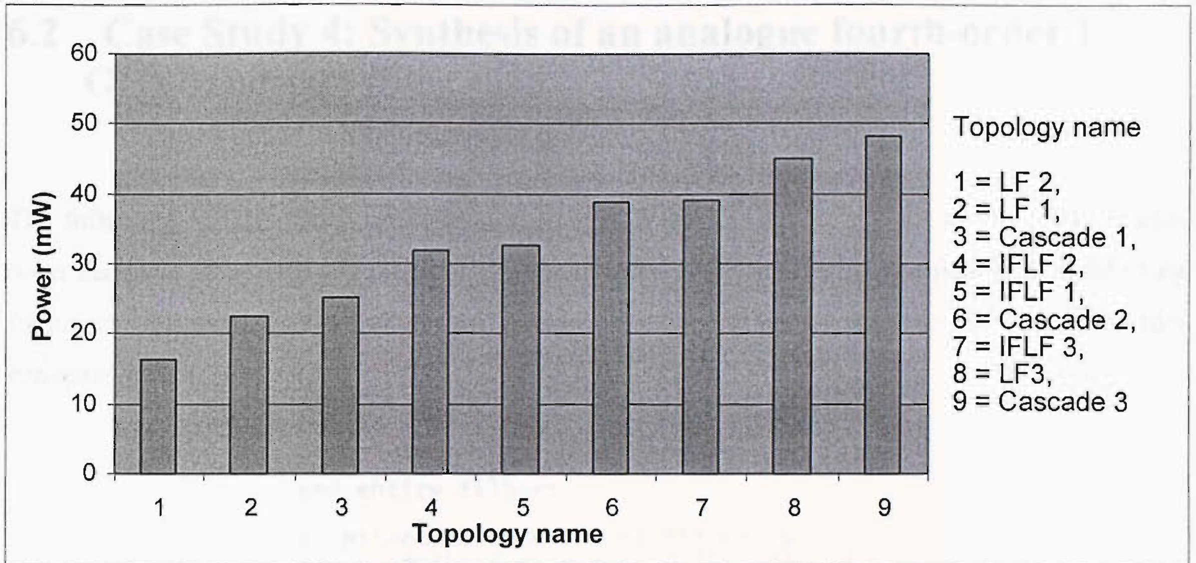


Figure 6.14 Results for Case Study 3 sorted by power consumption (ascending).

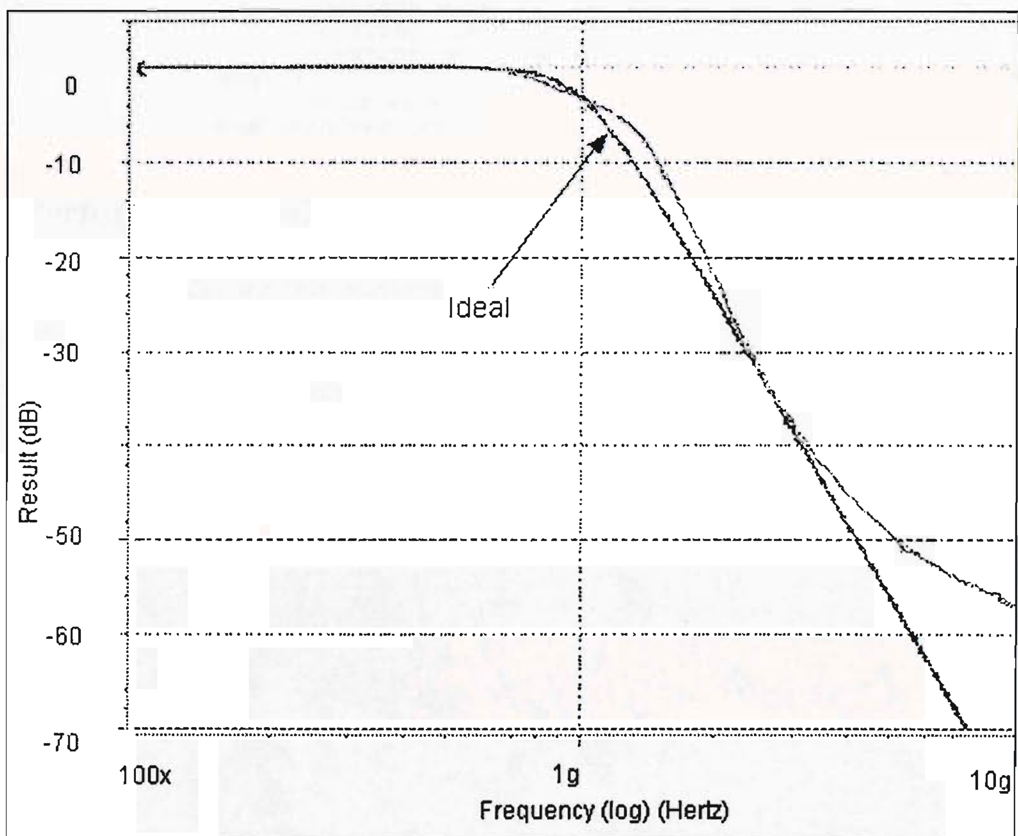


Figure 6.15 AC response for the best topology (LF2) in Case Study 3.

6.2 Case Study 4: Synthesis of an analogue fourth-order 1 GHz bandpass filter

The following VHDL-AMS model is used to specify the desired behaviour of the 1GHz fourth-order bandpass filter with a Q factor of 10. The transfer function for this specification is build using frequency transformation techniques that converts a second-order lowpass filter into a fourth-order bandpass filter.

```
entity filter is
      port (quantity Vin: real;
            quantity Vout: out real);
end entity filter;

architecture transfer of filter is
      constant w: real:= 2.0*3.142*1e9;
      constant w2: real:= w*w;
      constant w3: real:= w2*w;
      constant w4: real:= w2*w2;
      constant a:real:=1/w2;
      constant b:real:=14.142/w;
      constant c:real:=201/w2;
      constant d:real:=14.142/w3;
      constant e:real:=100/w4;
      constant num: real_vector:= (0,0,a);
      constant den: real_vector:= (100,b,c,d,e);
begin
      Vout == Vin'LTF(num,den);
end architecture;
```

6.2.1 Performance model

The performance model for this bandpass filter specification is represented by 11 data points between the -3 dB upper and lower frequencies of the 1 GHz bandpass filter. The plot of the magnitude versus frequency points is shown in Figure 6.16, where the normalised magnitude axis represents the output voltage or current.

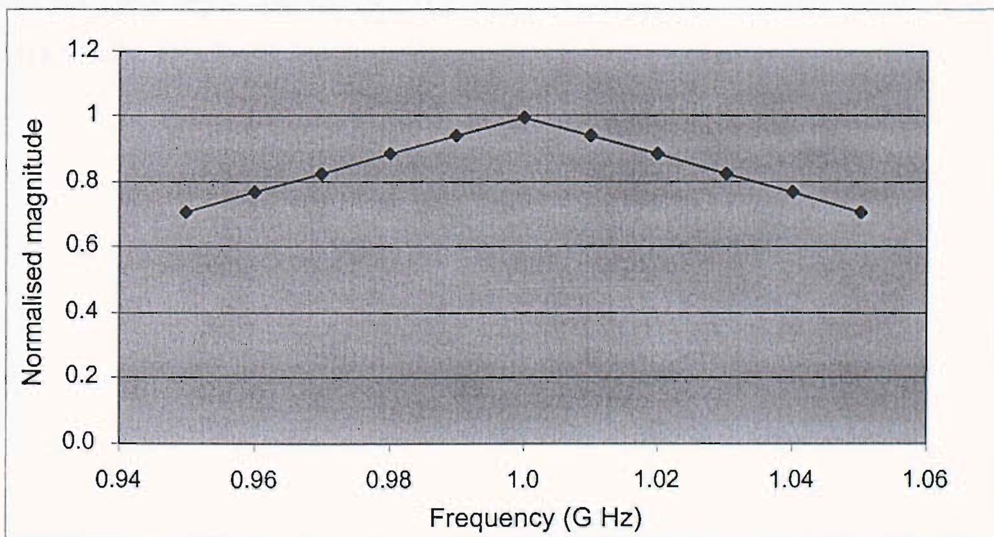


Figure 6.16 The plot of the normalised ideal curve for Case Study 4.

6.2.2 Fourth-order bandpass analogue filter candidates

As mentioned when discussing topologies for integrated high frequency analogue filters in Section 6.1.2, a bandpass filter for similar applications may be designed from OTA-C configurations. Q-enhanced LC circuits with silicon inductors are also suitable [124].

A basic second-order (biquad) bandpass OTA-C filter is shown below in Figure 6.17. The Q factor of the circuit is set by the ratios of capacitors C1 and C2 as shown in (6.2). The cascade of two similar biquad sections produces a fourth-order bandpass filter. As in Case Study 3, three different OTAs from [3] are used in this cascaded OTA-C structure for Case Study 4. These topologies are named Cascade 4, Cascade 5 and Cascade 6 for the OTA-C filter implemented using the wide-swing OTA, folded-cascode OTA, and wide-swing folded-cascode OTA respectively.

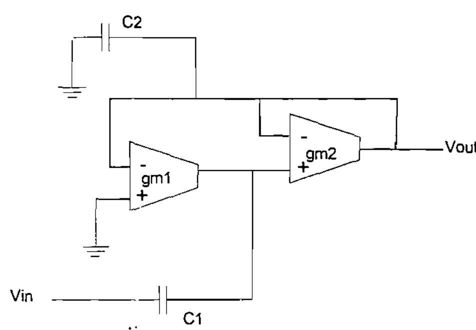


Figure 6.17 A second-order bandpass filter cell implemented using two OTAs and capacitors.

Apart from cascading two OTA-C sections, a fourth-order response may also be obtained by cascading a second-order active LC section with an OTA-C biquad bandpass filter, as shown in Figure 6.18. The second-order LC filter cell is based on the Colpitts LC oscillator that has been introduced in Case Study 1. The second OTA-C bandpass biquad is implemented using the folded-cascode OTA cells. This fourth topology for use in Case Study 4 is named as LC-OTA-C.

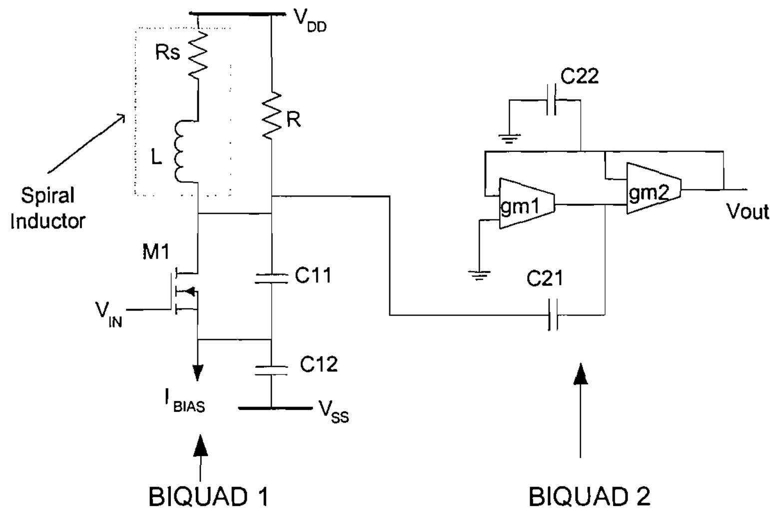


Figure 6.18 Cascade of bandpass biquads: Colpitts and OTA-C.

The fifth topology is another LC circuit known as the coupled resonator [127], which is build with two sections of LC circuits. The block diagram of the second-order Q-enhanced LC circuit is shown in Figure 6.19. The fourth-order circuit is obtained by connecting both parts as shown at the bottom of Figure 6.19. The circuit schematic of the resonant circuit consists of input and output buffers, an LC section, a negative resistance and coupling neutralisation circuits. For this case study, a simplified version of the circuit from [127] is implemented. The circuit diagrams of these blocks are shown in Figure 6.20.

The frequency of this circuit is set to 1 GHz by choosing appropriate values for the inductor and capacitor in the LC circuit. The LC circuit in Figure 6.20 shows that the losses in the spiral inductor are modelled as a series resistance RL. This loss will cause the dampening of the circuit's Q factor. Therefore the negative resistance circuitry [127] enhances the lossy Q by means of positive feedback via cross coupling the gate inputs with drain outputs. The circuit's Q is tuned by varying the tansconductance value using the current source IQ, shown in Figure 6.20. The input and output buffers [127] are differential amplifiers that control the voltage gain of the circuit and to drive off-chip drain loads, respectively. Both are controlled by the current source IB.

As the fourth-order filter requires the implementation of two similar circuits, there will be two inductors in the final design. The implication of this is there will be magnetic coupling between the inductors, which is represented by the coupling coefficient k, and is a function of the placement of the inductors on the die. It is desirable to keep the value of k as low as possible, and for this, the inductors must be adequately separated. Since chip area is very limited, inadequate separation makes the performance suffer from coupling effects. Hence, the coupling neutralisation circuit, shown in Figure 6.20 is essential in reducing this effect.

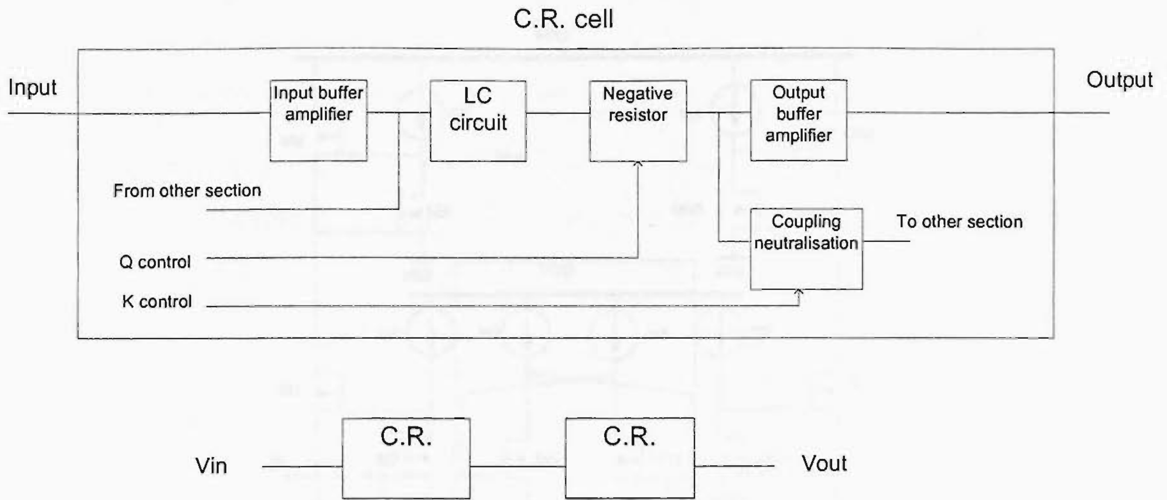


Figure 6.19 Upper: Block diagram of the second order coupled resonator filter (C.R. cell) [127]. Lower: The cascade of two C.R. cells to form a fourth-order bandpass filter.

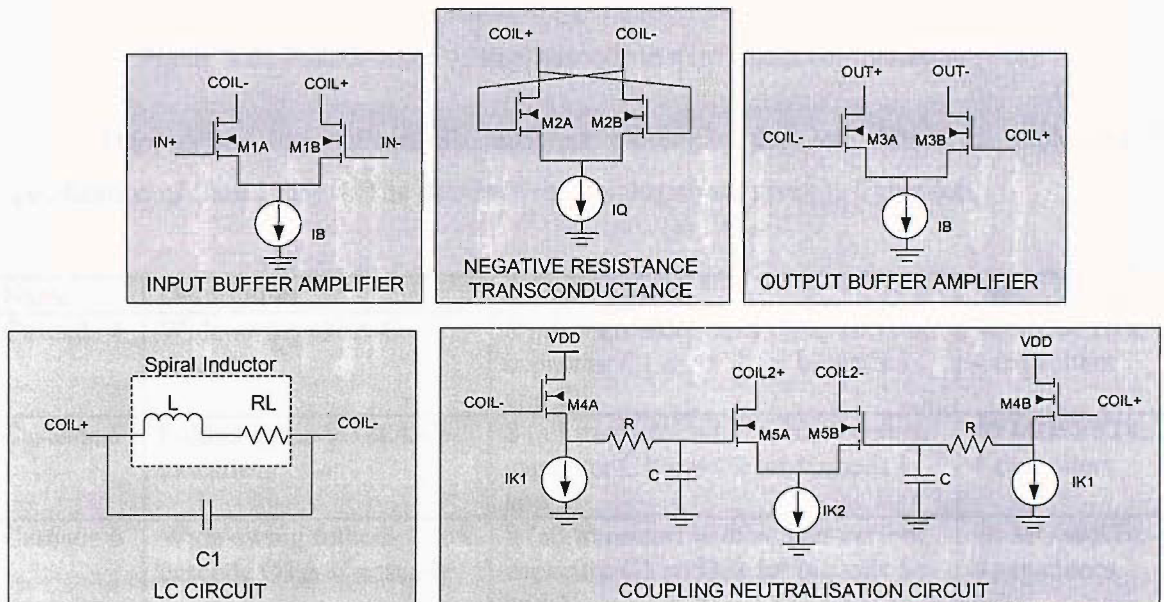


Figure 6.20 Circuit diagrams of the C.R. cell components.

The sixth and final topology to be evaluated in the architectural synthesis of Case Study 4 is the current-mode circuit introduced before in Case Study 2. It is a vertically stacked current-mode biquadratic filter implemented around the regulated cascode configuration [118] that may produce both bandpass and lowpass response. The fourth-order bandpass response can be obtained by taking the output current from the drain of transistors M5 and M55. The fourth-order bandpass filter, referred to as the vertical cascode, is shown in Figure 6.21. The Q-factor of the circuit is tuned via the bias current Io3, while the frequency is adjusted via bias current Io1.

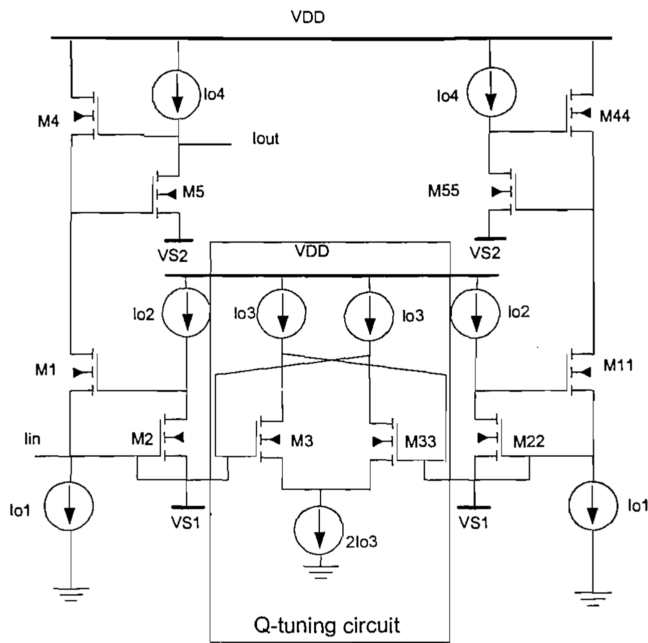


Figure 6.21 Fourth-order vertical cascode in a bandpass configuration [118].

This section has outlined six different topologies that are chosen to implement the specification of Case Study 4. The details of the topologies are given in Table 6.6.

Name	Description	Circuit variables	Size
Cascade 4	Wide-swing OTA-C cascade	8 (all transistor widths, bias current, capacitor C1 and C2 for biquads 1 and 2)	48 MOSFETs, 4 capacitors
Cascade 5	Folded-cascode OTA-C cascade	8 (all transistor widths, bias current, capacitor C1 and C2 for biquads 1 and 2)	48 MOSFETs, 4 capacitors
Cascade 6	Wide-swing folded-cascode OTA-C cascade	8 (all transistor widths, bias current, capacitor C1 and C2 for biquads 1 and 2)	74 MOSFETs, 4 capacitors
LC-OTA-C	Cascade of the colpitts circuit with the folded-cascode OTA-C	5 (2 sets of transistor widths, WM1, and WM3, and bias current IB, capacitor C1 and capacitor ratio k)	25 MOSFETs, 4 capacitors
CR	Coupled resonator circuit	5 (all transistor widths, 4 bias currents IB, IQ, IK1 and IK2)	12 MOSFETs, 2 spiral inductors, 4 capacitors
Vertical cascode	Vertically stacked regulated cascode biquads.	5 (all transistor widths, 4 bias currents I1, I2, I3 and I4)	10 MOSFETs

Table 6.6 The summary of the topologies used in the architectural synthesis of Case Study 4.

6.2.3 Architectural optimisation for Case Study 4

The setting up for architectural optimisation for OTA-C bandpass cascade circuits is similar to that of the lowpass as described in Section 6.1.3. For Cascade 4, Cascade 5 and Cascade 6, all bias currents and transistor widths are optimised, where the width of the pmos transistor is set to be 4 times the width of the nmos transistor.

The second order OTA-C section of the LC-OTA-C topology is also similarly optimised as Cascade 4 to 6, which are the transistor widths, bias currents and capacitor. In the LC part of the filter, three parameters are optimised; the width of its single transistor, capacitor ratio and bias current. The value of the bias current for both the LC and OTA-C parts are set to be equal.

The coupled resonator circuit contains only nmos transistors, whose widths are optimised. There are four bias currents that are independently optimised to provide Q-tuning, gain adjustment and coupling neutralisation control as indicated in the circuit diagram of Figure 6.20. The LC circuit has fixed value to set the frequency at 1 GHz, where the inductance is 8nH and the capacitance is 3.167 pF. The loss due to the spiral inductor implementation is represented by a 5Ω resistor in series with the inductor. The final topology, which is the vertical cascode, has its transistor widths optimised, and other bias currents for tuning the Q factor, frequency and gain.

6.2.4 Results

The optimisation results for Case Study 4 are shown in Table 6.7. The curve-fitting algorithm of the HSPICE optimisation is done against 11 points situated in the range of 0.95 to 1.05 GHz in the frequency response, as shown in Figure 6.16.

Name	Total iter.	Best iter.	Restart no.	Cost function	Power (mW)	err_{total}
Cascade 4	170	142	3	0.0469	30.07	0.017
Cascade 5	96	82	3	0.0325	10.00	0.022
Cascade 6	116	43	1	0.0642	37.60	0.027
LC-OTA-C	601	600	3	0.133	100.71	0.026
CR	1126	721	2	0.884	345.67	0.269
Vertical cascode	360	90	1	0.0211	4.85	0.016

Table 6.7 Architectural optimisation results for Case Study 4.

From Table 6.7, it can be seen that the best result is the vertical cascode filter. This topology has both the lowest error function and power consumption. In fact, the novelty of this design [118] is its low current consumption due to the proposed vertical connection of the

regulated cascade circuit. The frequency response of this curve is shown in Figure 6.25. It can be seen that when compared to the ideal magnitude response, the response of the vertical cascade perfectly matches the ideal within the range of 0 to – 5 dB in the passband, as specified by the performance model of Figure 6.16.

The rank of the topology according to the best to worst in terms of overall cost function, curve-fit error and power are illustrated in Figure 6.22, Figure 6.23 and Figure 6.24, respectively. From these figures, it can be seen that the topology with the worse performance is the coupled resonator, which has both a significantly higher curve-fit error and power consumption compared to the other topologies. This is attributed to the fact that this topology is implemented with simplifications made to the original circuit [127], where only the functionally-important components are implemented, as shown in Figure 6.20. However, with the curve-fit error of about 0.27, this topology is fairly accurate, and with a more enhanced implementation, it is expected that its performance would improve. From Figure 6.23, it can be seen that apart from the coupled resonator, the other five topologies are very accurate, with curve-fit error of less than 0.03. In terms of power consumption, after the coupled resonator, the LC-OTA-C consumes most power, followed by Cascade 6. The LC-OTA-C topology actually have five current sources, where one is used to bias the Colpitts circuit, and transconductances gm1 and gm2 of the OTA each contains two current sources (Figure 6.18). As only a single value for bias current is used for the five current sources in the optimisation process (Table 6.6), the 'heavy' load pushes the current source to its limit, hence the higher power consumption. Nevertheless, the LC-OTA-C topology gives accurate magnitude response despite consuming a fairly large amount of power.

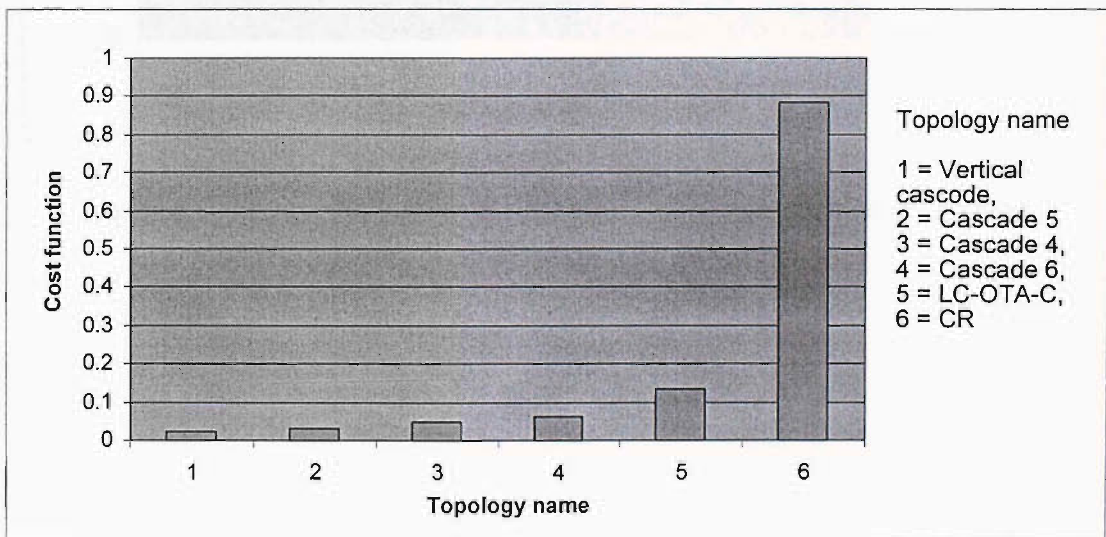


Figure 6.22 Results for Case Study 4 sorted by cost function (ascending).

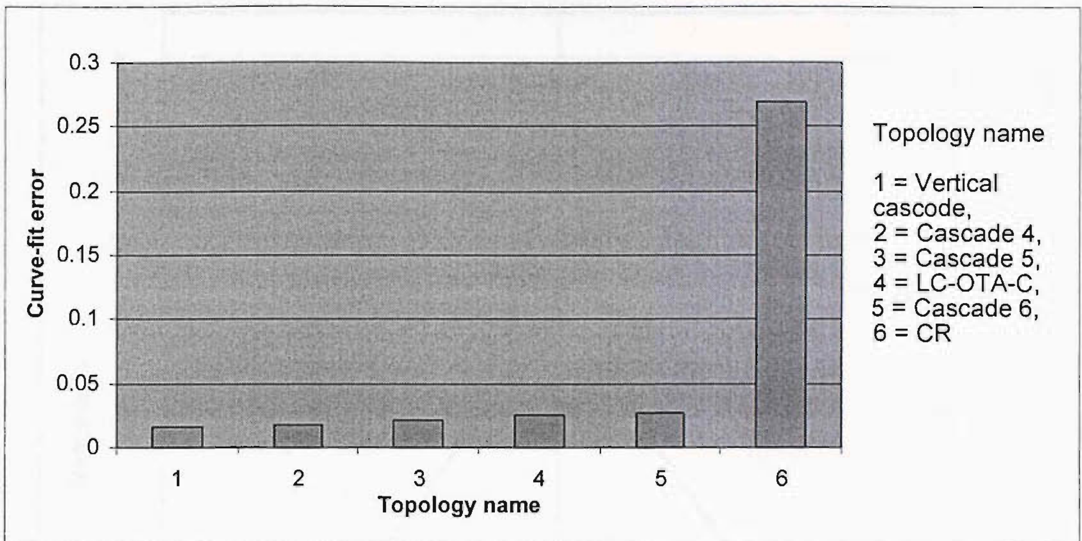


Figure 6.23 Results for Case Study 4 sorted by curve-fit error (ascending).

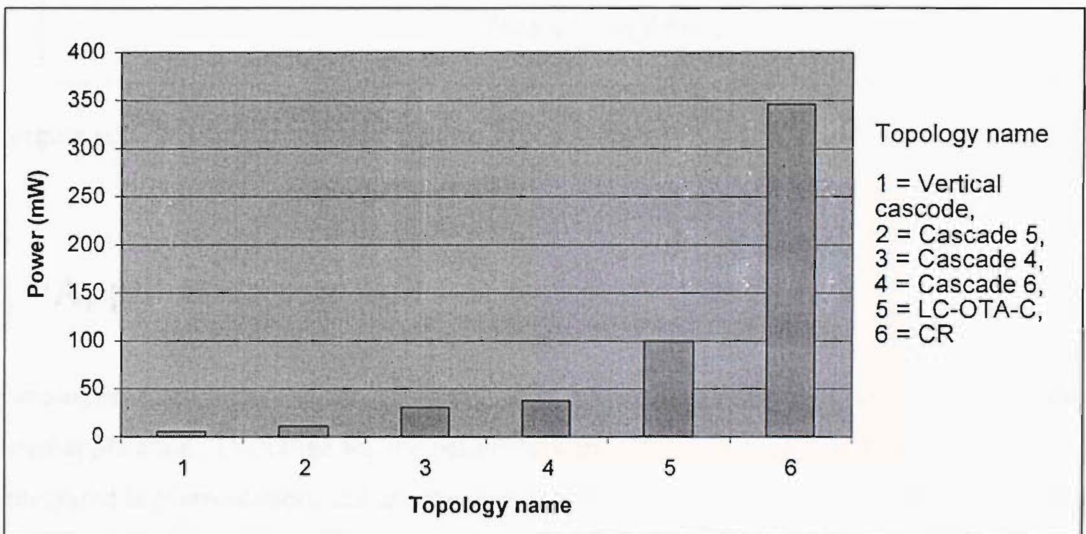


Figure 6.24 Results for Case Study 4 sorted by power consumption (ascending).

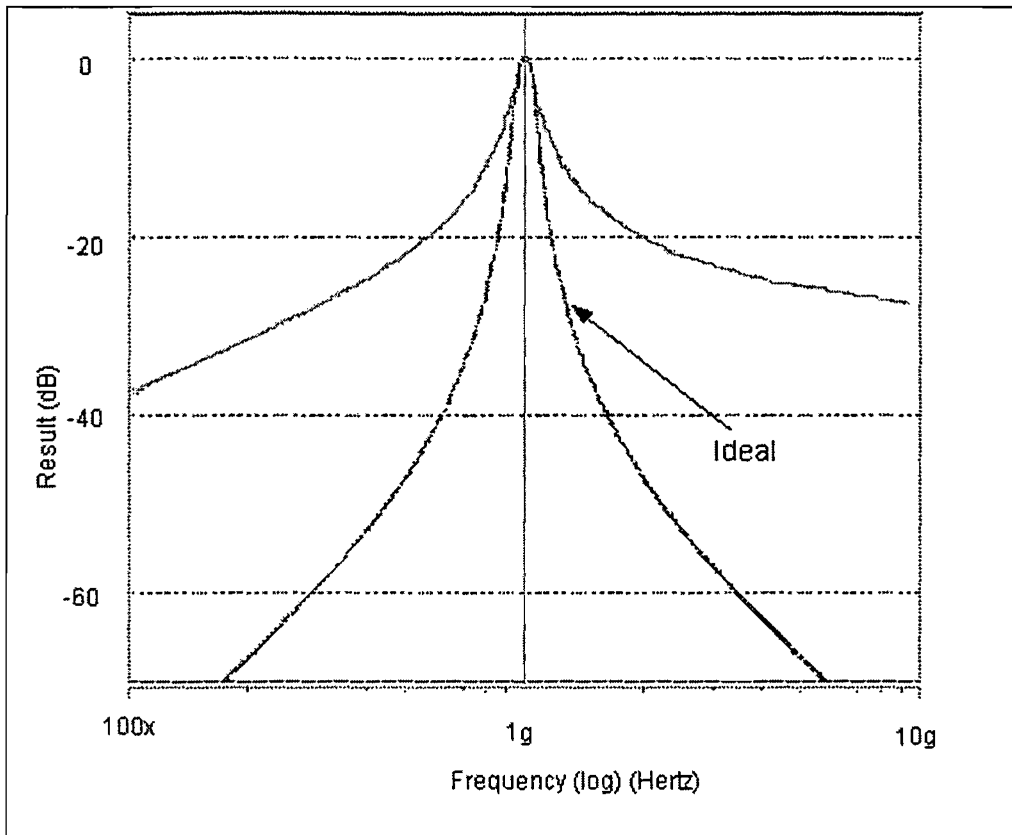


Figure 6.25 AC simulation results of the best topology (vertical cascode) for Case Study 4.

6.3 Application examples of topology Cascade 1

The analogue filter library contains library cells whose parameters can be adjusted to suit the required application. The range for the parameters are chosen to be within those that are practical for integrated implementation, and are the user is not required to constrain any of the parameter to realise the required specification. Therefore, to show that the knowledge required from the filter cell library is not application based, this section presents three application examples of fourth-order lowpass filters using topology Cascade 1. For each example, the VHDL-AMS description of the analogue filter is passed to the synthesiser, and then together with the given performance model, the topology Cascade 1 is optimised using the three-tier algorithm. The specified number of restarts for the algorithm is five.

The three examples including their VHDL-AMS description as well as the selected performance models are given in Sections 6.3.1, 6.3.2 and 6.3.3. For Example 1 and Example 2, the performance models for the Chebyshev response are represented by 20 points as the ripple in the passband requires more data points so that the curve-fitting optimisation result would be more accurate. These ideal curves are shown in Figure 6.26 and Figure 6.27 respectively. For the Butterworth response in Example 3, the ideal curve is represented by seven points, as shown in

Figure 6.28. The three-tier optimisation results for the examples are presented in Table 6.8 of Section 6.3.4. AC waveforms of the three examples are also shown.

6.3.1 Example 1: Chebyshev 0.5 db ripple, cut off at 1 GHz (Gain=1)

```
entity filter is
      port (quantity Vin: real;
            quantity Vout: out real);
end entity filter;

architecture transfer of filter is
     constant w: real:= 2.0*3.142*1e9;
     constant w2: real:= w*w;
     constant w3: real:= w2*w;
     constant w4: real:= w2*w2;
     constant a:real:=0.3579*w4;
     constant b:real:=0.3791*w4;
     constant c:real:=1.0255*w3;
     constant d:real:=1.7169*w2;
     constant e:real:=1.1974*w;
     constant num: real_vector:= (a);
     constant den: real_vector:= (b,c,d,e,1.0);
begin
      Vout == Vin'LTF(num,den);
end architecture;
```

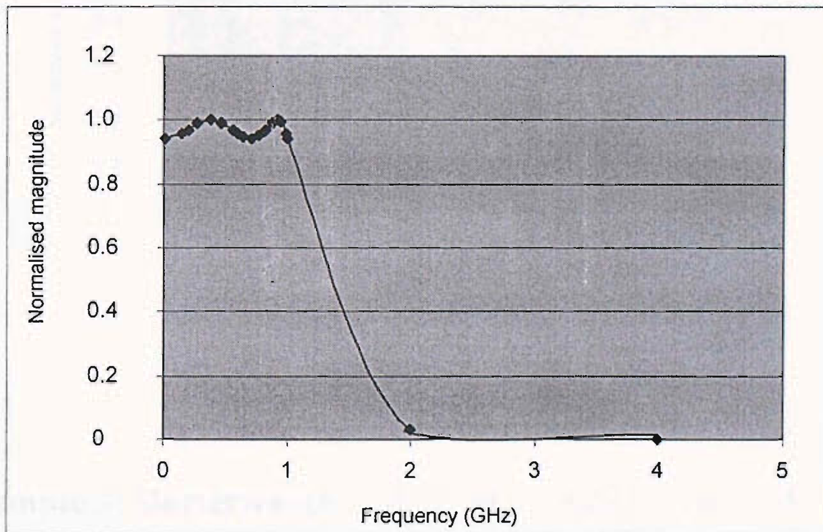


Figure 6.26 Performance model for Example 1.

6.3.2 Example 2: Chebyshev 3 db ripple, cut off at 0.5 GHz (Gain =1)

```
entity filter is
     port (quantity Vin: real;
           quantity Vout: out real);
end entity filter;

architecture transfer of filter is
     constant w: real:= 2.0*3.142*0.5e9;
     constant w2: real:= w*w;
     constant w3: real:= w2*w;
     constant w4: real:= w2*w2;
     constant c:real:=0.4048/w;
     constant d:real:=1.1691/w2;
     constant e:real:=0.5816/w3;
     constant f:real:=1/w4;
     constant num: real_vector:= (0.177);
     constant den: real_vector:= (0.1770,c,d,e,f);
begin
     Vout == Vin'LTF(num,den);
end architecture;
```

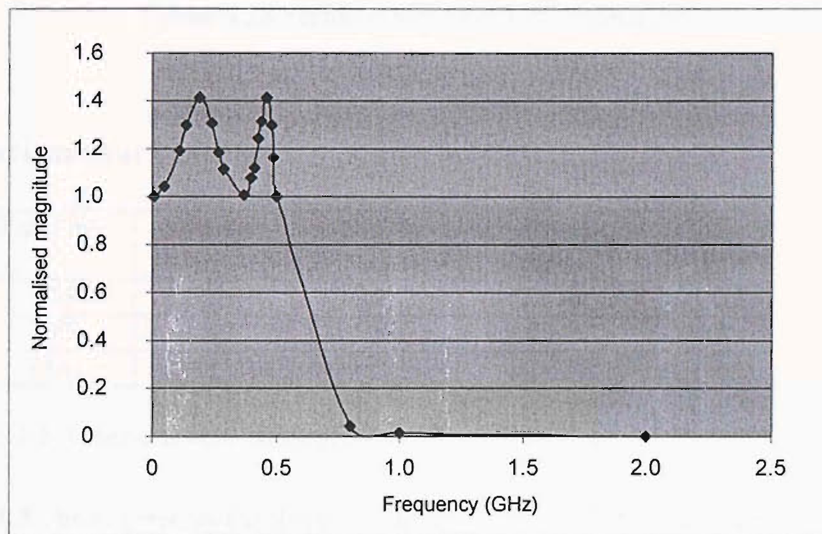


Figure 6.27 Performance model for Example 2.

6.3.3 Example 3: Butterworth, cut off at 0.5 GHz (Gain = 1)

```
entity filter is
     port (quantity Vin: real;
           quantity Vout: out real);
end entity filter;

architecture transfer of filter is
     constant w: real:= 2.0*3.142*0.5e9;
     constant w2: real:= w*w;
     constant w3: real:= w2*w;
     constant w4: real:= w2*w2;
     constant c:real:=2.6131/w;
     constant d:real:=3.4142/w2;
     constant e:real:=2.6131/w3;
     constant f:real:=1/w4;
     constant num: real_vector:= (1);
     constant den: real_vector:= (1,c,d,e,f);
begin
     Vout == Vin'LTF(num,den);
end architecture;
```

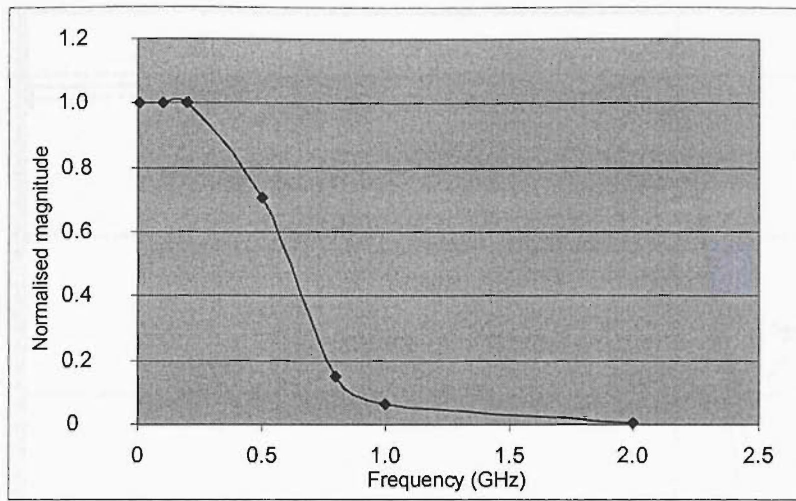


Figure 6.28 Performance model for Example 3.

6.3.4 Experimental results

Name	Total iter.	Best iter.	Restart no	Cost function	Power (mW)	Err_{total}
Example 1	778	576	4	0.116	29.53	0.0863
Example 2	887	447	3	0.0896	12.31	0.0773
Example 3	1363	285	1	0.0525	9.26	0.0432

Table 6.8 Three-tier optimisation results for different applications of Cascade 1.

Table 6.8 above presents the three-tier optimisation result for the Examples 1, 2 and 3. The second column shows the number of iterations for five restarts, and the next two columns give the index number of the best iteration and which restart it belongs to. The last three columns give the best cost function for each example together with the power consumption and curve-fit error. The optimisation proved to be successful as it can be seen that for all three examples, the resulting cost functions are all very low. The AC waveform using filter parameters for the best cost function of Example 1 is shown in Figure 6.29 and Figure 6.30, while for Example 2 is shown in Figure 6.31 and Figure 6.32. Figure 6.33 is the waveform for Example 3. From these diagrams, it can be seen that the best results from the optimisation closely matches the ideal response.

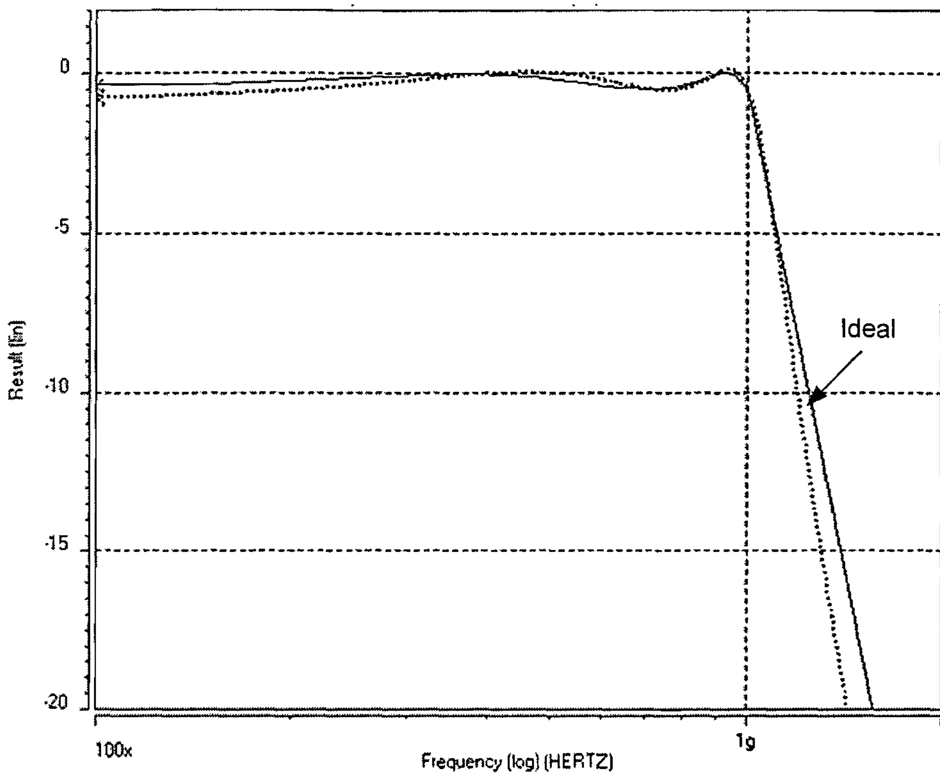


Figure 6.29 AC waveform for Example 1.

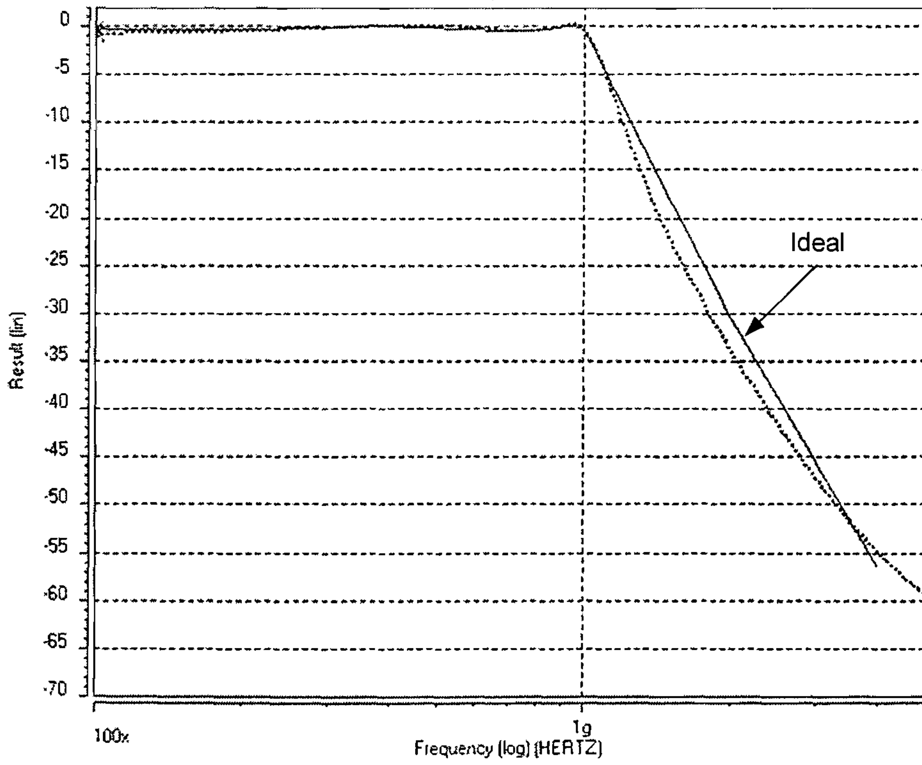


Figure 6.30 AC waveform for Example 1.

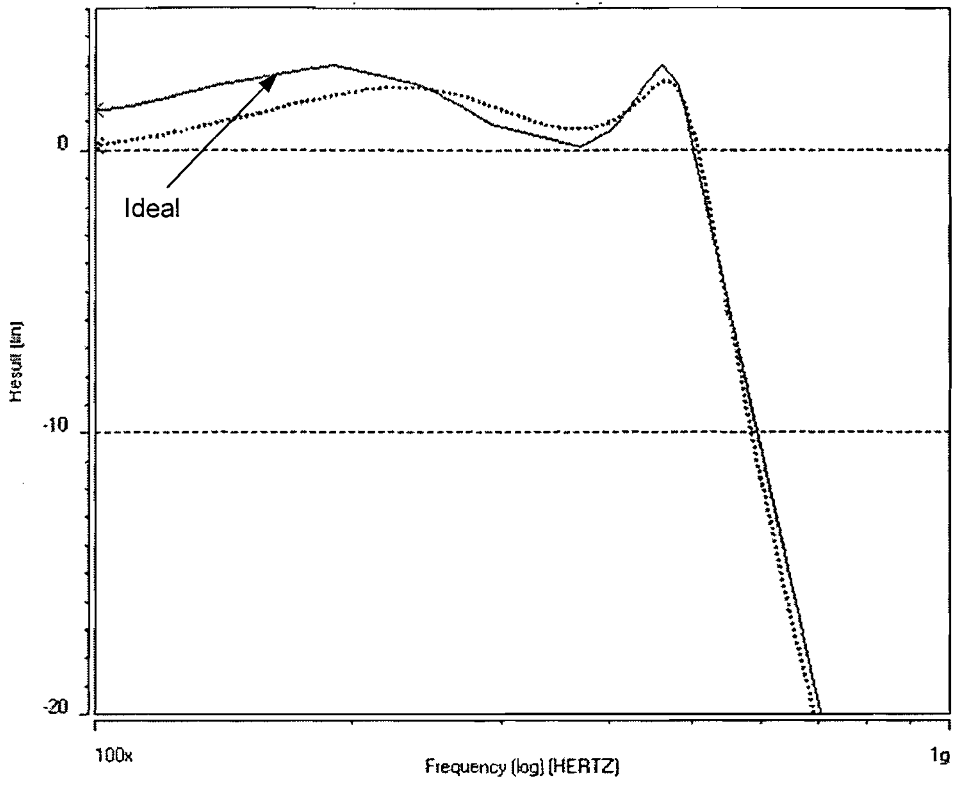


Figure 6.31 AC waveform for Example 2.

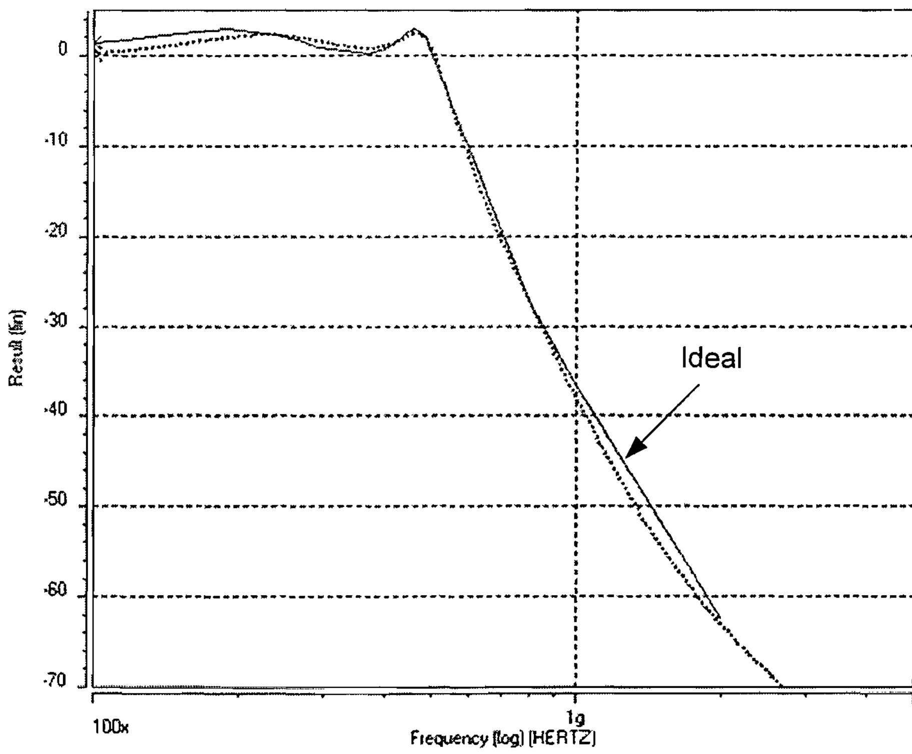


Figure 6.32 AC waveform for Example 2.

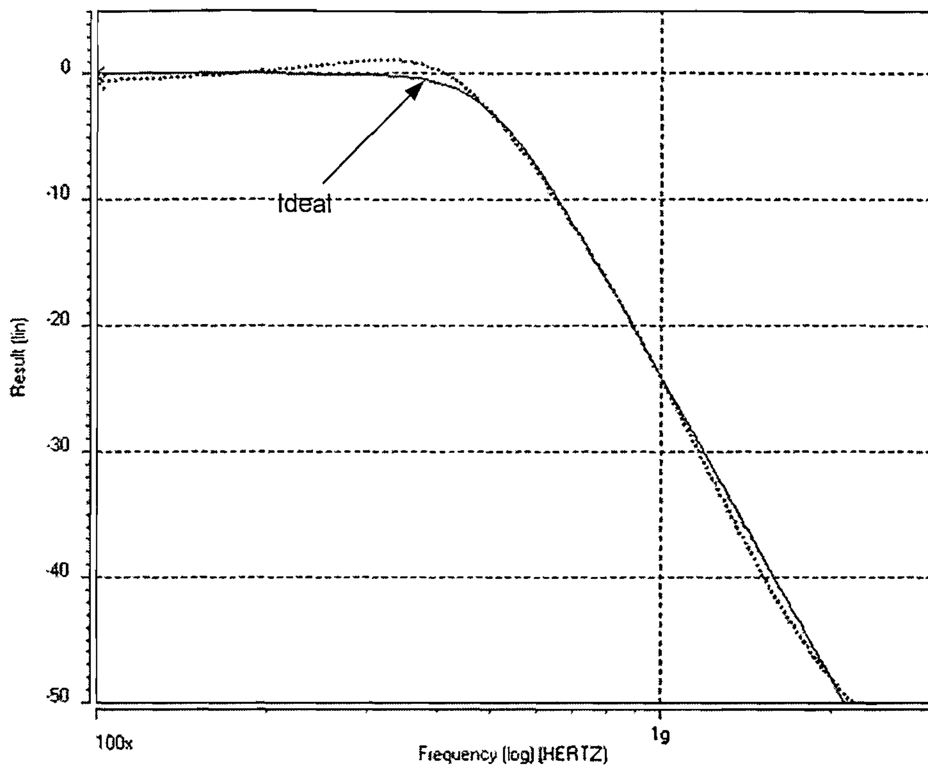


Figure 6.33 AC waveform of Example 3.

6.4 Concluding remarks

In this chapter, Case Studies 3 and 4 have been used as practical examples to demonstrate the process of architectural synthesis of high-frequency analogue filters from behavioural VHDL-AMS descriptions. In both case studies a VHDL-AMS description was used for the analogue filter specification. From this description, a suitable performance model to represent the ideal magnitude response of the filter is derived and utilised by the three-tier optimisation algorithm. It has been shown how a set of suitable filter cell topologies to realise the specification can be easily built from the wealth of information about various filter topologies and design techniques in the literature. It is shown in Case Study 3 and Case Study 4 that an excellent result can be achieved by using the three-tier parametric optimisation on all candidates. The last part of this chapter presents three different application examples for a single filter topology. All the three examples were synthesised successfully.

7 Conclusions and Future Work

The aims of this research as outlined in Chapter 1, have been achieved. The primary aim was to investigate and develop techniques for VHDL-AMS-based synthesis of high-frequency analogue filters suitable for use in a mixed-signal design environment. To achieve this aim, the research work focused on two areas. The first area concerns methods of behavioural analogue filter modelling for synthesis using VHDL-AMS and techniques to extract the necessary information from the model's parse tree. The information is then used to map the behavioural specification into structural architectures. The second part of the research focuses on techniques to optimise the synthesised filter architectures for accuracy of magnitude response, as well as power consumption.

The method of using parse trees to extract circuit-level structure from high-level descriptions proved to be effective. It was shown that parse trees allow for easy detection of synthesisable constructs and support recursive static evaluations of expressions necessary in calculation of filter parameters. Case Studies 1 and 2 demonstrated in the early stages of this research that optimised HSPICE netlists can be successfully obtained from VHDL-AMS behavioural descriptions. The initial strategy of stochastically optimising a filter topology has been further expanded into a complete synthesis strategy with better optimisation techniques that are suitable for architectural optimisation of several analogue filter candidate topologies. Elements of the synthesis methodology, such as performance specification, cell mapping and realisability analysis have been developed and tested in detail. Together with the three-tier architectural and parametric optimisation methodology they form FIST, a complete synthesis system. The practical use of FIST is demonstrated with Case Studies 3 and 4.

The novel contributions in both main areas of this research are as follows:

A) The methods and supporting tools for behavioural synthesis of analogue filters from VHDL-AMS descriptions:

- Methods to extract synthesisable VHDL-AMS constructs for behavioural models of analogue filters necessary for the development of an architectural synthesis methodology based on VHDL-AMS parse trees.

- The development and implementation of tools to support the behavioural modelling of synthesisable constructs, specifically a recursive static calculator.

B) Techniques for architectural synthesis of integrated high-frequency analogue filters from VHDL-AMS parse trees:

- The development of a synthesis methodology that can be easily extended to support a more general, mixed-signal synthesis system based on VHDL-AMS.
- The development of an effective architectural optimisation engine for analogue filter synthesis based on three-tier optimisation, in which a combination of the stochastic search, the downhill simplex algorithm, and built-in HSPICE optimisation provides a possibility for a global optimisation strategy in which the accuracy errors and power consumption are minimised.
- The practical demonstration of the feasibility to include full HSPICE simulation in a complex and numerically difficult synthesis environment. Coupled with foundry-supplied transistor models for a state-of-the-art 0.35um CMOS technology, this strategy enabled a successful and reliable implementation of the new synthesis methodology for a high-frequency application.

It can be concluded that architectural synthesis from VHDL-AMS parse trees has proven to be effective and provides a promising direction for further research into general, VHDL-based synthesis. The results obtained in the course of this research demonstrate that VHDL-AMS has the potential to play an important role in the realm of mixed-signal synthesis. Specific directions for further research can be identified as follows:

- Further exploration of using parse trees as an intermediate representation of synthesisable descriptions in a mixed-signal environment.
- Extension of the synthesis technique to other mixed-signal ASICs such as amplifiers, A/D and D/A converters and PLLs.

In the context of synthesis for high-frequency applications, the three-tier optimisation method can be extended in several aspects:

- As this method involves costly iterative simulations, the use of high-performance parallel processing or grid computing with a view to generalise the method and make it suitable for larger analogue designs might be investigated.
- To include automatic layout generation and post-layout verification. Thus the effect of parasitics that is expected to degrade the performance of analogue circuits, especially at high-frequencies, can be considered early in the design cycle.

- As the three-tier optimisation algorithm has been tested on a limited number of practical cases, it might be beneficial to investigate other heuristic approaches, such as simulated annealing or genetic optimisation and test alternative versions of the algorithm on a larger number of practical examples.

Appendix A: Analogue filter netlists for Case Studies 3 and 4

A.1 Cascade 1

```
.subckt OTA1 VINP VINM VOUT
I1     VDD 9 DC OIB AC 0 0
I2     11 VSS DC OIB AC 0 0
M1      1 VINM 11 VSS nmos0553  L=0.35u W=OWM1
M2      3 VINP 11 VSS nmos0553  L=0.35u W=OWM1
M3      1 1 VDD VDD pmos0553  L=0.35u W=OWM2
M4      3 3 VDD VDD pmos0553  L=0.35u W=OWM2
M5      VOUT 3 VDD VDD pmos0553  L=0.35u W=OWM2
M6      7 1 VDD VDD pmos0553  L=0.35u W=OWM2
M7      VOUT 7 VSS VSS nmos0553  L=0.35u W=OWM1
M8      3 8 VSS VSS nmos0553  L=0.35u W=OWM1
M9      8 VINM 9 VDD pmos0553  L=0.35u W=OWM2
M10     7 VINP 9 VDD pmos0553  L=0.35u W=OWM2
M11     8 8 VSS VSS nmos0553  L=0.35u W=OWM1
M12     7 7 VSS VSS nmos0553  L=0.35u W=OWM1
VDD     VDD 0 DC 3.3v AC 0 0
VSS     VSS 0 DC -3.3v AC 0 0
.ends

X_OTA1      VIN1 VOUT2 VOUT1 OTA1
X_OTA2      VOUT1 VOUT2 VOUT2 OTA1
X_OTA3      VOUT2 VOUT4 VOUT3 OTA1
X_OTA4      VOUT3 VOUT4 VOUT4 OTA1
C_1         VOUT1 0 OCAP1
C_1a        VOUT3 0 OCAP1
C_2         VOUT2 0 OCAP2
C_2a        VOUT4 0 OCAP2
V_IN        VIN1 0 DC 0 AC 1 0
RIN         VIN1 0 100MEG
```

A.2 Cascade 2

```
.subckt OTA2 VINP VINM VOUT
I1      4 VSS DC OIB AC 0 0
I2      1 VSS DC OIB AC 0 0
M1      2 VINP 1 VSS nmos0553  L=0.35u W=OWM1
M2      3 VINM 1 VSS nmos0553  L=0.35u W=OWM1
```

```
M3      5 5 VDD VDD pmos0553  L=0.35u W=OWM2
M4      4 4 5 VDD pmos0553  L=0.35u W=OWM2
M5      2 5 VDD VDD pmos0553  L=0.35u W=OWM2
M6      3 5 VDD VDD pmos0553  L=0.35u W=OWM2
M7      9 4 2 VDD pmos0553  L=0.35u W=OWM2
M8      VOUT 4 3 VDD pmos0553  L=0.35u W=OWM2
M9      9 9 7 VSS nmos0553  L=0.35u W=OWM1
M10     VOUT 9 8 VSS nmos0553  L=0.35u W=OWM1
M11     7 7 VSS VSS nmos0553  L=0.35u W=OWM1
M12     8 7 VSS VSS nmos0553  L=0.35u W=OWM1
VDD     VDD 0 DC 3.3v AC 0 0
VSS     VSS 0 DC -3.3v AC 0 0
.ends


X_OTA1       VIN1 VOUT2 VOUT1 OTA2
X_OTA2       VOUT1 VOUT2 VOUT2 OTA2
X_OTA3       VOUT2 VOUT4 VOUT3 OTA2
X_OTA4       VOUT3 VOUT4 VOUT4 OTA2
C_1          VOUT1 0 OCAP1
C_1a         VOUT3 0 OCAP1
C_2          VOUT2 0 OCAP2
C_2a         VOUT4 0 OCAP2
V_IN         VIN1 0 DC 0 AC 1 0
RIN          VIN1 0 100MEG
```

A.3 Cascade 3

```
.subckt OTA3 VINP VINM VOUT Vbias1 Vbias2 Vbias3 Vbias4
M1      2 VINP 1 VSS nmos0553  L=0.35u W=OWM1
M2      3 VINM 1 VSS nmos0553  L=0.35u W=OWM1
M5      2 Vbias1 VDD VDD pmos0553  L=0.35u W=OWM2
M6      3 Vbias1 VDD VDD pmos0553  L=0.35u W=OWM2
M7      8 Vbias2 2 VDD pmos0553  L=0.35u W=OWM2
M8      VOUT Vbias2 3 VDD pmos0553  L=0.35u W=OWM2
M9      8 Vbias3 5 VSS nmos0553  L=0.35u W=OWM1
M10     VOUT Vbias3 6 VSS nmos0553  L=0.35u W=OWM1
M11     5 8 VSS VSS nmos0553  L=0.35u W=OWM1
M12     6 8 VSS VSS nmos0553  L=0.35u W=OWM1
M13     1 Vbias3 4 VSS nmos0553  L=0.35u W=OWM1
M14     4 Vbias4 VSS VSS nmos0553  L=0.35u W=OWM1
M15     5 VINP 7 VDD pmos0553  L=0.35u W=OWM2
M16     6 VINM 7 VDD pmos0553  L=0.35u W=OWM2
M17     9 Vbias1 VDD VDD pmos0553  L=0.35u W=OWM2
M18     7 Vbias2 9 VDD pmos0553  L=0.35u W=OWM2
VDD     VDD 0 DC 3.3v AC 0 0
VSS     VSS 0 DC -3.3v AC 0 0
.ends

.subckt BIASCCT Vbias1 Vbias2 Vbias3 Vbias4
MB1     12 12 Vbias4 VSS nmos0553  L=40u W=30u
MB10    Vbias3 Vbias4 VSS VSS nmos0553  L=0.35u W=OWM1
MB2     10 10 Vbias1 VDD pmos0553  L=10u W=35u
MB3     Vbias4 Vbias4 VSS VSS nmos0553  L=0.35u W=OWM1
MB4     12 10 11 VDD pmos0553  L=0.35u W=OWM2
MB5     Vbias1 Vbias1 VDD VDD pmos0553  L=0.35u W=OWM2
MB6     11 Vbias1 VDD VDD pmos0553  L=0.35u W=OWM2
MB7     Vbias2 Vbias1 VDD VDD pmos0553  L=0.35u W=OWM2
```

```
MB8     VSS 10 Vbias2 VDD pmos0553  L=0.35u W=OWM2
MB9     VDD 12 Vbias3 VSS nmos0553  L=0.35u W=OWM1
I       10 VSS DC OIB
VDD     VDD 0 DC 3.3v AC 0 0
VSS     VSS 0 DC -3.3v AC 0 0
.ends

X_OTA1       VIN1 VOUT2 VOUT1 Vbias1 Vbias2 Vbias3 Vbias4 OTA3
X_OTA2       VOUT1 VOUT2 VOUT2 Vbias1 Vbias2 Vbias3 Vbias4 OTA3
X_OTA3       VOUT2 VOUT4 VOUT3 Vbias1 Vbias2 Vbias3 Vbias4 OTA3
X_OTA4       VOUT3 VOUT4 VOUT4 Vbias1 Vbias2 Vbias3 Vbias4 OTA3
X_BIASNET    Vbias1 Vbias2 Vbias3 Vbias4 BIASCCT
C_1          VOUT1 0 OCAP1
C_1a         VOUT3 0 OCAP1
C_2          VOUT2 0 OCAP2
C_2a         VOUT4 0 OCAP2
V_IN         VIN1 0 DC 0 AC 1 0
RIN          VIN1 0 100MEG
```

A.4 Cascade 4

```
.subckt OTA1 VINP VINM VOUT
I1      VDD 9 DC OIB AC 0 0
I2      11 VSS DC OIB AC 0 0
M1      1 VINM 11 VSS nmos0553  L=0.35u W=OWM1
M2      3 VINP 11 VSS nmos0553  L=0.35u W=OWM1
M3      1 1 VDD VDD pmos0553  L=0.35u W=OWM2
M4      3 3 VDD VDD pmos0553  L=0.35u W=OWM2
M5      VOUT 3 VDD VDD pmos0553  L=0.35u W=OWM2
M6      7 1 VDD VDD pmos0553  L=0.35u W=OWM2
M7      VOUT 7 VSS VSS nmos0553  L=0.35u W=OWM1
M8      3 8 VSS VSS nmos0553  L=0.35u W=OWM1
M9      8 VINM 9 VDD pmos0553  L=0.35u W=OWM2
M10     7 VINP 9 VDD pmos0553  L=0.35u W=OWM2
M11     8 8 VSS VSS nmos0553  L=0.35u W=OWM1
M12     7 7 VSS VSS nmos0553  L=0.35u W=OWM1
VDD     VDD 0 DC 3.3v AC 0 0
VSS     VSS 0 DC -3.3v AC 0 0
.ends

X_OTA1       0 VOUT2 VOUT1 OTA1
X_OTA2       VOUT1 VOUT2 VOUT2 OTA1
X_OTA3       0 VOUT4 VOUT3 OTA1
X_OTA4       VOUT3 VOUT4 VOUT4 OTA1
C_1          VIN1 VOUT1 OCAP1
C_1a         VOUT2 VOUT3 OCAP1
C_2          VOUT2 0 OCAP2
C_2a         VOUT4 0 OCAP2
V_IN         VIN1 0 DC 0 AC 1 0
RIN          VIN1 0 100MEG
```

A.5 Cascade 5

```
.subckt OTA2 VINP VINM VOUT
I1      4 VSS      DC OIB AC 0 0
```

```
I2      1 VSS DC OIB AC 0 0
M1      2 VINP 1 VSS nmos0553  L=0.35u W=OWM1
M2      3 VINM 1 VSS nmos0553  L=0.35u W=OWM1
M3      5 5 VDD VDD pmos0553  L=0.35u W=OWM2
M4      4 4 5 VDD pmos0553  L=0.35u W=OWM2
M5      2 5 VDD VDD pmos0553  L=0.35u W=OWM2
M6      3 5 VDD VDD pmos0553  L=0.35u W=OWM2
M7      9 4 2 VDD pmos0553  L=0.35u W=OWM2
M8      VOUT 4 3 VDD pmos0553  L=0.35u W=OWM2
M9      9 9 7 VSS nmos0553  L=0.35u W=OWM1
M10     VOUT 9 8 VSS nmos0553  L=0.35u W=OWM1
M11     7 7 VSS VSS nmos0553  L=0.35u W=OWM1
M12     8 7 VSS VSS nmos0553  L=0.35u W=OWM1
VDD     VDD 0 DC 3.3v AC 0 0
VSS     VSS 0 DC -3.3v AC 0 0
.ends

X_OTA1      0 VOUT2 VOUT1 OTA2
X_OTA2      VOUT1 VOUT2 VOUT2 OTA2
X_OTA3      0 VOUT4 VOUT3 OTA2
X_OTA4      VOUT3 VOUT4 VOUT4 OTA2
C_1         VIN1 VOUT1 OCAP1
C_1a        VOUT2 VOUT3 OCAP1
C_2         VOUT2 0 OCAP2
C_2a        VOUT4 0 OCAP2
V_IN        VIN1 0 DC 0 AC 1 0
RIN         VIN1 0 100MEG
```

A.6 Cascade 6

```
.subckt OTA3 VINP VINM VOUT Vbias1 Vbias2 Vbias3 Vbias4
M1      2 VINP 1 VSS nmos0553  L=0.35u W=OWM1
M2      3 VINM 1 VSS nmos0553  L=0.35u W=OWM1
M5      2 Vbias1 VDD VDD pmos0553  L=0.35u W=OWM2
M6      3 Vbias1 VDD VDD pmos0553  L=0.35u W=OWM2
M7      8 Vbias2 2 VDD pmos0553  L=0.35u W=OWM2
M8      VOUT Vbias2 3 VDD pmos0553  L=0.35u W=OWM2
M9      8 Vbias3 5 VSS nmos0553  L=0.35u W=OWM1
M10     VOUT Vbias3 6 VSS nmos0553  L=0.35u W=OWM1
M11     5 8 VSS VSS nmos0553  L=0.35u W=OWM1
M12     6 8 VSS VSS nmos0553  L=0.35u W=OWM1
M13     1 Vbias3 4 VSS nmos0553  L=0.35u W=OWM1
M14     4 Vbias4 VSS VSS nmos0553  L=0.35u W=OWM1
M15     5 VINP 7 VDD pmos0553  L=0.35u W=OWM2
M16     6 VINM 7 VDD pmos0553  L=0.35u W=OWM2
M17     9 Vbias1 VDD VDD pmos0553  L=0.35u W=OWM2
M18     7 Vbias2 9 VDD pmos0553  L=0.35u W=OWM2
VDD     VDD 0 DC 3.3v AC 0 0
VSS     VSS 0 DC -3.3v AC 0 0
.ends

.subckt BIASCCT Vbias1 Vbias2 Vbias3 Vbias4
MB1     12 12 Vbias4 VSS nmos0553  L=40u W=30u
MB10    Vbias3 Vbias4 VSS VSS nmos0553  L=0.35u W=OWM1
MB2     10 10 Vbias1 VDD pmos0553  L=10u W=35u
MB3     Vbias4 Vbias4 VSS VSS nmos0553  L=0.35u W=OWM1
MB4     12 10 11 VDD pmos0553  L=0.35u W=OWM2
MB5     Vbias1 Vbias1 VDD VDD pmos0553  L=0.35u W=OWM2
```

```
MB6     11 Vbias1 VDD VDD pmos0553  L=0.35u W=OWM2
MB7     Vbias2 Vbias1 VDD VDD pmos0553  L=0.35u W=OWM2
MB8     VSS 10 Vbias2 VDD pmos0553  L=0.35u W=OWM2
MB9     VDD 12 Vbias3 VSS nmos0553  L=0.35u W=OWM1
I       10 VSS DC OIB
VDD     VDD 0 DC 3.3v AC 0 0
VSS     VSS 0 DC -3.3v AC 0 0
.ends

X_BIASNET Vbias1 Vbias2 Vbias3 Vbias4 BIASCCT
X_OTA1       0 VOUT2 VOUT1 Vbias1 Vbias2 Vbias3 Vbias4 OTA3
X_OTA2       VOUT1 VOUT2 VOUT2 Vbias1 Vbias2 Vbias3 Vbias4 OTA3
X_OTA3       0 VOUT4 VOUT3 Vbias1 Vbias2 Vbias3 Vbias4 OTA3
X_OTA4       VOUT3 VOUT4 VOUT4 Vbias1 Vbias2 Vbias3 Vbias4 OTA3
C_1          VIN1 VOUT1 OCAP1
C_1a         VOUT2 VOUT3 OCAP1
C_2          VOUT2 0 OCAP2
C_2a         VOUT4 0 OCAP2
V_IN         VIN1 0 DC 0 AC 1 0
RIN          VIN1 0 100MEG
```

A.7 Coupled resonator

```
.subckt BPASS1 INP INM  COILP COILM VDD VSS
M1A COILM INP NIB VSS nmos0553 L=0.35U W=OWM1
M1B COILP INM NIB VSS nmos0553 L=0.35U W=OWM1
IBIAS1 NIB 0 OIB
M3A COILP COILM 3 VSS nmos0553 L=0.35U W=OWM1
M3B COILM COILP 3 VSS nmos0553 L=0.35U W=OWM1
IQCTL 3 0 OIQ
.ends

L1 OUTPUTP1 COILP1 Lind
RL1 COILP1 0 RL
C1 OUTPUTP1 0 CAP1

.subckt BPASS2 COIL2P COIL2M OUTP OUTM VDD VSS
M3AA COIL2P COIL2M N3 VSS nmos0553 L=0.35U W=OWM1
M3BB COIL2M COIL2P N3 VSS nmos0553 L=0.35U W=OWM1
IQCTL N3 0 OIQ
M5A OUTP COIL2M NIBB VSS nmos0553  L=0.35U W=OWM1
M5B OUTM COIL2P NIBB VSS nmos0553  L=0.35U W=OWM1
IBIAS2 NIBB 0 OIB
.ends

L2 INPUTP2 COIL2P1 Lind
RL2 COIL2P1 0 RL
C2 INPUTP2 0 CAP1

M8A VDD OUTPUTM1 NK1 VSS nmos0553 L=0.35U W=OWM1
RM9A NRCA NK1 1k
CM9A NRCA 0 200f
M9A INPUTP2 NRCA NK VSS nmos0553  L=0.35U W=OWM1
M9B INPUTM2 NRCB NK VSS nmos0553  L=0.35U W=OWM1
M8B VDD OUTPUTP1 NK2 VSS nmos0553 L=0.35U W=OWM1
RM9B NRCB NK2 1k
CM9B NRCB 0 200f
I_K NK 0 OIK
```

```
I_K1 NK1 0 OIK12
I_K2 NK2 0 OIK12
VDDSUP VDD 0 3.3V
VSSSUP VSS 0 -3.3V
VIN INPUTP1 0 AC 1
RIN INPUTP1 0 100meg
X_BP1 INPUTP1 0 OUTPUTP1 OUTPUTM1 VDD VSS BPASS1
X_BP2 INPUTP2 INPUTM2 OUTPUTP2 OUTPUTM2 VDD VSS BPASS2
k1 L1 L2 0.1
```

A.8 IFLF1

```
.subckt OTA VINP VINM VOUT
I1     VDD 9 DC OIB AC 0 0
I2     11 VSS DC OIB AC 0 0
M1      1 VINM 11 VSS nmos0553  L=0.35u W=OWM1
M2      3 VINP 11 VSS nmos0553  L=0.35u W=OWM1
M3      1 1 VDD VDD pmos0553  L=0.35u W=OWM2
M4      3 3 VDD VDD pmos0553  L=0.35u W=OWM2
M5      VOUT 3 VDD VDD pmos0553  L=0.35u W=OWM2
M6      7 1 VDD VDD pmos0553  L=0.35u W=OWM2
M7      VOUT 7 VSS VSS nmos0553  L=0.35u W=OWM1
M8      3 8 VSS VSS nmos0553  L=0.35u W=OWM1
M9      8 VINM 9 VDD pmos0553  L=0.35u W=OWM2
M10     7 VINP 9 VDD pmos0553  L=0.35u W=OWM2
M11     8 8 VSS VSS nmos0553  L=0.35u W=OWM1
M12     7 7 VSS VSS nmos0553  L=0.35u W=OWM1
VDD     VDD 0      DC 3.3v AC 0 0
VSS     VSS 0      DC -3.3v AC 0 0
.ends

X_OTA1      VIN1 VOUT4 VOUT1 OTA
X_OTA2      VOUT1 VOUT4 VOUT2 OTA
X_OTA3      VOUT2 VOUT4 VOUT3 OTA
X_OTA4      VOUT3 VOUT4 VOUT4 OTA
C_1         VOUT1 0 OCAP1
C_2         VOUT2 0 OCAP2
C_3         VOUT3 0 OCAP3
C_4         VOUT4 0 OCAP4
V_IN        VIN1 0 DC 0 AC 1 0
RIN         VIN1 0 100MEG
```

A.9 IFLF2

```
.subckt OTA VINP VINM VOUT
I1      4 VSS DC OIB AC 0 0
I2      1 VSS DC OIB AC 0 0
M1      2 VINP 1 VSS nmos0553  L=0.35u W=OWM1
M2      3 VINM 1 VSS nmos0553  L=0.35u W=OWM1
M3      5 5 VDD VDD pmos0553  L=0.35u W=OWM2
M4      4 4 5 VDD pmos0553  L=0.35u W=OWM2
M5      2 5 VDD VDD pmos0553  L=0.35u W=OWM2
M6      3 5 VDD VDD pmos0553  L=0.35u W=OWM2
M7      9 4 2 VDD pmos0553  L=0.35u W=OWM2
M8      VOUT 4 3 VDD pmos0553  L=0.35u W=OWM2
```

```
M9      9 9 7 VSS nmos0553  L=0.35u W=OWM1
M10     VOUT 9 8 VSS nmos0553  L=0.35u W=OWM1
M11     7 7 VSS VSS nmos0553  L=0.35u W=OWM1
M12     8 7 VSS VSS nmos0553  L=0.35u W=OWM1
VDD     VDD 0      DC 3.3v AC 0 0
VSS     VSS 0      DC -3.3v AC 0 0
.ends

X_OTA1       VIN1 VOUT4 VOUT1 OTA
X_OTA2       VOUT1 VOUT4 VOUT2 OTA
X_OTA3       VOUT2 VOUT4 VOUT3 OTA
X_OTA4       VOUT3 VOUT4 VOUT4 OTA
C_1          VOUT1 0 OCAP1
C_2          VOUT2 0 OCAP2
C_3          VOUT3 0 OCAP3
C_4          VOUT4 0 OCAP4
V_IN         VIN1 0 DC 0 AC 1 0
RIN          VIN1 0 100MEG
```

A.10 IFLF3

```
.subckt OTA VINP VINM VOUT Vbias1 Vbias2 Vbias3 Vbias4
M1      2 VINP 1 VSS nmos0553  L=0.35u W=OWM1
M2      3 VINM 1 VSS nmos0553  L=0.35u W=OWM1
M5      2 Vbias1 VDD VDD pmos0553  L=0.35u W=OWM2
M6      3 Vbias1 VDD VDD pmos0553  L=0.35u W=OWM2
M7      8 Vbias2 2 VDD pmos0553  L=0.35u W=OWM2
M8      VOUT Vbias2 3 VDD pmos0553  L=0.35u W=OWM2
M9      8 Vbias3 5 VSS nmos0553  L=0.35u W=OWM1
M10     VOUT Vbias3 6 VSS nmos0553  L=0.35u W=OWM1
M11     5 8 VSS VSS nmos0553  L=0.35u W=OWM1
M12     6 8 VSS VSS nmos0553  L=0.35u W=OWM1
M13     1 Vbias3 4 VSS nmos0553  L=0.35u W=OWM1
M14     4 Vbias4 VSS VSS nmos0553  L=0.35u W=OWM1
M15     5 VINP 7 VDD pmos0553  L=0.35u W=OWM2
M16     6 VINM 7 VDD pmos0553  L=0.35u W=OWM2
M17     9 Vbias1 VDD VDD pmos0553  L=0.35u W=OWM2
M18     7 Vbias2 9 VDD pmos0553  L=0.35u W=OWM2
VDD     VDD 0 DC 3.3v AC 0 0
VSS     VSS 0 DC -3.3v AC 0 0
.ends

.subckt BIASCCT Vbias1 Vbias2 Vbias3 Vbias4
MB1     12 12 Vbias4 VSS nmos0553  L=40u W=30u
MB10    Vbias3 Vbias4 VSS VSS nmos0553  L=0.35u W=OWM1
MB2     10 10 Vbias1 VDD pmos0553  L=10u W=35u
MB3     Vbias4 Vbias4 VSS VSS nmos0553  L=0.35u W=OWM1
MB4     12 10 11 VDD pmos0553  L=0.35u W=OWM2
MB5     Vbias1 Vbias1 VDD VDD pmos0553  L=0.35u W=OWM2
MB6     11 Vbias1 VDD VDD pmos0553  L=0.35u W=OWM2
MB7     Vbias2 Vbias1 VDD VDD pmos0553  L=0.35u W=OWM2
MB8     VSS 10 Vbias2 VDD pmos0553  L=0.35u W=OWM2
MB9     VDD 12 Vbias3 VSS nmos0553  L=0.35u W=OWM1
I       10 VSS DC OIB
VDD     VDD 0 DC 3.3v AC 0 0
VSS     VSS 0 DC -3.3v AC 0 0
.ends
```

```
X_OTA1      VIN1 VOUT4 VOUT1 Vbias1 Vbias2 Vbias3 Vbias4 OTA
X_OTA2      VOUT1 VOUT4 VOUT2 Vbias1 Vbias2 Vbias3 Vbias4 OTA
X_OTA3      VOUT2 VOUT4 VOUT3 Vbias1 Vbias2 Vbias3 Vbias4 OTA
X_OTA4      VOUT3 VOUT4 VOUT4 Vbias1 Vbias2 Vbias3 Vbias4 OTA
X_BIASNET   Vbias1 Vbias2 Vbias3 Vbias4 BIASCCT
C_1         VOUT1 0 OCAP1
C_2         VOUT2 0 OCAP2
C_3         VOUT3 0 OCAP3
C_4         VOUT4 0 OCAP4
V_IN        VIN1 0 DC 0 AC 1 0
RIN         VIN1 0 100MEG
```

A.11 LC-OTA-C

```
.subckt colp N5 N2
M_M1          N2 N5 N3F N4 nmos0553 L=0.35u  W=OWM3
L_L1          N7 N2  8nH
C_C1          N2 N3F  OCAP3
C_C2          N3F N4  OCAP4
I_I1          N3F N4 DC OIB
R_RL1         N7 N1  5
R_R           N2 N1  10k
V_VDD         N1 0 3.3V
V_VSS         0 N4 3.3V
.ends

.subckt OTA VINP VINM VOUT
I1      4 VSS DC OIB AC 0 0
I2      1 VSS DC OIB AC 0 0
M1      2 VINP 1 VSS nmos0553  L=0.35u W=OWM1
M2      3 VINM 1 VSS nmos0553  L=0.35u W=OWM1
M3      5 5 VDD VDD pmos0553  L=0.35u W=OWM2
M4      4 4 5 VDD pmos0553  L=0.35u W=OWM2
M5      2 5 VDD VDD pmos0553  L=0.35u W=OWM2
M6      3 5 VDD VDD pmos0553  L=0.35u W=OWM2
M7      9 4 2 VDD pmos0553  L=0.35u W=OWM2
M8      VOUT 4 3 VDD pmos0553  L=0.35u W=OWM2
M9      9 9 7 VSS nmos0553  L=0.35u W=OWM1
M10     VOUT 9 8 VSS nmos0553  L=0.35u W=OWM1
M11     7 7 VSS VSS nmos0553  L=0.35u W=OWM1
M12     8 7 VSS VSS nmos0553  L=0.35u W=OWM1
VDD     VDD 0 DC 3.3v AC 0 0
VSS     VSS 0 DC -3.3v AC 0 0
.ends

.subckt otabp VIN1 VOUT2
X_OTA1 0 VOUT2 VOUT1 OTA
X_OTA2 VOUT1 VOUT2 VOUT2 OTA
C_1    VIN1 VOUT1 OCAP1
C_2    VOUT2 0 OCAP2
.ends

X_BPCOLP INPUT1 OUTPUT1 colp
X_BPOTA  OUTPUT1 OUTPUT2  otabp
VINPUT   INPUT1 0 AC 1V
```

A.12 LF1

```
.subckt OTA VINP VINM VOUT
I1     VDD 9 DC OIB AC 0 0
I2     11 VSS DC OIB AC 0 0
M1      1 VINM 11 VSS nmos0553  L=0.35u W=OWM1
M2      3 VINP 11 VSS nmos0553  L=0.35u W=OWM1
M3      1 1 VDD VDD pmos0553  L=0.35u W=OWM2
M4      3 3 VDD VDD pmos0553  L=0.35u W=OWM2
M5      VOUT 3 VDD VDD pmos0553  L=0.35u W=OWM2
M6      7 1 VDD VDD pmos0553  L=0.35u W=OWM2
M7      VOUT 7 VSS VSS nmos0553  L=0.35u W=OWM1
M8      3 8 VSS VSS nmos0553  L=0.35u W=OWM1
M9      8 VINM 9 VDD pmos0553  L=0.35u W=OWM2
M10     7 VINP 9 VDD pmos0553  L=0.35u W=OWM2
M11     8 8 VSS VSS nmos0553  L=0.35u W=OWM1
M12     7 7 VSS VSS nmos0553  L=0.35u W=OWM1
VDD     VDD 0 DC 3.3v AC 0 0
VSS     VSS 0 DC -3.3v AC 0 0
.ends

X_OTA1       VIN1 VOUT2 VOUT1 OTA
X_OTA2       VOUT1 VOUT3 VOUT2 OTA
X_OTA3       VOUT2 VOUT4 VOUT3 OTA
X_OTA4       VOUT3 VOUT4 VOUT4 OTA
C_1          VOUT1 0 OCAP1
C_2          VOUT2 0 OCAP2
C_3          VOUT3 0 OCAP3
C_4          VOUT4 0 OCAP4
V_IN         VIN1 0 DC 0 AC 1 0
RIN          VIN1 0 100MEG
```

A.13 LF2

```
.subckt OTA VINP VINM VOUT
I1      4 VSS DC OIB AC 0 0
I2      1 VSS DC OIB AC 0 0
M1      2 VINP 1 VSS nmos0553  L=0.35u W=OWM1
M2      3 VINM 1 VSS nmos0553  L=0.35u W=OWM1
M3      5 5 VDD VDD pmos0553  L=0.35u W=OWM2
M4      4 4 5 VDD pmos0553  L=0.35u W=OWM2
M5      2 5 VDD VDD pmos0553  L=0.35u W=OWM2
M6      3 5 VDD VDD pmos0553  L=0.35u W=OWM2
M7      9 4 2 VDD pmos0553  L=0.35u W=OWM2
M8      VOUT 4 3 VDD pmos0553  L=0.35u W=OWM2
M9      9 9 7 VSS nmos0553  L=0.35u W=OWM1
M10     VOUT 9 8 VSS nmos0553  L=0.35u W=OWM1
M11     7 7 VSS VSS nmos0553  L=0.35u W=OWM1
M12     8 7 VSS VSS nmos0553  L=0.35u W=OWM1
VDD     VDD 0 DC 3.3v AC 0 0
VSS     VSS 0 DC -3.3v AC 0 0
.ends

X_OTA1       VIN1 VOUT2 VOUT1 OTA
X_OTA2       VOUT1 VOUT3 VOUT2 OTA
X_OTA3       VOUT2 VOUT4 VOUT3 OTA
```

```
X_OTA4      VOUT3 VOUT4 VOUT4 OTA
C_1         VOUT1 0 OCAP1
C_2         VOUT2 0 OCAP2
C_3         VOUT3 0 OCAP3
C_4         VOUT4 0 OCAP4
V_IN        VIN1 0 DC 0 AC 1 0
RIN         VIN1 0 100MEG
```

A.14 LF3

```
.subckt OTA VINP VINM VOUT Vbias1 Vbias2 Vbias3 Vbias4
M1      2 VINP 1 VSS nmos0553  L=0.35u W=OWM1
M2      3 VINM 1 VSS nmos0553  L=0.35u W=OWM1
M5      2 Vbias1 VDD VDD pmos0553  L=0.35u W=OWM2
M6      3 Vbias1 VDD VDD pmos0553  L=0.35u W=OWM2
M7      8 Vbias2 2 VDD pmos0553  L=0.35u W=OWM2
M8      VOUT Vbias2 3 VDD pmos0553  L=0.35u W=OWM2
M9      8 Vbias3 5 VSS nmos0553  L=0.35u W=OWM1
M10     VOUT Vbias3 6 VSS nmos0553  L=0.35u W=OWM1
M11     5 8 VSS VSS nmos0553  L=0.35u W=OWM1
M12     6 8 VSS VSS nmos0553  L=0.35u W=OWM1
M13     1 Vbias3 4 VSS nmos0553  L=0.35u W=OWM1
M14     4 Vbias4 VSS VSS nmos0553  L=0.35u W=OWM1
M15     5 VINP 7 VDD pmos0553  L=0.35u W=OWM2
M16     6 VINM 7 VDD pmos0553  L=0.35u W=OWM2
M17     9 Vbias1 VDD VDD pmos0553  L=0.35u W=OWM2
M18     7 Vbias2 9 VDD pmos0553  L=0.35u W=OWM2
VDD     VDD 0 DC 3.3v AC 0 0
VSS     VSS 0 DC -3.3v AC 0 0
.ends

.subckt BIASCCT Vbias1 Vbias2 Vbias3 Vbias4
MB1     12 12 Vbias4 VSS nmos0553  L=40u W=30u
MB10    Vbias3 Vbias4 VSS VSS nmos0553  L=0.35u W=OWM1
MB2     10 10 Vbias1 VDD pmos0553  L=10u W=35u
MB3     Vbias4 Vbias4 VSS VSS nmos0553  L=0.35u W=OWM1
MB4     12 10 11 VDD pmos0553  L=0.35u W=OWM2
MB5     Vbias1 Vbias1 VDD VDD pmos0553  L=0.35u W=OWM2
MB6     11 Vbias1 VDD VDD pmos0553  L=0.35u W=OWM2
MB7     Vbias2 Vbias1 VDD VDD pmos0553  L=0.35u W=OWM2
MB8     VSS 10 Vbias2 VDD pmos0553  L=0.35u W=OWM2
MB9     VDD 12 Vbias3 VSS nmos0553  L=0.35u W=OWM1
I       10 VSS DC OIB
VDD     VDD 0 DC 3.3v AC 0 0
VSS     VSS 0 DC -3.3v AC 0 0
.ends

X_OTA1      VIN1 VOUT2 VOUT1 Vbias1 Vbias2 Vbias3 Vbias4 OTA
X_OTA2      VOUT1 VOUT2 VOUT2 Vbias1 Vbias2 Vbias3 Vbias4 OTA
X_OTA3      VOUT2 VOUT4 VOUT3 Vbias1 Vbias2 Vbias3 Vbias4 OTA
X_OTA4      VOUT3 VOUT4 VOUT4 Vbias1 Vbias2 Vbias3 Vbias4 OTA
X_BIASNET   Vbias1 Vbias2 Vbias3 Vbias4 BIASCCT
C_1         VOUT1 0 OCAP1
C_2         VOUT2 0 OCAP2
C_3         VOUT3 0 OCAP3
C_4         VOUT4 0 OCAP4
V_IN        VIN1 0 DC 0 AC 1 0
RIN         VIN1 0 100MEG
```

A.15 Vertical Cascode

```
M_M5          N21 N6 N16 0 nmos0553  L=0.35u   W=OWM1
V_Viout5      N2 N21 0V
R_R3          0 N6  100Meg
R_R4          0 N3  100Meg
V_VS2         N16 0 0.65V
V_VS1         N11 0 0.3V
I_Io44        N1 N5 DC OI4
I_Io22        N1 N10 DC OI2
V_VDD         N1 0 2.4V
M_M55         N5 N3 N16 0 nmos0553  L=0.35u   W=OWM1
M_M44         iout11 N5 N3 0 nmos0553  L=0.35u  W=OWM1
I_Io4         N1 N2 DC OI4
V_Viout11     iout11 N1 0V
I_Io333       N13 0 DC OI33
M_M11         N3 N10 N8 0 nmos0553  L=0.35u  W=OWM1
M_M22         N10 N8 N11 0 nmos0553  L=0.35u  W=OWM1
I_Io11        N8 0 DC OI1
R_R2          0 N8  100Meg
Iiin2         N8 0 DC 0A AC 0.5A
M_M4          iout1 N2 N6 0 nmos0553  L=0.35u  W=OWM1
V_Viout1      N1 iout1 0V
M_M3          N8 N7 N13 0 nmos0553  L=0.35u  W=OWM1
I_Io33        N1 N7 DC OI3
M_M33         N7 N8 N13 0 nmos0553  L=0.35u  W=OWM1
I_Io1         N7 0 DC OI1
I_iin1        0 N7  DC 0A AC 0.5A
R_R1          0 N7  100Meg
M_M1          N6 N14 N7 0 nmos0553  L=0.35u  W=OWM1
I_Io2         N1 N14 DC OI2
I_Io3         N1 N8 DC OI3
M_M2          N14 N7 N11 0 nmos0553  L=0.35u  W=OWM1
```

Appendix B: Alcatel CMOS 0.35μm BSIM 3v3 transistor models

B.1 NMOS transistor model

```
.model              nmos0553 nmos level   =53
+version=3.2
+tnom=27             tox=6.8e-9           xj=2.3e-7
+nch=2e17          vth0=0.5410286
+k1=0.5538175        k2=-2.01788e-5      k3=43.254121
+k3b=-8.3666578           w0=5.7493e-6      nlx=1.72968e-7
+dvt0w=0.018702          dvt1w=5.3e6 dvt2w=-0.032
+dvt0=4.10248      dvt1=0.4697625      dvt2=-0.05
+u0=524.74      ua=1.476303e-9
+ub=2.083775e-19      uc=5.368193e-11     vsat=8.83e4
+a0=0.8775883         ags=0.214565       b0=4.40815e-8
+b1=1e-7      keta=0.0166414   a1=0
+a2=1              rdsw=792    prwg=9.336953e-4
+prwb=0.0539535     wr=1                wint=1.572104e-8
+lint=5.15e-8    dwg=-2.687564e-9    dwb=4.696235e-9
+voff=-0.1406745     nfactor=1.4442501   cit=0
+cdsc=1e-3              cdscd=0              cdscb=0
+eta0=0            etab=-0.0722136         dsub=0.56
+pclm=0.8351951      pdiblc1=0.2896433   pdiblc2=2.920887e-3
+pdiblcb=0    drout=0.7796106      pscbe1=6.510097e8
+pscbe2=2.948305e-5         pvag=0.0587596        delta=1.618913e-3
+alpha0=2.2e-7              beta0=18.45
+alpha1=0.78      rsh=2.7      js=6.4e-7         jsw=1.6e-12
+mobmod=1            prt=0      ute=-1.7395947
+kt1=-0.1635661      kt1l=-1.173597e-8   kt2=0.022
+ua1=1.081907e-10     ub1=-8.22235e-19    uc1=-1e-10
+at=3.3e4      elm=5   wl=9.246632e-22   wln=1        ww=0
+wwn=1               wwl=-1.28698e-20
+ll=0                lln=1
+lw=0                lwn=1                lwl=0
+kf=3.9167e-28         af=1
+noimod=1          ef=1  capmod=3
+xpart=0
+cgdo=1.1274e-10  cgso=1.1274e-10
+cj=8.65e-4
+pb=0.904   mj=0.369
+cjsw=2.43e-10          pbsw=0.894  mjsw=0.356
+cjswg=2.678e-10
+pbswg=0.896            mjswg=0.356       ckappa=0.6
+clc=1e-8   cle=0.6
+noff=1     acde=1
+moin=15    tpb=0 tpbsw=0
+tpbswg=0   tcj=0 tcjsw=0
+tcjswg=0
```

B.2 PMOS transistor model

```
.model              pmos0553 pmos level=53
+version=3.2
+tnom=27    tox=6.8e-9  xj=3e-7
+nch=2.8e17
+vth0=-0.5445258
+k1=0.64566        k2=-8.459721e-5      k3=1e-3
+k3b=1.27313          w0=9.9e-6           nlx=7.315701e-8
+dvt0w=0.21      dvt1w=3.325e5      dvt2w=-0.0455073
+dvt0=7.6469      dvt1=0.74924           dvt2=-0.0356391
+u0=119.56      ua=1.736799e-9
+ub=2.067129e-20     uc=-5.97265e-11     vsat=1.510148e5
+a0=1.2275109        ags=0.1823146       b0=4.165587e-8
+b1=1e-7      keta=-9.803756e-3     a1=0
+a2=1              rdsw=1.938327e3    prwg=4.254056e-3
+prwb=6.047599e-3     wr=1               wint=1.931593e-8
+lint=5.318836e-8 dwg=-3.547194e-9              dwb=9.874385e-9
+voff=-0.1246442     nfactor=0.6522602   cit=0
+cdsc=2.4e-4             cdscd=0             cdscb=0
+eta0=0.8389729      etab=-0.07     dsub=1
+pclm=1.1132227      pdiblc1=0.0192481 pdiblc2=6.943741e-4
+pdiblcb=0       drout=0.999             pscbe1=7.992e8
+pscbe2=1.001e-5         pvag=0.1810426     delta=0.0131564
+ngate=0    alpha0=1.786e-7            beta0=28.5
+alpha1=0.42      rsh=2.4           js=7.64e-7
+jsw=2.744e-12 mobmod=1
+prt=350.3826014     ute=-1.2024509        kt1=-0.1263843
+kt11=-3.483014e-8   kt2=0           ua1=8.676042e-14
Chapter 1            +ub1=-1.88675e-18    uc1=-1e-10  at=3.3e4
+elm=5    wl=9.350479e-20             wln=1              ww=0
+wwn=1      wwl=-1.04839e-20
+ll=0                lln=1
+lw=0                lwn=1              lwl=0
+af=1             kf=2.166e-28
+noimod=1   ef=1  capmod=3
+xpart=0    cgdo=1.1274e-10   cgso=1.1274e-10
+cj=1.23e-3
+pb=0.908   mj=0.442   cjsw=2.2e-10
+pbsw=0.899 mjsw=0.373  cjswg=2.30e-10
+pbswg=0.904      mjswg=0.411 clc=1e-7 cle=0.6
```

References

[1]. *IEEE standard VHDL analog and mixed-signal extensions.* 23 Dec. 1999, Design Automation Standards Committee of the IEEE Computer Society.

[2]. Schaumann, R. and M.E.V. Valkenburg, *Design of Analog Filters.* 2001: Oxford University Press.

[3]. Baker, R., H. Li, and D.E. Boyce, *CMOS circuit design, layout, and simulation.* IEEE Press series on Microelectronic systems. 1998.

[4]. Phelps, R., et al., *Anaconda: simulation-based synthesis of analog circuits via stochastic pattern search.* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2000. **19**(6): p. 703 -717.

[5]. Kazmierski, T.J. and F.A. Hamid. *Analogue circuit synthesis from VHDL-AMS (invited paper).* in *FDL 2000.* 2000.

[6]. Hamid, F.A. and T.J. Kazmierski, *Analogue circuit synthesis from VHDL-AMS*, R. Seepold, Editor. 2001, Kluwer Academic Publishers: Boston.

[7]. Hamid, F. and T. Kazmierski. *Analog filter synthesis from VHDL-AMS.* in *FDL'01.* 2001. Lyon.

[8]. Hamid, F.A. and T.J. Kazmierski. *Synthesis And Optimization Of Analog VLSI Filters From VHDL-AMS Parse Trees.* in *ISCAS'2002.* 2002.

[9]. Hamid, F.A. and T.J. Kazmierski, *Analog filter synthesis from VHDL-AMS*, L. Spruiell, Editor. 2002, Kluwer Academic Publishers: Boston.

[10]. Hamid, F. and T.J. Kazmierski. *FIST - a VHDL-AMS based architectural synthesis strategy for integrated high-frequency analogue filters.* in *FDL.* 2003. Frankfurt, Germany.

[11]. Kazmierski, T.J. and F.A. Hamid. *Behavioral Modelling of RF Filters in VHDL-AMS for Automated Architectural and Parametric Optimization.* in *ISCAS 2004.* 2004. Vancouver, Canada.

[12]. Gielen, G. and R.Rutenbar, *Computer-Aided Design of Analog and Mixed-Signal Integrated Circuits.* Proceedings of the IEEE, 2000. **88**(12): p. 1825-1852.

[13]. Oehler, P., C. Grimm, and K.Waldschmidt. *KANDIS - A Tool for Construction of Mixed Analog/Digital Systems.* in *European Design Automation Conference.* 1995. Brighton, UK.

[14]. Vemuri, R., et al. *Analog System Performance Estimation in the VASE.* in *EETimes Analog And Mixed-Signal Applications Conference.* 1998.

[15]. Lopez, J.A., et al. *Automated High Level Synthesis of Hardware Building Blocks Present in art-based neural network, from VHDL-AMS descriptions.* in *ISCAS 2002.* 2002.

[16]. Doboli, A. and R. Vemuri, *Behavioural Modelling for HIgh-Level Synthesis of Analog and Mixed-Signal Systems from VHDL-AMS.* IEEE Transactions on CAD of Integrated Circuits and Systems, 2003. **22**(11): p. 1504-1520.

[17]. Grimm, C. and K. Waldschmidt. *Repartitioning and Technology Mapping of Electronic Hybrid Systems*. in *Design, Automation and Test in Europe, 1998.,*. 1998.

[18]. *Mixed-Signal Synthesis (VASE), Digital Design Environments Laboratory, University of Cincinnati:http://www.ececs.uc.edu/~ddel/.*

[19]. Nunez-Aldana, A. and R. Vemuri. *A Hierarchical Modeling of Analog CMOS Components for Synthesis*. in *Proceedings of the IEEE/VIUF International Workshops on Behavioral Modeling and Simulation (BMAS'98)*. 1998.

[20]. Nunez-Aldana, A., et al. *A Methodology for Behavioral Synthesis of Analog Systems*. in *Proceedings of the IEEE Southwest Symposium on Mixed-Signal Design*. 1999. Tucson, Arizona.

[21]. Aldana, A.-N., A. Doboli, and R. Vemuri. *A Top-down Synthesis Methodology for Behavioral Mixed-Signal Systems Specified in VHDL-AMS*. in *Proceedings of Second International Workshop on Design of Mixed-Mode Integrated Circuits and Applications*. 1998: IEEE Press.

[22]. Doboli, A. and R. Vemuri. *The Definition of a VHDL-AMS Subset for Behavioral Synthesis of Analog Systems*. in *1998 IEEE/VIUF International Workshop on Behavioral Modeling and Simulation (BMAS'98)*. 1998.

[23]. Doboli, A., et al. *Behavioral Synthesis of Analog Systems using Two-Layered Design Space Exploration*. in *36th Design Automation Conference*. 1999.

[24]. Nunez-Aldana, A. and R. Vemuri. *An Analog Performance Estimator for Improving the Effectiveness of CMOS Analog Systems Circuit Synthesis*. in *Design, Automation and Test in Europe (DATE), Conference Proceedings*. 1999.

[25]. Doboli, A. and R. Vemuri. *A VHDL-AMS compiler and architecture generator for behavioral synthesis of analog systems*. in *DATE*. 1999.

[26]. Ganesan, S. and R. Vemuri. *A Methodology for Rapid Prototyping of Analog Systems*. in *International Conference on Computer Design (ICCD'99)*. 1999: IEEE Computer Society.

[27]. Ganesan, S., et al. *Rapid Prototyping of Mixed Signal Systems from VHDL-AMS*. in *Proc. Intl. Workshop on Beh. Modeling and Simulation (BMAS'99)*. 1999.

[28]. Ghosh, A. and R. Vemuri. *Formal Verification of Synthesized Analog Designs*. in *International Conference on Computer Design (ICCD'99)*. 1999: IEEE Computer Society.

[29]. Domenech-Asensi, G. and T.J. Kazmierski, *Automated synthesis of high-level VHDL-AMS analog descriptions*. 2000.

[30]. Doménech-Asensi, G., R. Ruiz-Merino, and T.J. Kazmierski. *Automatic synthesis of analog systems using a VHDL-AMS to HSPICE translator*. in *DCIS'2000*. 2000. Montpelier.

[31]. Domenech-Asensi, G., et al., *Architectural synthesis of high-level analogue VHDL-AMS descriptions using netlist extraction from parse trees*. Electronics Letters, 2000. **36**(20): p. 1680 -1682.

[32]. Antao, B.A.A. and A.J. Brodersen, *Techniques for Synthesis of Analog Integrated Circuits*, in *IEEE Design & Test of Computers*. 1992.

[33]. Krasnicki, M.J., et al. *ASF: A Practical Simulation-Based Methodology for the Synthesis of Custom Analog Circuits*. in *ICCAD*. 2001.

[34]. Berkcan, E., M. d'Abreu, and W. Laughton. *Analog compilation based on successive decompositions*. in *25th ACM/IEEE Design Automation Conference*. 1988.

[35]. Fung, A.H., et al. *Knowledge based analog circuit synthesis with flexible architecture*. in *Proceedings of the 1988 IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD '88)*. 1988.

[36]. Harjani, R., R.Rutenbar, and L. Carley, *OASYS: a framework for analog circuit synthesis*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1989. **8**(12): p. 1247-1266.

[37]. El-Turky, F. and E.E. Perry, *BLADES: An Artificial Intelligence Approach to Analog Circuit Design*. IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, 1989. **8**(6): p. 680-692.

[38]. Makris, C.A., et al. *CHIPAIDE: A New Approach to Analogue Integrated Circuit Design*. in *IEE Colloqium on Analogue VLSI*. 1990.

[39]. Harvey, J.P., M.I. Elmasry, and B. Leung, *STAIC: an interactive framework for synthesizing CMOS and BiCMOS analog circuits*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1992. **11**(11): p. 1402 -1417.

[40]. Degrauwe, M.G.R., et al., *IDAC: An Interactive Design Tool for Analog Circuits*. IEEE Journal of Solid-State Circuits, 1987. **22**(6): p. 1106-1115.

[41]. Jongsma, J., et al. *An open design tool for analog circuits*. in *IEEE International Symposium on Circuits and Systems*. 1991.

[42]. Koh, H.Y., C.H. Sequin, and P.R. Gray, *OPASYN: A Compiler for CMOS Operational Amplifiers*. IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, 1990. **9**(2): p. 113-125.

[43]. Onodera, H., H. Kanbara, and K. Tamaru, *Operational-amplifier compilation with performance optimization*. IEEE Journal of Solid-State Circuits, 1990. **25**(2): p. 466-473.

[44]. Hershenson, M., S. Boyd, and T. Lee. *GPCAD: A tool for CMOS op-amp synthesis*. in *1998 IEEE/ACM International Conference on Computer-Aided Design (ICCAD 98)*. 1998.

[45]. DeGrauwe, M., et al. *The ADAM analog design automation system*. in *IEEE International Symposium on Circuits and Systems*. 1990.

[46]. Carley, L.R., et al. *ACACIA: The CMU analog design system*. in *1989 IEEE Custom Integrated Circuits Conf.* 1989.

[47]. Plas, G.V.d., *AMGIE-A synthesis environment for CMOS analog integrated circuits*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2001. **20**(9): p. 1037 -1058.

[48]. Gielen, G. and W. Sansen, *Symbolic Analysis for Automated Design of Analog Integrated Circuits*. 1991: Kluwer Academic Publishers.

[49]. Harjani, R., R.A. Rutenbar, and L.R. Carley. *Analog circuit synthesis for performance in OASYS*. in *Computer-Aided Design, 1988. ICCAD-88. Digest of Technical Papers., IEEE International Conference on ,*. 1988.

[50]. Garrod, D., R.A. Rutenbar, and L.R. Carley. *Automatic layout of custom analog cells in ANAGRAM*. in *IEEE International Conference on Computer-Aided Design (ICCAD-88)*. 1988.

[51]. Gielen, G., H. Walscharts, and W. Sansen, *ISAAC: A symbolic simulator for analog integrated circuits*. IEEE Journal of Solid-State Circuits, 1989. **24**(6): p. 1587-1597.

[52]. Gielen, G., K. Swings, and W. Sansen. *An Intelligent Design System for Analogue Integrated Circuits*. in *European Design Automation Conference*. 1990.

[53]. Gielen, G., H. Walscharts, and W. Sansen, *Analog Circuit Design Optimization Based on Symbolic Simulation and Simulated Annealing.* IEEE Transaction on Solid-State Circuits, 1990. **25**(3): p. 707-713.

[54]. Swings, K., G. Gielen, and W. Sansen. *An intelligent analog IC design system based on manipulation of design equations.* in *IEEE Custom Integrated Circuits Conference.* 1990.

[55]. Ochotta, E., *The OASYS virtual machine: Formalizing the OASYS analog synthesis framework.* 1989, Carnegie Mellon Univ.

[56]. Rijmenants, J., et al., *ILAC: an automated layout tool for analog CMOS circuits.* IEEE Journal of Solid-State Circuits, 1989. **24**(2): p. 417 -425.

[57]. Harjani, R., R.A. Rutenbar, and L.R. Carley. *Analog circuit synthesis and exploration in OASYS.* in *Computer Design: VLSI in Computers and Processors, 1988. ICCD '88., Proceedings of the 1988 IEEE International Conference on , 1988.* 1988.

[58]. Young, K.H., C.H. Sequin, and P.R. Gray. *Automatic layout generation for CMOS operational amplifiers.* in *Computer-Aided Design, 1988. ICCAD-88. Digest of Technical Papers., IEEE International Conference on ,.* 1988.

[59]. Maulik, P.C., L.R. Carley, and R.A. Rutenbar. *A mixed-integer nonlinear programming approach to analog circuit synthesis.* in *29th ACM/IEEE Design Automation Conference.* 1992.

[60]. Torralba, A., J. Chavez, and L.G. Franquelo, *FASY: A Fuzzy-Logic Based Tool for Analog Synthesis.* IEEE Transactions on Computer-Aided Design of Integrated Circtuits and Systems, 1996. **15**(7): p. 705-715.

[61]. Fares, M. and B. Kaminska, *FPAD: A Fuzzy Nonlinear Programming Approach to Analog Circuit Design.* IEEE Transactions on CAD of Integrated Circuits and Systems, 1995. **14**(7): p. 785-793.

[62]. Ochotta, E.S., R.A. Rutenbar, and L.R. Carley, *Synthesis of High-Performance Analog Circuits in ASTRX/OBLX.* Computer-Aided Design of Integrated Circuits and Systems, 1996. **15**(3): p. 273-294.

[63]. Nye, W., et al., *DELIGHT.SPICE: an optimization based system for the design of integrated circuits.* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1988. **7**(4): p. 501-519.

[64]. Krasnicki, M., et al. *MAELSTROM: Efficient Simulation-based synthesis for custom analog cells.* in *ACM/IEEE 36th Design Automation Conference.* 1999.

[65]. Phelps, R., et al. *ANACONDA: Robust Synthesis of Analog Circuits via Stochastic Patern Search.* in *IEEE Custom Integrated Circuits.* 1999.

[66]. Kirkpatrick, S., C.D. Gelatt, and M.P.J. Vecchi, *Optimization by Simulated Annealing.* Science, 1983. **220**(4598): p. 671-680.

[67]. Lam, Y.Y.H., *Synthesis of Analogue Circuits*, in *Department of Electronics and Computer Science.* 2001, University of Southampton. p. 248.

[68]. Alpaydin, G., S. Balkir, and G. Dundar, *An Evolutionary Approach to Automatic Synthesis of High-Performance Analog Itegrated Circuits.* IEEE Transactions on Evolutionary Computation, 2003. **7**(3): p. 240-252.

[69]. Antao, B. and A. Brodersen, *ARCHGEN: Automated Synthesis of Analog Systems.* IEEE Transactions on VLSI, 1995. **3**(2): p. 231-244.

[70]. De Smedt, B. and G.G.E. Gielen, *Watson: design space boundary exploration and model generation for analog and RF IC design.* Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, Feb 2003. **20**(9): p. 213- 224.

[71]. Doboli, A. and R. Vemuri, *Exploration-based High-Level Synthesis of Linear Analog Systems Operating at Low/Medium Frequencies.* IEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2003. **22**(11): p. 1556-1568.

[72]. Hjalmarson, E., R. Hägglund, and L. Wanhammar. *Optimization-Based Design Space Exploration of Analog Circuits.* in *Proc. European Conference on Circuit Theory and Design.* 2003. Krakow, Poland.

[73]. Hjalmarson, E., Robert Hägglund, and L. Wanhammar. *An Equation-Based Optimization Approach for Analog Circuit Design.* in *Proc. Int. Symp. on Signals, Circuits & Systems.* 2003. Iasi, Romania.

[74]. Ray, B.N., P.P. Chaudhuri, and P.K. Nandi, *Efficient synthesis of OTA network for linear analog functions.* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2002. **21**(5): p. 517 -533.

[75]. Wawryn, K. *PROLOG-based active filter synthesis.* in *European Conference on Circuit Theory and Design.* 1989.

[76]. Huelsman, L.P. *Filter synthesis using multi-dimensional search techniques.* in *33rd Midwest Symposium on Circuits and Systems.* 1990.

[77]. Barua, A., *Obtaining Expert Advice*, in *Circuits & Devices.* 1999. p. 17-25.

[78]. Barua, A. and S. Sinha. *CHOFIL: a knowledge based approach to active biquad synthesis.* in *1994 IEEE Asia-Pacific Conference on Circuits and Systems (APCCAS '94).* 1994.

[79]. Younis, A.T. and R.E. Massara, *Automated synthesis of switched-capacitor ladder filters within an analogue silicon compilation environment.* IEE Proceedings of Circuits, Devices and Systems, 1992. **139**(2): p. 249-255.

[80]. Barua, A. and M.K. Chandrakar. *SYSCUF: automated synthesis of switched current filter.* in *The 7th IEEE International Conference on Electronics, Circuits and Systems.* 2000.

[81]. Alpaydin, G., et al., *Multi-level optimisation approach to switched capacitor filter synthesis.* IEE Proceedings of Circuits, Devices and Systems, 2000. **147**(4): p. 243-249.

[82]. *VHDL Language Reference Manual*: IEEE Standard 1076-1993.

[83]. Christen, E. and K. Bakalar, *VHDL-AMS-a hardware description language for analog and mixed-signal applications.* IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, 1999. **46**(10): p. 1263 -1272.

[84]. KasulaSrinivas, V.R., *Modeling Semiconductor Devices using the VHDL-AMS Languange*, in *Department of Electrical and Computer Engineering and Computer Science.* 1999, University of Cincinnati.

[85]. Kasulasrinivas, V.R. and H.W. Carter. *Modeling and simulating semiconductor devices using VHDL-AMS.* in *Behavioral Modeling and Simulation (BMAS).* 2000.

[86]. *VHDL-AMS composite system modeling tutorials and guidelines* *http://www.ee.duke.edu/research/IMPACT/documents/mode_g.pdf.*

[87]. Alcantud, J. and T.J. Kazmierski. *VHDL-AMS modeling of self-organizing neural systems.* in *International Symposium of Circuits and Systems 2000 (ISCAS 2000).* 2000.

[88]. *http://www.mentor.com/ams/,* Mentor Graphics.

[89]. *Mast modelling. http://www.analogy.com*

[90]. *Standard Description Language Based on the Verilog(TM) Hardware Description Language*: IEEE Standard 1364-1995.

[91]. Miller, I. and T. Cassagnes. *Verilog-A and Verilog-AMS provides a new dimension in modeling and simulation*. in *Devices, Circuits and Systems, 2000. Proceedings of the 2000 Third IEEE International Caracas Conference on , 2000*. 2000.

[92]. K. Einwich, et al. *SystemC extension to mixed-signal design*. in *FDL01*. 2001. Lyon, France.

[93]. *VHDL-AMS Frontend (Java-Version) http://www.ti.informatik.uni-frankfurt.de/grimm/hybrid.html#VHDL-AMS*. 1997.

[94]. *Southampton University VHDL-AMS Validation Suite http://www.syssim.ecs.soton.ac.uk*. 1997.

[95]. *VHDL-AMS Analyzer*. 1997, Electronic Design Automation Research Center, Distributed Processing Laboratory, University of Cincinnati, http://www.ececs.uc.edu/~ddel.

[96]. *VHDL-AMS compiler, http://worldserver.oleane.com/leda*. 1997, LEDA S.A.

[97]. *VeriasHDL Simulator*. 1999, Analogy.

[98]. *VHDL-AMS Design Station User Manual*. 1999, Mentor Graphics.

[99]. *SMASH mixed-signal simulator http://www.matricsgroup.com/smash.asp*.

[100]. *hAMSter mixed-signal simulator www.hamster-ams.com*.

[101]. Valkenburg, M.E.V., *Analog Filter Design*. 1982: CBS College Publishing.

[102]. Zverev, A.I., *Handbook of filter synthesis*. 1967: Wiley.

[103]. El-licy, F.A. and H.S. Abdel-Aty-Zohdy. *Verification system interface for VLSI combinational circuits*. in *1998 Midwest Symposium on Circuits and Systems*. 1998.

[104]. Tanaka, T., *Parsing electronic circuits in a logic grammar*. IEEE Transactions on Knowledge and Data Engineering, 1993. **5**(2): p. 225 -239.

[105]. Tanaka, T. and L.C. Jain. *Circuit representation in a logic grammar*. in *Electronic Technology Directions to the Year 2000*. 1995.

[106]. Kazmierski, T.J. and C. Chalk. *A web-based VHDL-AMS design environment*. in *IEEE/VIUF BMAS'98*. 1998.

[107]. Aho, A.V., R. Sethi, and J.D. Ullman, *Compilers principles techniques,and tools*. 1986: Addison-Wesley.

[108]. Carlson, G.E., *Signal and linear system analysis*. 1992: Houghton Mifflin Company.

[109]. Press, W.H., et al., *Numerical Recipes in C - The art of scientific computing*. 1988: Cambridge University Press.

[110]. *BSIM3v3 Manual*, Department of Electrical Engineering and Computer Science, University of California, Berkeley.

[111]. Jia-Jiunn Ou, et al. *CMOS RF modeling for GHz communication IC's*. in *1998 Symposium on VLSI Technology*. 1998.

[112]. Kuhn, W.B., A. Elshabini-Riad, and F.W. Stephenson, *Centre-tapped spiral inductors for monolithic bandpass filters*. Electronic Letters, 1995. **31**(8): p. 625-626.

[113]. Li, D. and Y. Tsividis, *Active LC filters on silicon*. IEE Proceedings of Circuits, Devices and Systems, 2000. **147**(1): p. 49-56.

[114]. Burghartz, J.N., et al., *RF circuit design aspects of spiral inductors on silicon*. IEEE Journal of Solid-State Circuits, 1998. **33**(12): p. 2028-2034.

[115]. Yue, C.P., et al. *A Physical Model for Planar Spiral Inductors on Silicon*. in *International Electron Devices Meeting, 1996*. 1996.

[116]. Yue, C.P. and S.S. Wong, *Physical modeling of spiral inductors on silicon.* IEEE Transactions on Electron Devices, 2000. **47**(3): p. 560 -568.

[117]. Lee, T.H., *The Design of CMOS Radio-Frequency Integrated Circuit.* 1998: Cambridge University Press.

[118]. Yodprasit, U. and K. Sirivathanant. *A Compact Low-Power Vertical Filter for Very-High-Frequency Applications.* in *The 2001 IEEE International Symposium on Circuits and Systems (ISCAS 2001).* 2001.

[119]. *Star-HSPICE Manual.* 2000, Avant!

[120]. Nelder, A. and R. Mead, *A Simplex Method for Function Minimization.* Computer Journal, 1965. **7**: p. 308-313.

[121]. Massara, R.E., *Optimization methods in electronic circuit design.* Longman Scientific & Technical. 1991: Longman Group UK Limited.

[122]. *Programming Analysis and Optimization, APLAC 7.61, Reference manual, vol. 1.* 2001, Aplac Solutions Corp.

[123]. Walker, R., et al. *Circuit Optimization Using the Simplex Algorithm.* in *Hewlett-Packard 1989 VLSI Design Technology Conference.* 1989.

[124]. Sun, Y., ed. *Design of High-frequency integrated Analogue Filters.* 2002, IEE Press (UK).

[125]. Sun, Y. and J.K. Fidler, *Synthesis and performance analysis of universal minimum component integrator-based IFLF OTA-grounded capacitor filter.* IEE Proceedings of Circuits, Devices and Systems, 1996. **143**(2): p. 107-114.

[126]. Al-Hashimi, B.M., et al., *Integrated universal biquad based on triple-output OTAs and using digitally programmable zero.* IEE Proceedings of Circuits, Devices and Systems, 1998. **145**(3): p. 192-196.

[127]. Kuhn, W.B., F.W. Stephenson, and A. Elshabini-Riad, *A 200 MHz CMOS Q-Enhanced LC Bandpass Filter.* IEEE Journal Solid-State Circuits, 1996. **31**(8): p. 1112-1121.