

**THE UNIVERSITY OF HULL**

**Automatic Generation of High Level Fault  
Simulation Models for Analogue Circuits**

being a Thesis submitted for the Degree of

**Doctor of Philosophy**

in the University of Hull

by

Likun Xia, MSc.

November, 2008

# *Acknowledgements*

The author would like to express his deepest gratitude to the following people:

Dr. I.M. Bell, the first supervisor, for his guidance, support and useful discussion throughout the research.

Dr. A.J. Wilkinson, the second supervisor, for his encouragement and useful discussion regarding development of the automated model generation system during the research.

Thanks are also due to the postgraduate and academic staff of the Department of Engineering, Hull University for their support.

The author wishes to give most special gratitude to his wonderful parents for their encouragement, tolerance and their financial support during these years in UK.

And last but not least, the author wishes to thank his friends around the world, especially to Miss Ning Zhang, for their encouragement and support throughout.

# *Abstract*

High level modelling (HLM) for operational amplifiers (op amps) has been previously carried out successfully using models generated by published automated model generation (AMG) approaches. Furthermore, high level fault modelling (HLFM) has been shown to work reasonably well using manually designed fault models. However, no evidence shows that published AMG approaches based on op amps have been used in HLFM.

This thesis describes an investigation into the development of adaptive self-tuning algorithms for automated analogue circuit modelling suitable for HLM and HLFM applications. The algorithms and simulation packages were written in MATLAB and the hardware description language - VHDL-AMS.

The properties of these self-tuning algorithms were investigated by modelling a two-stage CMOS op amp and a comparator, and comparing simulations of the macromodel against those of the original SPICE circuit utilizing transient analysis.

The proposed algorithms generate multiple models to cover a wide range of input conditions by detecting nonlinearity through variations in output error, and can achieve bumpless transfer between models and handle nonlinearity.

This thesis describes the design, implementation and validation of these algorithms, their performance being evaluated for HLFM for both analogue and mix mode systems.

HLFM results show that the models can handle both linear and nonlinear situations with good accuracy in a low-pass filter, and model digital outputs in a flash ADC correctly. Comparing with a published fault model, better accuracy has been achieved in terms of output signals using fault coverage measurement.

# Table of Contents

<b>Acknowledgements</b> .....	<b>i</b>
<b>Abstract</b> .....	<b>ii</b>
<b>Table of Contents</b> .....	<b>iii</b>
<b>Table of Figures</b> .....	<b>iv</b>
<b>List of Terminologies</b> .....	<b>v</b>
<b>Chapter 1: Introduction</b> .....	<b>1-1</b>
<b>Chapter 2: Literature Review of Approaches for High Level Modelling and Automated Model Generation</b> .....	<b>2-1</b>
2.1 Review of Possible Models of a System .....	2-1
2.2 Review of Various Modelling Techniques .....	2-3
2.2.1 <i>Macromodeling (structural approach)</i> .....	2-3
2.2.2 <i>Behavioural modelling</i> .....	2-4
2.2.3 <i>Transfer function modelling</i> .....	2-4
2.3 Review of Automated Model Generation (AMG) Approaches .....	2-5
2.4 Review of Approaches to Generating a Model in Different Computing Languages .....	2-18
<b>Chapter 3: Literature Review of High Level Fault Modelling and Simulation</b> .....	<b>3-1</b>
3.1 Different Test Techniques for Analogue and Mixed Mode Circuits .....	3-1
3.1.1 <i>Description of IC Failure Mechanism and Defect Analysis</i> .....	3-1
3.1.2 <i>Structural Test</i> .....	3-4
3.2 High Level Fault Modelling and Simulation .....	3-6
3.3 Inductive Fault Analysis (IFA) .....	3-10
3.4 Design for Testability with Controllability and Observability (Design and quality issue) .....	3-13
3.5 Test Coverage and Test Quality .....	3-14

<b>Chapter 4: High Level Fault Modelling and Simulation based on Other's Fault Models .....</b>	<b>4-1</b>
4.1 Introduction .....	4-1
4.2 Two-stage CMOS Op amp .....	4-1
4.3 High Level Fault Models .....	4-3
4.3.1 <i>Linear HLFMs</i> .....	4-3
4.3.1.1 DC op amp model .....	4-3
4.3.1.2 DC/AC op amp model.....	4-4
4.3.1.3 DC and dc/ac Macromodel with Complex Input Impedance Function...4-8	
4.3.2 <i>Nonlinear HLFMs</i> .....	4-10
4.3.2.1 Model Architecture .....	4-10
4.3.2.2 Implementation in MAST .....	4-10
4.4 Conclusion .....	4-13
<b>Chapter 5: The Multiple Model Generation System (MMGS) for Automated Model Generation .....</b>	<b>5-1</b>
5.1 Introduction .....	5-1
5.2 Algorithm Evaluation based on an Mathematical Equation.....	5-4
5.3 Training Data for Estimation using PseudoRandom Binary Sequence Generator (PRBSG) .....	5-5
5.4 The Multiple Model Generation System (MMGS).....	5-6
5.4.1 <i>Manual Implementation</i> .....	5-6
5.4.2 <i>The MMGS</i> .....	5-13
5.4.2.1 The Automated Model Estimator (AME) .....	5-13
5.4.2.1.1 <i>The Pre-analysis</i> .....	5-14
5.4.2.1.2 <i>The Estimator</i> .....	5-15
5.4.2.1.3 <i>Post-analysis</i> .....	5-15
5.4.2.2 The Automated Model Predictor (AMP) .....	5-17
5.5 Key Factors to Improvement of Estimation Quality .....	5-17
5.5.1 <i>The Offset Parameter Related to Model Operating Points</i> .....	5-17
5.5.2 <i>Quality Improvement based on the Number of Samples</i> .....	5-19
5.6 Experimental Results .....	5-20
5.6.1 <i>Simulation for Nonlinearity</i> .....	5-20

5.6.2	<i>Validation Test for MMGS Generated Models via Time-domain (transient) Simulations</i> .....	5-21
5.7	Conclusion .....	5-22
<b>Chapter 6:</b>	<b>The Multiple Model Generation System (MMGS) for Multiple-Input Single-Output (MISO) Systems</b> .....	<b>6-1</b>
6.1	Introduction .....	6-1
6.2	The Algorithm on the MMGS for MISO Models .....	6-1
6.3	Experimental Results .....	6-3
6.3.1	<i>Analysis of MMGS</i> .....	6-3
6.3.2	<i>Simulation for Nonlinearity</i> .....	6-6
6.3.3	<i>Validation for Test for MMGS Generated Models via Time-domain (Transient) Simulations</i> .....	6-7
6.4	Conclusion .....	6-7
<b>Chapter 7:</b>	<b>High Level Modelling based on Models from the MMGS</b> .....	<b>7-1</b>
7.1	Introduction .....	7-1
7.2	Manual Conversion .....	7-1
7.2.1	<i>Structure of the Behavioural Model</i> .....	7-1
7.2.2	<i>Investigation to Bumpless Transfer using SMASH Simulator</i> .....	7-1
7.3	The Multiple Model Conversion System (MMCS) .....	7-9
7.4	Experimental Results .....	7-10
7.4.1	<i>The Inverting Amplifier</i> .....	7-10
7.4.2	<i>The Differential Amplifier</i> .....	7-12
7.5	Conclusion .....	7-14
<b>Chapter 8:</b>	<b>Multiple Model Generation System using Delta Operator</b> .....	<b>8-1</b>
8.1	Introduction .....	8-1
8.2	Overview of MMGSD .....	8-1
8.2.1	<i>The Deltarise Function</i> .....	8-4
8.2.2	<i>The Undeltarise Function</i> .....	8-5
8.2.3	<i>Two Functions Utility in MMGSD</i> .....	8-7
8.2.3.1	The AME.....	8-7
8.2.3.2	The AMP .....	8-9

8.3	Experimental Results .....	8-10
8.3.1	<i>A Single Model Detection</i> .....	8-10
8.3.2	<i>Comparison between MMGS and MMGSD</i> .....	8-14
8.3.3	<i>System Test Using a Lead-lag Circuit</i> .....	8-15
8.3.4	<i>Verification on the Multiple Model Generation Approach</i> .....	8-17
8.3.5	<i>Nonlinearity Modelling</i> .....	8-19
8.4	Conclusion .....	8-20
<b>Chapter 9:</b>	<b>High Level Fault Modelling and Simulation based on Models from the MMGSD .....</b>	<b>9-1</b>
9.1	Introduction .....	9-1
9.2	The Approach for Multiple Model Conversion System (MMCS).....	9-2
9.3	High Level Modelling and High Level Fault Modelling .....	9-7
9.4	Conclusion .....	9-19
<b>Chapter 10:</b>	<b>High Level Fault Simulation of a 3bit Flash Analogue to Digital Converter.....</b>	<b>10-1</b>
10.1	Introduction .....	10-1
10.2	Introduction to the 3bit Flash ADC.....	10-1
10.3	Multiple Model Generation by the Simulator and MMGSD .....	10-5
10.3.1	<i>Sample &amp; Hold and Decoder Model</i> .....	10-5
10.3.2	<i>Comparator Model</i> .....	10-5
10.4	High Level Fault Modelling of the 3bit Flash ADC .....	10-7
10.5	Conclusion .....	10-10
<b>Chapter 11: Conclusions and Future Work .....</b>	<b>11-1</b>	
11.1	Introduction .....	11-1
11.2	Automated Model Generation Approaches.....	11-1
11.3	High Level Fault Modelling.....	11-2
11.4	Future Work .....	11-2
<b>Appendix A: Characterises of the Two-stage Op Amp in HSPICE.....</b>	<b>A-1</b>	
A.1	Open-loop Mode with the Offset Compensation .....	A-2
A.2	Open-loop Gain Measurement .....	A-3

A.3	Input Offset Voltage.....	A-4
A.4	Common-mode gain.....	A-5
A.5	Common-mode Reject Ratio (CMRR) .....	A-6
A.6	Power Supply Reject Ratio (PSRR).....	A-7
A.7	Configuration of Unit Gain for Input and Output CMR .....	A-8
A.8	The Output Resistance .....	A-9
A.9	The Slew Rate and Settling Time.....	A-10
A.10	Comparison of the Simulation with Specification .....	A-12

**Appendix B: User Guide for MAST Language and Cosmos Simulator in Saber**

	.....	<b>B-1</b>
B.1	Introduction to Saber Simulator.....	B-1
B.2	Introduction to the MAST.....	B-2
B.2.1	<i>Construction of the MAST Language</i> .....	B-2
B.2.2	<i>The complete program in MAST language</i> .....	B-3
B.2.3	<i>Explanation</i> .....	B-4
B.2.3.1	Comment Section.....	B-4
B.2.3.2	Template Section.....	B-4
B.2.3.3	Declaration Section .....	B-5
B.2.3.3.1	<i>Header Declaration</i> .....	B-5
B.2.3.3.2	<i>Local declaration</i> .....	B-6
B.2.3.4	Values Section.....	B-6
B.2.3.5	The Equations Section.....	B-8
B.2.3.6	Netlist Section .....	B-9
B.3	Implementation in the Cosmos Simulator.....	B-10
B.3.1	<i>Simulation Run</i> .....	B-10
B.3.2	<i>Analysis</i> .....	B-10
B.3.2.1	The Transient Analysis .....	B-11
B.3.2.2	DC Analysis .....	B-12
B.3.2.3	AC analysis .....	B-13
B.4	Conclusion .....	B-14

**Appendix C: Behavioral Models Written in MAST .....**

C.1	MAST Code of Linear HLFMs .....	C-1
-----	---------------------------------	-----



C.1.1	MAST Code of opdc .....	C-1
C.1.2	MAST Code of opac .....	C-3
C.2	MAST Code of Nonlinear HLFMs .....	C-6
<b>Appendix D:</b>	<b>Analysis of Boyle's Output Stage in the Complex Frequency Domain .....</b>	<b>D-1</b>
D.1	Input Output Transfer Function .....	D-1
D.2	Output Impedance .....	D-3
<b>Appendix E:</b>	<b>Manual Implementation for the MMGS.....</b>	<b>E-1</b>
E.1	Process With the Offset Parameter .....	E-1
E.1.1	The Estimator .....	E-1
E.1.2	The Predictor .....	E-5
E.2	Process without the offset parameter .....	E-7
E.2.1	The Estimator .....	E-7
E.2.2	The Predictor .....	E-11
<b>Appendix F:</b>	<b>Quality Measurement based on Number of Samples .....</b>	<b>F-1</b>
<b>Appendix G:</b>	<b>Methodologies for Quality Improvement of the MMGS in MATLAB.....</b>	<b>G-1</b>
G.1	Suitable Values Check .....	G-1
G.2	Sample Detection .....	G-1
G.3	Observation of Covariance p.....	G-2
G.4	Stability Detector .....	G-3
G.5	The Saturation Detector .....	G-4
<b>Appendix H:</b>	<b>Codes for the MMGS and MMGSD.....</b>	<b>H-1</b>
H.1	The MMGS .....	H-1
H.1.1	The AME.....	H-1
H.1.2	The AMP.....	H-6
H.2	The MMGSD .....	H-9
H.2.1	The AME.....	H-9
H.2.2	The AMG.....	H-16

<b>Appendix I: Analytical Systems in MATLAB .....</b>	<b>I-1</b>
I.1 Analytical System in Signal Processing Toolbox .....	I-1
I.2 Analytical System in System Identification Toolbox .....	I-3
<b>Appendix J: The AME System Validation Using the HDL Simulator.....</b>	<b>J-1</b>
J.1 Introduction .....	J-1
J.2 Analogue Simulation.....	J-2
<i>J.2.1 VHDL-AMS Model Structure .....</i>	<i>J-2</i>
<i>J.2.2 Output Signal Using Data Loading and Writing Functions .....</i>	<i>J-2</i>
J.3 Test for the AME system .....	J-3
<b>Appendix K: Comparison of Various HDLs and Simulators .....</b>	<b>K-1</b>
K.1 Introduction .....	K-1
K.2 Brief Introduction to Different HDLs .....	K-2
K.3 Introduction to Different Simulators.....	K-2
<i>K.3.1 Introduction to SMASH.....</i>	<i>K-2</i>
<i>K.3.2 Introduction to SystemVision .....</i>	<i>K-3</i>
<i>K.3.3 Introduction to Cosmos in Saber.....</i>	<i>K-3</i>
K.4 Experimental Results .....	K-3
K.5 Conclusion .....	K-9
<b>Appendix L: The RML Estimation Algorithm Updates Equations for Both z and delta Transforms.....</b>	<b>L-1</b>
L.1 Estimation in z Transform .....	L-1
L.2 Estimation in Delta Transform .....	L-2
<i>L.2.1 Deltarise Function .....</i>	<i>L-3</i>
<i>L.2.2 Undeltarise Function .....</i>	<i>L-4</i>

## References

### List of Publications

# Table of Figures

Figure 2-1: Abstraction hierarchy [Pella97] .....	2-1
Figure 2-2: The proposed general structure of a high-level model.....	2-3
Figure 2-3: Linear time invariant block .....	2-5
Figure 2-4: Linear time varying block .....	2-6
Figure 2-5: The general process of the estimation .....	2-7
Figure 2-6: Overview of conversion routine within a design process [Grout05] .....	2-19
Figure 2-7: SAMSA general architecture and dependences with other MATLAB toolboxes [Zorzi02].....	2-21
Figure 3-1: Analogue fault category [Maly88] .....	3-2
Figure 3-2: Source (a)/ Drain (b) open fault models; Gate oxide short model (c).....	3-3
Figure 3-3: MOS transistor short fault models: DSS (a), GDS (b) and FSS (c).....	3-3
Figure 3-4: Specification for catastrophic faults.....	3-4
Figure 3-5: Graphical representation of the realistic defect based testability methodology for analogue circuit [Sachdev95] .....	3-5
Figure 3-6: Macromodel of inverting and non-inverting amplifiers [Zwo96].....	3-7
Figure 3-7: Structure of IFA [Ferguson88].....	3-10
Figure 3-8: Various Applications for IFA [Olbrich97].....	3-11
Figure 3-9: Analogue fault modelling from concept and schematic to layout. The arrows width represents the size of the fault lists [Sebeke95] .....	3-16
Figure 4-1: Schematic of the two-stage CMOS op amp .....	4-2
Figure 4-2: dc macromodel (see Appendix C: C.1.1).....	4-3
Figure 4-3: AC macromodel op amp (see Appendix C: C.1.2) .....	4-5
Figure 4-4: Additional pole/zero stages .....	4-8
Figure 4-5: Linear HLFM with arbitrary number of poles and zeros of the input impedance function opdc_zin and opac_zin (see Appendix C: C.1.1 and C.1.2).....	4-9
Figure 4-6: Nonlinear macromodels (see Appendix C: C.2 (for block 1&4)).....	4-10
Figure 4-7: Linear model parameterization.....	4-11
Figure 4-8: Algorithm of obtaining the output current function.....	4-11

Figure 4-9: Nonlinear two dimensional output current function $i = f(V_{out}, V_{in})$ .....	4-12
Figure 5-1: Schematics for the procedure of MMGS.....	5-2
Figure 5-2: The general process of the estimation .....	5-3
Figure 5-3: Schematic of the two-stage CMOS operational amplifier .....	5-4
Figure 5-4: The training data from PRBSG and output response using the open-loop amplifier .....	5-6
Figure 5-5: Input and output circuit transfer characteristic (one linear model) .....	5-7
Figure 5-6: Input and output circuit transfer characteristic ( $m1-m5$ and input threshold selected linear models).....	5-8
Figure 5-7: The input voltage and the output error voltage .....	5-9
Figure 5-8: Comparison of output signals based on one model during estimation process .....	5-10
Figure 5-9: The variation in <i>epsilon</i> vs input range based on one model .....	5-11
Figure 5-10: The variation in <i>epsilon</i> vs input range based on five models .....	5-12
Figure 5-11: Comparison of output signals based on five models.....	5-12
Figure 5-12: The flowchart for the AME.....	5-14
Figure 5-13: The algorithm for post-analysis .....	5-16
Figure 5-14: The predicted signal without the offset parameter .....	5-18
Figure 5-15: The predicted signal with the offset parameter .....	5-19
Figure 5-16: Signals from the predictor with 10,000 samples.....	5-19
Figure 5-17: The estimated signal from the AME system .....	5-20
Figure 5-18: The signal from the AMP system.....	5-21
Figure 5-19: Predicted square waveform based on models from the MMGS.....	5-22
Figure 6-1: Schematic of the two-stage CMOS operational amplifier .....	6-2
Figure 6-2: Two inputs and one output signals from HSPICE .....	6-3
Figure 6-3: The lead-lag circuit with two low-pass filters.....	6-4
Figure 6-4: The coefficients from the system identification toolbox.....	6-4
Figure 6-5: The model under discrete-time from MMGS.....	6-5
Figure 6-6: The predicted signal from the analytical system.....	6-5
Figure 6-7: The predicted signal from MMGS .....	6-6
Figure 6-8: The predicted signal with multiple models generated from MMGS.....	6-7

Figure 7-1: Structure of the behavioural op amp model .....	7-2
Figure 7-2: Model selection algorithm based on input range .....	7-4
Figure 7-3: The model selection process for the predictor .....	7-4
Figure 7-4: Data writing based on the sampling interval.....	7-6
Figure 7-5: Assigning data as stimuli.....	7-6
Figure 7-6: Data writing based on the sampling interval.....	7-7
Figure 7-7: Predicted signal from the AMP based on all model.....	7-8
Figure 7-8: High level modelling based on all model.....	7-9
Figure 7-9: Signals between the transistor level and the high level modelling .....	7-11
Figure 7-10: Simulation Speed Comparison between level 2 transistors and level 3 transistors .....	7-12
Figure 7-11: The differential amplifier .....	7-13
Figure 7-12: Signals between the transistor level and the high level modelling .....	7-13
Figure 8-1: The algorithm for the AME system .....	8-2
Figure 8-2: The square PRBS signal.....	8-11
Figure 8-3: The triangle PRBS signal .....	8-11
Figure 8-4: The predicted signal .....	8-12
Figure 8-5: The estimated signal.....	8-13
Figure 8-6: The predicted signal .....	8-13
Figure 8-7: The estimated signal.....	8-14
Figure 8-8: Coefficients under discrete-time from the AMP in the MMGS.....	8-15
Figure 8-9: A linear system with a high pass and low pass filter .....	8-16
Figure 8-10: The coefficients from the high-pass filter .....	8-16
Figure 8-11: The estimated signal.....	8-17
Figure 8-12: Threshold and samples for each model.....	8-20
Figure 8-13: The estimated signal with nonlinearity .....	8-20
Figure 9-1: The structure of the behavioural op amp model.....	9-2
Figure 9-2: The algorithm for the model selection .....	9-3
Figure 9-3: A flowchart for fault coverage measurement.....	9-4
Figure 9-4: The conditions for detecting the distance of two signals .....	9-5
Figure 9-5: The input signals with the saturation part .....	9-8
Figure 9-6: The biquadratic low-pass filter.....	9-9

Figure 9-7: The output signals from the low-pass filter.....	9-9
Figure 9-8: The output signals from the low-pass filter.....	9-10
Figure 9-9: The output signal from the transistor level fault-free simulation.....	9-11
Figure 9-10: HLFM for <i>M11_dss_1</i> .....	9-13
Figure 9-11: HLFM for <i>M10_gss_2</i> .....	9-13
Figure 9-12: Investigation which model is applied in relation to input and output .....	9-14
Figure 9-13: Threshold and samples for each model.....	9-15
Figure 9-14: New thresholds and samples for each model .....	9-15
Figure 9-15: HLFM for <i>M10_gss_2</i> based on new threshold set .....	9-15
Figure 9-16: ACM for TLFS, HLFM and Linear HLFM .....	9-16
Figure 9-17: Average speed measurement for TLFS, HLFM and Linear HLFM .....	9-17
Figure 9-18: Simulation Speed Comparison between level 2 transistors and level 3 transistors .....	9-18
Figure 10-1: Block diagram of the 3bit flash ADC .....	10-2
Figure 10-2: Schematic of the 3bit flash ADC .....	10-3
Figure 10-3: The CMOS comparator .....	10-4
Figure 10-4: Architecture of the comparator model .....	10-5
Figure 10-5: Threshold and samples for each model.....	10-6
Figure 10-6: The estimated signal.....	10-6
Figure 10-7: Nominal operation of the 3bit flash ADC .....	10-7
Figure 10-8: Failure of modelling in <i>M6_gss_1</i> .....	10-8
Figure 10-9: Average speed for each simulation .....	10-9
Figure A-1: Open-loop circuit with the offset compensation .....	A-2
Figure A-2: The signal for the offset voltage.....	A-3
Figure A-3: A method of measuring open-loop characteristics with dc bias stability..	A-3
Figure A-4: The open-loop gain measurement .....	A-4
Figure A-5: Configuration for measuring the input offset voltage .....	A-5
Figure A-6: a) Configuration for simulation the common-mode gain, b) Signal for common-mode gain.....	A-5
Figure A-7: Configuration for direct measurement of CMRR.....	A-6
Figure A-8: CMRR frequency response of magnitude and phase .....	A-7
Figure A-9: Configuration for direct measurement of PSRR .....	A-8

Figure A-10: Signals for $PSRR^+$ and $PSRR^-$ .....	A-8
Figure A-11: a) Unit-gain for input CMR and b) the signal .....	A-9
Figure A-12: Signal for the input CMR a) and the signal b) .....	A-10
Figure A-13: a) Measurement of the output resistance, b) output signal.....	A-10
Figure A-14: Measurement of the slew rate and settling time.....	A-11
Figure A-15: Signals from a) settling time and b) slew rate .....	A-11
Figure A-16: Improved slew rate by reducing the compensating capacitance .....	A-12
Figure B-1: The structure of the MAST language .....	B-2
Figure B-2: The program in MAST language.....	B-3
Figure B-3: The syntax for the <i>values</i> statement .....	B-7
Figure B-4: The resistor .....	B-7
Figure B-5: The syntax for if statement .....	B-8
Figure B-6: The syntax for <i>equations</i> section .....	B-8
Figure B-7: The syntax for the netlist .....	B-9
Figure B-8: The netlist for the operational amplifier.....	B-9
Figure B-9: Signals from the transient analyses .....	B-11
Figure B-10: Combination between input and output signals.....	B-12
Figure B-11: Results from the DC transfer analysis .....	B-13
Figure B-12: The RC filter for the ac analysis.....	B-13
Figure B-13: Signals from AC analysis .....	B-14
Figure D-1: Boyle's output stage .....	D-1
Figure F-1: Predicted signals based on different samples.....	F-2
Figure G-1: Number of samples in each model .....	G-2
Figure G-2: 1 <sup>st</sup> iteration for unstable model detection .....	G-4
Figure G-3: 2 <sup>nd</sup> iteration for unstable models replacement.....	G-4
Figure J-1: The triangle PRBS .....	J-2
Figure J-2: The output signal .....	J-3
Figure J-3: The estimated signal .....	J-3

Figure J-4: The predicted signal in the analogue system .....	J-4
Figure K-1: The linear op amp.....	K-3
Figure K-2: The op amp model written in MAST .....	K-4
Figure K-3: The model written in VHDL-AMS .....	K-5
Figure K-4: The top level in MAST.....	K-5
Figure K-5: VHDL-AMS top level in SMASH .....	K-7
Figure K-6: The output signal from Cosmos .....	K-7
Figure K-7: The signal in VHDL-AMS from SMASH .....	K-8
Figure K-8: The signal in VHDL-AMS from SystemVision.....	K-8
Figure K-9: The signal from SMASH with $r_i=15k\Omega$ .....	K-9



# *List of Terminologies*

AC:	Alternate Current
ACM:	Average Distance Confidence Measure
ADC:	Analogue to Digital Converter
AHDL:	Analogue Hardware Description Language
AMC:	Automated Model Conversion
AMG:	Automated Model Generator
AMS:	Analogue and Mixed Signal
ANN:	Artificial Neural Network
ARIMAX:	AutoRegressive Integrated Moving Average with eXogenous
ARX:	AutoRegressive with eXogenous variables
ASA:	Analogue Signature Analyzer
ASIC:	Application Specific Integrated Circuit
ATE:	Analogue Test Equipment
ATPG:	Automated Test Pattern Generation
AWE:	Asymptotic Waveform Evaluation
BDF:	Backward Differences Formula
BE:	Backward Euler
BIST:	Built-In Self-Test
BJT:	Bipolar Junction Transistor
CAD:	Computer Aided Design
CAFFEINE:	Canonical Functional Form Expression in Evolution
CCCS:	Current Controlled Current Source
CCVS:	Current Controlled Voltage Source
CG:	Conjugate Gradient
CMCOS:	Current-Mode Control and Observation Structure
CMOS:	Complementary Metal Oxide Semiconductor
CMR:	Common-mode Range
CMRR:	Common-mode Reject Ratio
CPU:	Central Processing Unit
CUT:	Circuit under Test

DAEs:	Differential-Algebraic Equations
DC:	Direct Current
DFT:	Design-for-Test
DPM:	Defects Per Million
DPTG:	Deterministic Pattern Testability Generation
DUT:	Design under Test
EDA:	Electronic Design Automation
e.g.:	for instance, for example
ELS:	Extended Least Squares
Eq.:	Equation
etc.:	et cetera
ff:	Forgetting Factor
FL:	Fuzzy Logic
FPC:	Fault Propagation Capabilities
FPGA:	Field Programmable Gate Arrays
GP:	Genetic Programming
HBIST:	Hybrid Built-In Self Test
HDL:	Hardware Description Language
HLFM:	High-Level Fault Model(ing)
HLFS:	High-Level Fault Simulation
HLM:	High-Level Model(ing)
HTSG:	Hybrid Test Stimulus Generator
IC:	Integrated Circuit
i.e.:	that is to say
LMM:	Levenberg-Marquardt Method
LMSE:	Least-Mean-Square Error
LSE:	Least Square Estimate
LSM:	Least Square Method
LTI:	Linear-time Invariant
LTV:	Linear-time Varying
MBFDI:	Model Based Fault Detection and Identification
MC:	Monte Carlo
MEMS:	Microelectromechanical System
MFD:	Multiple Fault Diagnosis

MISO:	Multi-input Single-output
MMCS:	Multiple Model Conversion System
MMCS D:	Multiple Model Conversion System using Delta operator
MMGS:	Multiple Model Generation System
MMGS D:	Multiple Model Generation System using Delta operator
MNA:	Modified Nodal Analysis
MOR:	Model Order Reduction
MOSFET:	Complementary Metal Oxide Semiconductor Field Effect Transistor
NaN:	Not A Number
NARX:	Nonlinear AutoRegressive with eXogenous input
NF:	Neural-Fuzzy
NN:	Neural Network
NORM:	Nonlinear model Order Reduction Method
ODE:	Ordinary Differential Equation
Op amp:	Operational Amplifier
OTA:	Operational Trans-conductor Amplifier
PCB:	Printed Circuit Board
PDE:	Partial Differential Equation
PLD:	Programmable Logic Device
PLL:	Phase Locked Loop
PR:	Positive-Real
PRBSG:	PseudoRandom Binary Sequence Generator
PRESS:	Predicted Residual Error Sum of Squares
PRIMA:	Passive Reduced-Order Interconnect Macromodeling Algorithm
PR-TBR:	Positive-Real TRB
PRW:	PseudoRandom Walk
PSRR:	Power Supply Rejection Ratio
PVL:	Padé-via-Lanczos
PWL:	Piecewise Linear
PWP:	Piecewise Polynomial
RBF:	Radial Basis Functions
RML:	Recursive Maximum Likelihood
RPEM:	Recursive Prediction Error Method
RPLR:	Recursive Pseudo-Linear Regression

RTL:	Register Transfer Level
SAG:	Simplification after Generation
SAT:	Simulation after Test
SBT:	Simulation before Test
SDARX:	Situation-dependent ARX
SPD:	Symmetric Positive Definite
SPICE:	Simulation Program with Integrated Circuit Emphasis
SR:	Slew Rate
SRAM:	Static Random Access Memory
TBR:	Truncated Balanced Realization
tlu:	Table Look-up
TPWL:	Trajectory PWL
TRAP:	Trapezoidal
VCCS:	Voltage Controlled Current Source
VCVS:	Voltage Controlled Voltage Source
VHDL:	VHSIC Hardware Description Language
VHSIC:	Very High Speed Integrated Circuits
VLSI:	Very Large Scale Integration

# ***Chapter 1: Introduction***

It is well known that most electronics manufacturing processes are not perfect; most products may contain defects. An electronic product may fail either due to manufacturing defects or out of specification performance. The latter can be caused by inadequate design. Manufacturing defects include excess material shorting interconnection wiring on printed circuit boards (PCBs) or integrated circuits (ICs), open circuits due to breaks in wires and use of out-of-specification components. In industry, many types of defects can happen with varying probability, which can be reduced by improving the quality of the manufacturing process. Moreover, if the design is not sufficiently tolerant of in-specification component or process variations a circuit may be faulty even though it is manufactured correctly. With fault-free components, fault tolerance should be improved if a chip is designed far beyond the specification, but this design may be more expensive, so no profits are made. Other factors including simulation time, test application time and commercial costs also need to be taken into account. Therefore, efficient test is required so that manufacturers will not lose profits.

There are two approaches to test: structural test and functional test. The former, also known as defect oriented test (DOT), is used to detect manufacturing defects directly rather than the functional error produced. The latter ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions, i.e., test the specifications of the device under test (DUT). Functional test has always been the dominant approach for analogue circuits, and it is often the only possibility without the availability of tools to support structural test such as analogue fault simulation and associated fault models. Therefore, the DUT is sometimes overtested. Moreover, complete functional test is economically impractical for production stage test of most digital integrated circuits (ICs), for example, a 32 bit adder may require about 6,000 years to test all operations at 100MHz. Therefore, structural test is normally adopted for digital circuits. This is feasible because digital fault simulation using stuck-at faults and gate-level models is straightforward and effective. Fault simulation and modelling can also be used in structural test, so that DOT may be more efficient and effective.

In recent years, functional complexity, speed and performance of analogue and mixed signals ICs have increased, which have resulted in even higher test costs because the tester needs to be more accurate and faster than the circuit under test (CUT), otherwise it can not test all the specifications. Buying and running high performance analogue test equipment (ATE) is very expensive. Thus structural test is required for analogue circuits due to its potential efficiency and low cost [Olbrich96] [Fang01] [Aktouf05]. Furthermore, structural test may provide additional information on quality and reliability which is not available from functional test [Rich92] [Healy05].

Analogue test is very time consuming and expensive, its costs dominate approximately 90% of the whole testing cost in modern analogue and mixed mode ICs [Abra95]. Analogue test suffers from a lack of automated test pattern generation (ATPG) algorithms, long testing time and unknown test quality [Bartsch99] [Healy05]. These can be resolved by establishing design for test (DFT) and built-in self-test (BIST) techniques [Ohletz91] [Russell93] [Renovell96] [Healy05] to improve testability. Moreover, the test pattern generation can be simplified, because the test vectors are generated internally, and it allows field testing to be performed for many years after manufacture [Zwo00]. For digital circuits DFT and BIST are well established and easily applied to almost all circuits. Unfortunately, in analogue domain, these techniques can only be employed for certain classes of circuits [Spinks98].

It is desirable to use defect oriented test (DOT) strategies at the layout level in order to simplify test of analogue or mixed mode circuits [Bratt95] [Kalpana04]. Application of inductive fault analysis (IFA) techniques for DOT development has received a lot of attention. IFA was a subject of research beginning in the middle 1980s, including several significant projects at Carnegie Mellon University [Ferguson88]. It is a systematic and automatic method for determining what faults are most likely to take place in a large circuit from details of the manufacture process such as circuit's technology, realistic defect occurrences, fabrication defect statistics, and physical layout. IFA can distribute defects over the physical layout of the circuit, and simulate to determine what faults may result. However, a description of the manufacturing defect statistics is required, which provides a list of possible electrical faults when mapped onto the layout. At circuit level the fault list includes short, opens, breaks in lines and

parameter variations in both active and passive components such as resistors, capacitors and transconductors. Therefore, DOT can be enhanced with IFA [Harvey95].

During the last few years, high level fault modelling (HLFM) and high level fault simulation (HLFS) techniques have been proposed for modern complex analogue and mixed mode system design due to its high speed [Wilson02] [Kilic04] [Joannon08]. Fault simulation is an essential element in the development of structural test programs for digital, analogue and mixed mode ICs, and can be carried out at transistor level and high level. The aim of simulation is to define an efficient structural test program and to simulate the behaviour of a circuit in the presence of a fault specification. Simulation and modelling are dynamically related, especially when high level simulation is run, hence a modelling technique is required.

High level models comprise both faulty and fault-free models. High level fault-free modelling may simply indicate behaviour of a fault-free circuit, but normally it is not able to cope with faulty conditions with strong nonlinearity. The only way to solve this is to replace the fault-free model with a faulty one. Furthermore, in fault-free simulation, the difference between transistor level and high level may not be obvious, but this may be shown under fault simulation. HLFM techniques have shown the potential ability to deal with at least some degree of nonlinearity in large systems.

Unlike for linear systems, no technique currently guarantees for completely general nonlinear systems, even in principle, to produce a macromodel that conforms to any reasonable fidelity metric. The difficulty stems from the fact that nonlinear systems can be widely varied, with extremely complex dynamical behaviour possible, which is very far from being exhaustively investigated or understood. Generally in view of the diversity and complexity of nonlinear systems, it is difficult to conceive of a single overarching theory or method that can be employed for effective modelling of an arbitrary nonlinear block.

Models can be obtained either manually or automatically. Automated model generation (AMG) methodologies are becoming an increasingly important component of methodologies for effective system verification. Similar to manual creation, AMG can generate lower order macromodels via an automated computational procedure by

receiving the information from transistor level models [Roychowdhury03] [Roychowdhury04].

There are several broad methodologies for AMG, a fundamental decision is the model structure, which in general terms divides into linear time-invariant (LTI), linear time-variant (LTV), nonlinear time-invariant and nonlinear time-variant types. An estimation algorithm is then required in order to obtain parameters for these models. These algorithms may use lookup tables [Yang04], radial basis functions (RBF) [Mutnury03], artificial neural networks (ANN) [Davallo91] [Zhang00] and its derivations such as fuzzy logic (FL) [Kaehler] and neural-fuzzy network (NF) [Uppal05], and regression [Middleton90] [Ljung99]. Model generators can also be categorized into the black, grey or white box approaches, depending on the level of existing knowledge of the system's structure and parameters.

These models are in the form of mathematical equations that reproduce the input-output relationships of the original circuit, and can be easily converted into any format convenient for use with system-level simulation tools, e.g., VHDL-AMS [Ashenden03], MAST [Saber04], and even as SPICE subcircuits. piecewise polynomial (PWP) is further used to capture different loading effects, simultaneous switching noise (SSN), crosstalk noise and so on [Dong04] [Dong05], faster modelling speed is achieved, but multiple training data is required to cover various operating regions.

However, AMG may produce high order models of excessive complexity (e.g., [Huang03] [Tan03] [Wei05]), in which case model order reduction (MOR) techniques are required [Gielen05]. A survey paper [Roychowdhury04] discusses MOR techniques with respect to various model types, and in a variety of contexts: LTI MOR [Pillige90], LTV MOR [Phillips98] [Roychowdhury99] and weakly nonlinear methods including polynomial-based [Li03] [Li05], trajectory piecewise linear (TPWL) [Rewiński01], and piecewise polynomial (PWP) [Dong03].

Unfortunately, there are not any papers describing the use of AMG approaches for HLFM at a system level. For straightforward system simulation relatively simple models may be adequate, but they can prove inadequate during HLFM and HLFS. The accuracy and speedup of existing models may be doubted when fault simulation is



implemented because faulty behaviour may force (non-faulty) subsystems into highly nonlinear regions of operation, which may not be covered by their models. Multiple training data is required to cover the potentially wide range of operating conditions.

Although a fault model can behave accurately in a circuit, it may fail in a large system due to fault propagation [Zwo97] [Bartsch99]. It is a major problem in the industry because if a fault distributes along a fault propagation path in a system, several process can be affected. It is therefore important to understand the mechanism of propagation that identify the order of occurrence of events and specify the paths of fault propagation in causal qualitative models [Batra04]. For HLFM techniques in a system, it is crucial to know whether or not the high level fault-free operational amplifier (op amp) model is able to correctly model propagation of the faulty behaviour, and how fault propagation can be predicted so that the suitable model is chosen for the whole system. Therefore, not only parameters of the model but also the system need to be varied [Bartsch99].

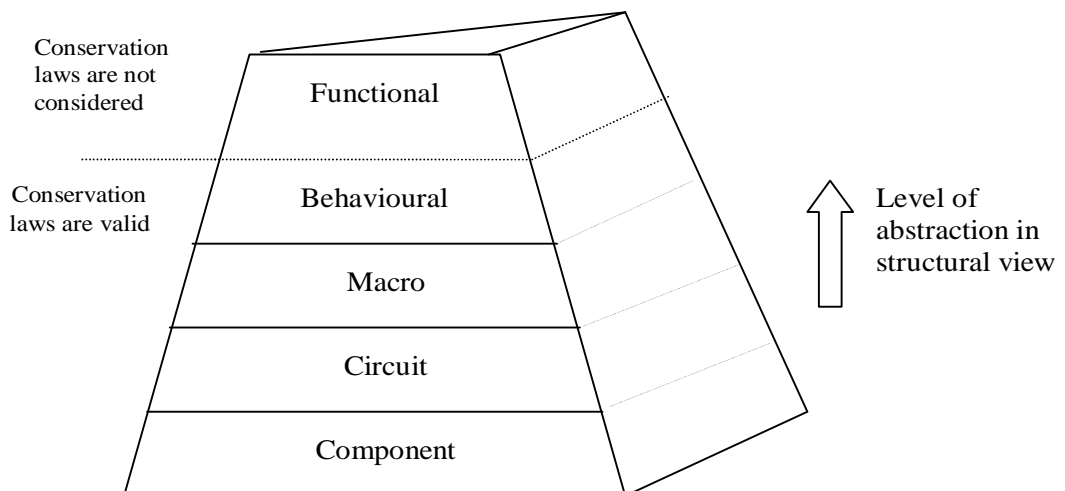
The general aim of this study is to prove that models generated using automated model generation can be employed to implement high level fault modelling (HLFM). This thesis begins with an introduction to model generation and model order reduction approaches in chapter 2. In chapter 3 fault modelling techniques and other important approaches for fault detection are introduced. High level fault modelling and simulation using MAST is reviewed in chapter 4. The algorithm termed multiple model generation system (MMGS) for the automated model generation (AMG) is developed for single-input and single-output (SISO) systems in chapter 5 followed by the introduction of MMGS for multiple-input single-output (MISO) systems in chapter 6. Chapter 7 introduces high level modelling using the models generated from last two chapters. The multiple model generation system using the delta operator (MMGSD) is discussed in chapter 8 followed by the high level fault modelling in chapter 9. In chapter 10 HLFM of a 3bit flash ADC is investigated. In chapter 11 conclusion and future discussion are supplied.

# ***Chapter 2: Literature Review of Approaches for High Level Modelling and Automated Model Generation***

The aim of the chapter is to review some model generation approaches for high level fault-free models. These approaches can be realised either manually or automatically. The general structure of system modelling is reviewed in section 2.1. In section 2.2 various modelling techniques are introduced. Automated model generation approaches for high level fault-free models are discussed in section 2.3. Section 2.4 reviews various approaches to generate a model in different computing languages.

## ***2.1 Review of Possible Models of a System***

Figure 2-1 shows the possible abstraction hierarchy of an electronic system [Pella97]. This is also reckoned by [Ashenden03] [Joannon08]. Models, which can refer to both “circuits” and “devices” [Getreu93], are produced by combining lower-level building blocks to create higher-level building blocks. Lower-level blocks can be either elemental models or previously created hierarchical models, elemental intrinsic building blocks are at the bottom of all hierarchical models [Fang01] [Joannon06].



**Figure 2-1:** Abstraction hierarchy [Pella97]

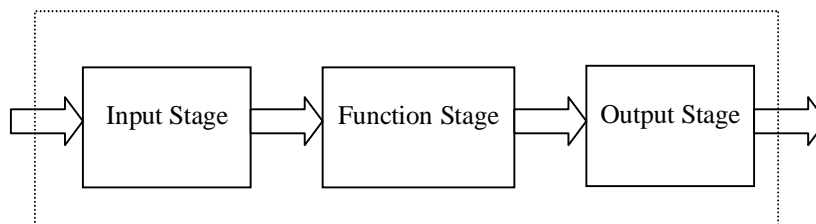
On moving from bottom to top the structure, the circuit model becomes simpler and the simulation speed can be increased significantly. At the component level transistors and passive devices are available as basic elements; the circuit level consists of operational amplifiers (op amps), comparators and so on; at the macro level the functionality of a basic element is replaced by a more abstract representation including controlled voltage and current sources. Models at the behavioural level comprise mathematical relationships between input and output signals. The connection pins of these models carry physical signals, which are subjected to conservation laws. These are Kirchhoff's current and voltage law in electrical systems; at the functional level, complex elements such as data acquisition or even modem blocks are available, but the conservation laws are not available at connection pins, that is, signals are not physical any more [Rosen98] [Nikitin07]. Each level in the hierarchy may be represented by a model, so the whole system can be simulated by high level modelling (HLM) to deal with nonlinear behaviour. Moreover, structure of the system is less complex which may result in reduction of simulation time [Kalpana04] [Joannon08].

Another design flow methodology is from top to bottom methodology seen in Figure 2-1, which is the opposite way to bottom to top methodology. One of the advantages of this methodology is that behavioural models have previously been built, so each structural block description can be validated in the overall system using other block behavioural models [Joannon06].

Moreover, a method between top to bottom and bottom to top methodologies is the "Meet in the middle" methodology [Joannon06]. It takes advantages of each method. The first step is coming from top to bottom design method: system architecture is described thanks to functional block models. Once again, system specifications are budgeted into numerous block specifications. In this methodology, a trade-off between system engineers and component designers allow them to determine realistic specifications for each block. Then these blocks are separately designed and assembled together. Finally, the system is validated as in a bottom to top flow.

## 2.2 Review of Various Modelling Techniques

The aim of the section is mainly to introduce three types of models because they will be used in the thesis: macromodels, behavioural models and transfer function models. Macromodels and behavioural model are commonly used in these days. These models normally include three stages as shown in Figure 2-2.



**Figure 2-2:** The proposed general structure of a high-level model

The input stage acts as an interface between electrical signals outside the block and the internal representation of the function by mathematical equations in the function stage. The input stage models properties such as input resistance, input capacitance, input bias current, input offset voltage, the bounds of input current and the limits of the input voltage. The function stage is the central part of the model, which makes use of mathematical functions to represent any kind of analogue circuit such as op amps, ADCs and DACs. The output stage maps the results provided by the function stage into the electrical environment of the output block. Like the input stage, it can be implemented with components such as transistors, resistors and mathematical functions [Fang01] [Nikitin07]. There may be some cases, in which more stages are required to model extra information. [Bartsch99] developed a behavioural model that consists of two middle stages in order to include extra poles and zeros. In following subsections the strengths and weakness of these modelling techniques are addressed.

### 2.2.1 Macromodelling (structural approach)

In macromodelling two techniques can be utilised: the simplification and the built-up technique [Joannon06]. The former replaces parts of an electronic circuit with ideal components, while the built-up technique completely rebuilds a part of the circuit with ideal elements to meet certain external circuit specification. The well-known Boyle op amp macromodel [Boyle74] is an excellent example of macromodelling. The model

employs ideal components such as ideal current sources and ideal transistors based on SPICE primitives [Bartsch99]. Macromodels themselves can become building blocks for other models. However, there is a significant restriction: the modeller must use the pre-defined element, which can limit the ability to easily incorporate required effects [Getreu93].

### **2.2.2 Behavioural modelling**

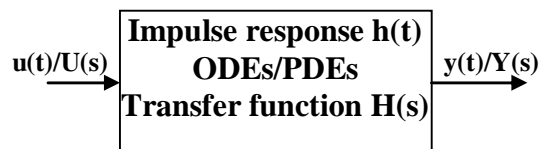
Behavioural modelling uses mathematical equations and control statements such as *If...then...else*. It is well-suited to provide models that match requirements of designers at all stages of the design process. This type of model can describe the behaviour of a block directly without consideration of how the block is built. A simple example is the 2-pole transfer function:  $k/(s+1)(s+2)$ . Behavioural modelling through the use of a hardware description language (HDL) is often preferred with respect to macromodelling because of its high speed and simplicity [Joannon06] [Nikitin07]. Moreover, its mathematical equations can be easily converted into the format of HDLs such as VHDL-AMS and its high simulation speed [Getreu93] [Dong03]. Like macromodelling there are also two ways to realize behavioural models: analytical and statistical [Getreu93]. The analytical description of a sub-circuit is usually created by a designer. However, it may still be an expensive approach and behavioural modelling still needs verification even though an analytical approach is adopted. A statistical behavioural model can be built in the same way as a statistical macromodel [Yang98]. Unlike macromodelling, behavioural modelling does not need look-up techniques. Once the structure of a macromodel is known, a behavioural model can always be developed, however, not always vice versa.

### **2.2.3 Transfer function modelling**

Transfer function models do not have the full capabilities of equation based descriptions. They make use of the flexibility of the SPICE dependent sources (VCCS, CCCS, VCVS and CCVS), where the output source has a transfer function or nonlinear dependency on the input [Bartsch99]. The implementation of these sources can be “hidden” by being coded directly into the simulator, but they can be recognised by the restrictions of using a fixed number of pins and being unidirectional [Getreu93].

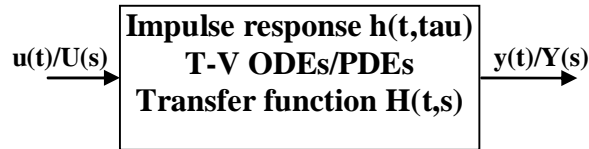
## 2.3 Review of Automated Model Generation (AMG) Approaches

In this subsection work on automated model generation (AMG) methodologies is reviewed. The model generated can be structured as either linear-time invariant (LTI), linear-time varying (LTV), nonlinear-time invariant or nonlinear-time varying. LTI no doubt form the most important class of dynamical systems. The basic structure of a LTI block for mixed mode circuits is illustrated in Figure 2-3, where  $u(t)$  and  $y(t)$  represent inputs, and output to the system in the time domain, respectively.  $U(s)$  and  $Y(s)$  are forms in the Laplace domain. The definitive property of any LTI system is that the input and output are related by convolution with an impulse response  $h(t)$  in the time-domain, i.e.,  $y(t) = x(t) \cdot h(t)$ , their transforms are related to multiplication with a system transfer function  $H(s)$ , i.e.,  $Y(s) = X(s) \cdot H(s)$ . Their relationship can be expressed by partial differential equations (PDEs) or ordinary differential equations (ODEs). Such differential equations can be easily implemented using analogue hardware description language (AHDL) descriptions. A typical model structure for LTI is AutoRegressive with eXogenous (ARX) that is able to describe any single-input single-output (SISO) linear discrete-time dynamic system [Ljung99].



**Figure 2-3:** Linear time invariant block

LTV are used in practice because most real-world systems are time-varying as a result of system parameters changing as function of time. They also permit linearization of nonlinear systems in the vicinity of a set of operating points of a trajectory. Similar to LTI systems, LTV can also be completely characterized by impulse responses or transfer functions. The main difference between them is that time-shift in the input of LTV does not necessarily result in the same time-shift of the output. A basic structure of LTV is depicted in Figure 2-4, where  $u(t)$  and  $y(t)$  represent inputs, and output to the system in the time domain, respectively.  $U(s)$  and  $Y(s)$  are forms in the Laplace domain.



**Figure 2-4:** Linear time varying block

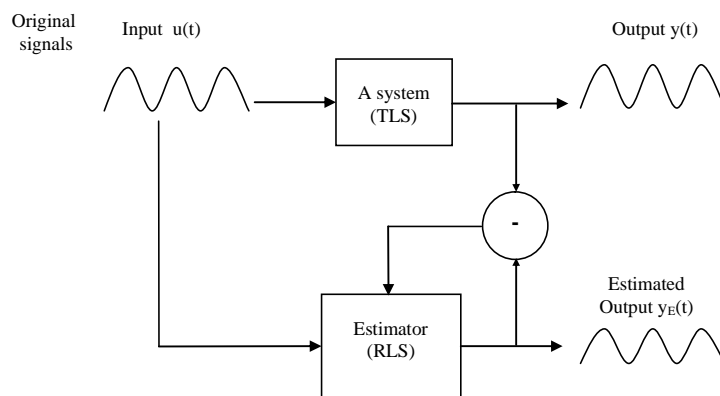
LTV are better able to handle time variation in state-space forms [Ljung99]. Furthermore, nonlinear models such as the Wiener and Hammerstein model, and Situation-Dependent AutoRegressive with eXogenous (SDARX) give much richer possibilities to describe systems.

These models can be generated by using estimation algorithms, which comprise lookup tables [Yang04], radial basis functions (RBF) [Mutnury03], artificial neural networks (ANN) [Davallo91] [Zhang00] and its derivations such as fuzzy logic (FL) [Kaehler] and neural-fuzzy network (NF) [Uppal05], and regression [Simeu05]. Model generators can also be categorized into the black, grey or white box approaches, depending on the level of existing knowledge of the system's structure and parameters. [Dong05] indicates that white-box methods can produce more accurate macromodels than black-box methods. However, this work was only applied to a limit number of digital circuits.

Regression is an approach that is of interest in this thesis. It is a form of statistical modelling that attempts to evaluate the relationship between one variable (termed the dependent variable) and one or more other variables (termed the independent variables) [Regression]. It can be divided into linear regression and nonlinear regression [Ljung99] for generating linear or nonlinear models. [McConaghy05] [McConaghy05a] use the regression approach [Hong03], via the predicted residual error sums of squares (PRESS) statistic [Breiman96], to test predictive robustness of linear models that are generated by an automatic symbolic model generator named CAFFEINE (Canonical Functional Form Expression in Evolution). CAFFEINE takes SPICE simulation data as inputs to generate open-loop symbolic models by using genetic programming (GP) via a grammar that is specially designed to constrain the search to a canonical functional form without cutting out good solutions. Results show that these models are interpretable, and handle nonlinearity with better prediction quality than posynomials (coefficients of a polynomial need not be positive, and, on the other hand, the exponents of a posynomial

can be real numbers, while for polynomials they must be non-negative integers). Unfortunately, McConaghy et al did not address whether the generated model can be fitted into a large system and model nonlinearity well. Additionally, speed of model generation was not mentioned.

Linear models can be obtained using recursive least square (RLS) estimation. It is a mathematical procedure for finding the best-fitting curve to a given set of points by minimizing the sum of the squares of the offsets of the points from the curve [Ljung99]. Its general process is shown in Figure 2-5, where  $u(t)$  is the input stimulus, which is used to connect both a system and the estimator;  $y(t)$  is the output response from a system using the transistor level simulation (TLS);  $y_E(t)$  is the output response using an estimation approach such as the RLS.



**Figure 2-5:** The general process of the estimation

Both the system and estimator use the input stimulus to produce individual output response, both response are then compared, if the difference is significant, the parameters of the model will be changed in order to achieve smaller difference.

This thesis will make use of algorithms to derive models based on [Wilkinson91] [Middleton90]. The following outlines the algorithms used. [Wilkinson91] employ RLS estimation combined with the delta operator [Middleton90] to obtain the transfer function of a real time controller for a servo motor system instead of using discrete-time transfer function because that model coefficients in discrete-time models strongly depend on the sampling rate, which result in aliasing and slow simulation time. By using the delta operator the coefficients produced relate to physical quantities, as in the



continuous-time domain model, but are less susceptible to the choice of sampling interval [Wilkinson91]. Initially a discrete-time system is given in Eq. 2-1:

$$y(t) = -a_1 y(t-1) - a_2 y(t-2) - \dots - a_{na} y(t-na) + b_1 u(t-1) + b_2 u(t-2) + \dots + b_{nb} u(t-nb) \quad \text{Eq. 2-1}$$

This equation is then written in a linear regression form, as shown in Eq. 2-2:

$$y(t) = \varphi^T(t)\theta \quad \text{Eq. 2-2}$$

where  $\theta$  is the parameter vector shown in Eq. 2-3,  $\varphi(t)$  is the regression vector displayed Eq. 2-4.

$$\theta = [a_1 \ a_2 \ \dots \ a_{na} \ b_1 \ b_2 \ \dots \ b_{nb}]^T \quad \text{Eq. 2-3}$$

$$\varphi^T(t) = [-y(t-1) \ \dots \ -y(t-na) \ \ u(t-1) \ \dots \ u(t-nb)] \quad \text{Eq. 2-4}$$

The least square estimate (LSE) of the parameter vector can be found from measurements of  $u(t)$  and  $y(t)$  using Eq. 2-5 [Ljung99]:

$$\theta(t) = \left[ \frac{1}{N} \sum_{t=1}^N \varphi(t)\varphi^T(t) \right]^{-1} \left[ \frac{1}{N} \sum_{t=1}^N \varphi(t)y(t) \right] \quad \text{Eq. 2-5}$$

Its recursive form is expressed in Eq. 2-6, where  $\varepsilon(t)$  is the prediction error,  $\lambda(t)$  represents forgetting factor (ff),  $P(t)$  indicates covariance matrix,  $L(t)$  is the gain vector.

$$\begin{aligned} \theta(t) &= \theta(t-1) + L(t)\varepsilon(t) \\ \varepsilon(t) &= y(t) - \varphi^T(t)\theta(t-1) \\ L(t) &= \frac{P(t-1)\varphi(t)}{\lambda(t) + \varphi^T(t)P(t-1)\varphi(t)} \\ P(t) &= \frac{1}{\lambda(t)} \left[ P(t-1) - \frac{P(t-1)\varphi(t)\varphi^T(t)P(t-1)}{\lambda(t) + \varphi^T(t)P(t-1)\varphi(t)} \right] \end{aligned} \quad \text{Eq. 2-6}$$

The linear regression is then restructured using the delta operator as shown Eq. 2-7 [Middleton90], where  $\delta$  represents delta,  $q$  is the forward shift operator and  $T_s$  is the sampling interval. The relationship between  $\delta$  and  $q$  is a simple linear function, so  $\delta$  can offer the same flexibility in the modelling of discrete-time systems as  $q$  does.

$$\delta = \frac{q-1}{T_s} \quad \text{Eq. 2-7}$$

This operator behaves as a form of the forward-difference formula, as shown in Eq. 2-8 [Burden85]. This is used extensively in numerical analysis for computing the derivative of a function at a point.

$$f'(x) = \frac{f(x+h) - f(x)}{h} \quad \text{Eq. 2-8}$$

The delta operator makes use of the discrete incremental difference (or delta) operator that whilst operating on discrete data samples, is similar to those of the continuous-time Laplace operator. A better correspondence can be obtained between continuous and discrete time if the shift operator is replaced by a difference operator that is more like a derivative [Middleton90].

A similar procedure is used to achieve regression based on the delta operator. This starts by considering a continuous time transfer function shown in Eq. 2-9.

$$G(s) = \frac{b_0 s^n + b_1 s^{n-1} + \dots b_n s^0}{s^m + a_1 s^{m-1} + \dots a_m s^0} \quad \text{Eq. 2-9}$$

When  $T_s$  is sufficiently short, the continuous time transfer function  $G(s)$  is equal to the delta transfer function  $G(\delta)$  [Middleton90] displayed in Eq. 2-10.

$$G(\delta) = \frac{y(t)}{u(t)} = \frac{b_0 \delta^n + b_1 \delta^{n-1} + \dots b_n \delta^0}{\delta^m + a_1 \delta^{m-1} + \dots a_m \delta^0} \quad \text{Eq. 2-10}$$

After arranging this equation, Eq. 2-11 is obtained:

$$y(t)\delta^m = -(a_1\delta^{m-1} + \dots + a_m)y(t) + (b_0\delta^n + \dots + b_n)u(t) \quad \text{Eq. 2-11}$$

This can be written as Eq. 2-12 [Middleton90], which is similar to Eq. 2-2:

$$\delta^m y(t) = \varphi^T(t)\theta \quad \text{Eq. 2-12}$$

where

$$\theta = [a_1 \ a_2 \ \dots \ a_m \ b_0 \ b_1 \ \dots \ b_n]^T \quad \text{Eq. 2-13}$$

$$\varphi^T(t) = [-\delta^{m-1}y(t) \ \dots \ -\delta^0 y(t) \ \delta^n u(t) \ \dots \ \delta^0 u(t)] \quad \text{Eq. 2-14}$$

Using a similar approach to least square estimate (LSE) in the discrete-time transform, the parameter vector is obtained using the delta operator in Eq. 2-15:

$$\theta(t) = \left[ \frac{1}{N} \sum_{t=1}^N \varphi(t)\varphi^T(t) \right]^{-1} \left[ \frac{1}{N} \sum_{t=1}^N \varphi(t)\delta^m y(t) \right] \quad \text{Eq. 2-15}$$

RLS is also obtained in Eq. 2-16:

$$\begin{aligned} \theta(t) &= \theta(t-1) + L(t)\varepsilon(t) \\ \varepsilon(t) &= \delta^m y(t) - \varphi^T(t)\theta(t-1) \\ L(t) &= \frac{P(t-1)\varphi(t)}{\lambda(t) + \varphi^T(t)P(t-1)\varphi(t)} \\ P(t) &= \frac{1}{\lambda(t)} \left[ P(t-1) - \frac{P(t-1)\varphi(t)\varphi^T(t)P(t-1)}{\lambda(t) + \varphi^T(t)P(t-1)\varphi(t)} \right] \end{aligned} \quad \text{Eq. 2-16}$$

However, the approach in [Wilkinson91] is only available to any single-input single-output (SISO) systems.

Unfortunately, AMG may produce high order models of excessive complexity for both continuous-time and discrete-time systems, so model order reduction (MOR) techniques are required. The purpose of MOR is to use the properties of dynamical systems in order to find approaches for reducing their complexity, while preserving (to the

maximum possible extent) their input-output behaviour. It comprises a branch of systems and control theory [Roychowdhury04]. Combining MOR with the model structures produces new model structures dubbed LTI MOR [Pillige90], LTV MOR [Phillips98] [Roychowdhury99] and weakly nonlinear methods including polynomial-based [Li03] [Li05], trajectory piecewise linear (TPWL) [Rewiński01], and piecewise polynomial (PWP) [Dong03].

Mathematically, a LTI model with a MOR method is expressed as a set of differential equations. In Eq. 2-17  $u(t)$  represents the input waveforms to the block and  $y(t)$  are the outputs. The number of inputs and outputs is relatively small compared to the size of  $x(t)$ , which is the state of the internal variables of the block.  $A$ ,  $B$ ,  $C$ ,  $D$  and  $E$  are constant matrices,  $E \& A \in R^{n \times n}$ ,  $B \in R^{n \times p}$ ,  $C \in R^{p \times n}$ ,  $u(t) \in R^p$ .

$$\begin{aligned} E\dot{x} &= Ax(t) + Bu(t) \\ y(t) &= C^T x(t) + Du(t) \end{aligned} \quad \text{Eq. 2-17}$$

MOR methods for LTI systems fall into two major groups: Projection-based methods and Non-projection based methods. The former consists of such methods as Krylov-subspace (moment matching methods), Balanced-truncation method, proper orthogonal decomposition (POD) methods etc. Krylov-subspace based techniques such as Padé-via-Lanczos (PVL) techniques [Feldmann95], Krylov-subspace projection methods were an important milestone in LTI MOR macromodelling [Grimme97]. Non-projection based methods comprise methods such as Hankel optimal model reduction, singular perturbation method, various optimization-based methods etc. Via Krylov-subspace operation, reduced models are obtained in Eq. 2-18, where  $\tilde{E}, \tilde{A}, \tilde{B}, \tilde{C}$  are reduced order matrices,  $\tilde{E} \& \tilde{A} \in R^{q \times p}$ ,  $\tilde{B} \in R^{q \times p}$ ,  $\tilde{C} \in R^{p \times q}$ ,  $W$  and  $V$  are matrices for spanning the matrices.

$$\tilde{E} = W^T E V, \tilde{A} = W^T A V, \tilde{B} = W^T B, \tilde{C} = C V \quad \text{Eq. 2-18}$$

However, the reduced models using Krylov methods retained the possibility of violating passivity, or even being unstable [Roychowdhury03]. In this thesis we model operational amplifiers (op amps) instead of passive systems. A passive system is

defined as one that can not generate energy under any circumstances. A system is stable for any bounded inputs, its response remains bounded. In a LTI model passivity guarantees stability if its response remains bounded for any bounded input. Passivity is a natural characteristic of many LTI networks, especially interconnect networks. It is essential that reduced models of these networks also be passive, since converse implies that under some situation of connectivity, the reduced system will become unstable and diverge unboundedly from the response of the original system [Roychowdhury04]. An algorithm termed PRIMA (Passive Reduced-Order Interconnect Macromodeling Algorithm) [Odabasioglu97] has been developed to preserve this possibility. It generates provably passive reduced-order N-port models for RLC interconnect circuits. The modified nodal analysis (MNA) equation is formed using these ports along with sources in time domain as seen in Eq. 2-19:

$$\begin{aligned} C\dot{x}_n &= -Gx_n + Bu_N \\ i_N &= L^T x_n \end{aligned} \tag{Eq. 2-19}$$

where the vectors  $i_N$  and  $u_N$  indicate the port currents and voltages respectively, and  $C$ ,  $G$  are matrices representing the conductance and susceptance matrices.

The Arnoldi algorithm [Silveira96] is employed by PRIMA to generate vectors required for applying congruence transformations to the MNA matrices, i.e.,  $V=W$ . These transformations are used to reduce the order of circuits [Kerns95]. Because of the moment-matching properties of Krylov-subspaces, the order reduced model can comply with the original model up to the first  $q$  derivatives, where  $q$  is the order of the reduced model. Models from PRIMA are able to improve accuracy compared with Arnoldi. Unfortunately, it has drawback in that model size is proportional to the number of moments (moment is matched by multiplying by the number of ports). Thus for large port numbers the algorithm leads to impractically large models.

This can be improved by using the truncated balanced realization (TBR) approach presented originally in [Moore81]. TBR based techniques can be classed as positive-real TBR (PR-TBR), bounded-real TBR (BR-TBR) and hybrid TBR [Phillips02]. Phillips et al present an algorithm based on the input-correlated TBR for parasitic

models, which shares some of their advantages such as computable error bounds. They claim that the size of parasitic models from projection-like procedures can be reduced by exploiting input information such as nominal circuit function. This algorithm can compute guaranteed passive, reduced-order models of controllable accuracy for state-space systems with arbitrary internal structure. However, TBR methods are not commonly used for reduction from three-dimensional simulation because the computational cost grows cubically with original system's size. [Kamon00] combines Krylov subspace techniques with TBR methods so that the size of TBR is reduced and potentially the computational cost can be reduced.

LTI MOR may not be applicable for many functional blocks in mixed signal systems that are usually nonlinear. It is unable to model behaviours such as distortion and clipping in amplifiers. Therefore, LTV MOR is required. The detailed behaviour of the system is described using time-varying differential equations as shown in Eq. 2-20:

$$\begin{aligned} E(t)\dot{x} &= A(t)x(t) + B(t)u(t) \\ y(t) &= C(t)^T x(t) + D(t)u(t) \end{aligned} \tag{Eq. 2-20}$$

The dependence of  $A$ ,  $B$ ,  $C$ ,  $D$  and  $E$  on  $t$  is able to capture time-variation in the system. This time-variation is periodic in some practical case such as in mixers, the local oscillator input is often a square waveform or a sine waveform, switched or clocked systems are driven by periodic clocks [Roychowdhury03].

It is known that LTV systems can not directly use LTI MOR methods due to the time-variation of the impulse response and transfer function. However, [Roychowdhury99] demonstrates that LTI model reduction techniques can be applied to LTV systems, by reformulating Eq. 2-20 as a LTI system similar to Eq. 2-17, but with extra artificial inputs that capture the time-variation. The reformulation firstly separates the input and system time variations explicitly using multiple time scales [Roychowdhury01] in order to obtain an operator expression for the transfer function  $H(t,s)$  in Figure 2-4. This expression is then evaluated using periodic steady-state methods [Kundert90] to achieve an LTI system with extra artificial inputs. Once this LTI system is reduced to a smaller one using any LTI MOR technique, the reduced LTI is reformulated back into the LTV

system form seen in Eq. 2-20. Moreover, [Phillips98] [Phillips00] claim that without the use of multiple time scales the LTV-to-LTI reformulation may still be performed using standard linear system theory concepts [Zadeh63].

Although LTV MOR may be used when modelling some weakly nonlinear systems, in most of cases nonlinear system techniques are required for such systems. A standard nonlinear system formation is based on a set of nonlinear differential-algebraic equations (DAEs) shown in Eq. 2-21, where,  $x \in R^n$ ,  $n$  is the order of matrices,  $x(t)$  and  $y(t)$  indicate the vectors of circuit unknowns and outputs,  $u$  is the input,  $q(\cdot)$  and  $f(\cdot)$  are nonlinear vector functions, and  $b$  and  $c$  are input and output matrices, respectively.

$$\begin{aligned} \dot{q}(x(t)) &= f(x(t)) + bu(t) \\ y(t) &= c^T x(t) \end{aligned} \quad \text{Eq. 2-21}$$

A polynomial approximation is simply extension of linearization, with  $f(x)$  and  $q(x)$  replaced by the first few terms of a Taylor series at the bias point  $x_0$  as shown in Eq. 2-22, where  $q(x) = x$  (assumed for simplicity),  $\otimes$  is the Kronecker tensor products operator,  $A_i = \left. \frac{1}{i!} \frac{\partial^i f}{\partial x^i} \right|_{x=x_0} \in R^{n \times n^i}$ . The utility of this system in Eq. 2-22 is that it becomes possible to leverage an existing body of knowledge on weakly polynomial differential equation systems.

$$\begin{aligned} \frac{d}{dt}(x(t)) &= f(x_0) + A_1(x - x_0) + A_2(x - x_0) \otimes (x - x_0) + \dots + A_i(x - x_0)^{(i)} + bu(t) \\ y(t) &= c^T x(t) \end{aligned} \quad \text{Eq. 2-22}$$

Volterra series theory [Schetzen80] and weakly nonlinear perturbation techniques [Nayfeh95] can then be used to justify a relaxation-like approach for this kind of systems. The former provides an elegant way to characterize weakly nonlinear systems in terms of nonlinear transfer functions [Volterra]. By using Volterra series, response  $x(t)$  in Eq. 2-22 can be expressed as a sum of responses at different orders, i.e.,

$x(t) = \sum_{n=1}^{\infty} x_n(t)$ ,  $x_n$  is the  $n$ th-order response. The linearized first order through third

order nonlinear responses in Eq. 2-22 need to be solved recursively using Volterra series as shown in Eq. 2-23 to Eq. 2-25, where  $\overline{(x_1 \otimes x_2)} = \frac{1}{2}((x_1 \otimes x_2) + (x_2 \otimes x_1))$ .

$$\frac{d}{dt}(x_1(t)) = A_1 x_1 + bu \quad \text{Eq. 2-23}$$

$$\frac{d}{dt}(x_2(t)) = A_1 x_2 + A_2(x_1 \otimes x_1) - \frac{d}{dt}(x_1 \otimes x_1) \quad \text{Eq. 2-24}$$

$$\frac{d}{dt}(x_3(t)) = A_1 x_3 + 2A_2 \overline{(x_1 \otimes x_2)} + A_3(x_1 \otimes x_1 \otimes x_1) + \frac{d}{dt}(x_1 \otimes x_1 \otimes x_1) - 2\overline{(x_1 \otimes x_2)} \quad \text{Eq. 2-25}$$

The  $n$ th-order response can be related to a Volterra kernel of order  $n$ ,  $h_n(\tau_1, \dots, \tau_n)$ , which is an extension to the impulse response function of the LTI system exhibited in Eq. 2-26, to capture both nonlinearities and dynamics by convolution. Volterra kernels are the backbone of any Volterra series. They contain knowledge of a system's behaviour, and predict the response of the system [Volterra].

$$x_n(t) = \int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} h_n(\tau_1, \dots, \tau_n) u(t-\tau_1) \dots u(t-\tau_n) d\tau_1 \dots d\tau_n \quad \text{Eq. 2-26}$$

Alternatively, a variant that matches moments at multiple frequency points is shown in Eq. 2-27, where  $h_n(\tau_1, \dots, \tau_n)$  is transformed into the frequency domain via Laplace transform.

$$H_n(s_1, \dots, s_n) = \int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} h_n(\tau_1, \dots, \tau_n) e^{-(s_1 \tau_1 + \dots + s_n \tau_n)} d\tau_1 \dots d\tau_n \quad \text{Eq. 2-27}$$

$H_n(s_1, \dots, s_n)$  is referred to as the nonlinear transfer function of order  $n$ . The  $n$ th-order response,  $x_n$ , can also be related to the input using  $H_n(s_1, \dots, s_n)$ .

Unfortunately, the size of Volterra based nonlinear descriptions often increase dramatically with problem size. [Li03] combines and extends Volterra and projection approaches using a method termed NORM (Nonlinear model Order Reduction Method) to reduce the model size. This method computes a projection matrix by explicitly



considering moment-matching of the nonlinear transfer function. For the system in Eq. 2-22, the first-order transfer function of the linearized system is seen in Eq. 2-28:

$$(s - A_1)H_1(s) = b \text{ or } H_1(s) = (s - A_1)^{-1}b \quad \text{Eq. 2-28}$$

Without loss of generality, Eq. 2-28 is expanded at the origin (0,0) as shown in Eq. 2-29, where  $M_{1,k}$  is a  $k$ th-order moment of the first-order transfer function,  $r_1 = -A_1^{-1}b$ .

$$H_1(s) = \sum_{k=0}^{\infty} s^k A^k r_1 = \sum_{k=0}^{\infty} s^k M_{1,k} \quad \text{Eq. 2-29}$$

This approach can also be applied to achieve the moments of the second-order or third-order transfer functions. Comparing with existing projection based reduction models such as [Phillips98] [Phillips00], this method provides a significant reduction of model size. A particularly attractive property of NORM is that the reduced order model produced matches certain number of transfer function moments.

[Batra04] employ NORM to generate reduced-order models of circuits from transistor level netlists. The difference from [Li03] is that Batra et al exploit least-mean-square error (LMSE) fitting techniques to find the 3<sup>rd</sup> order macromodel coefficients instead of from the model equations. Results show that the macromodels generated achieve significant decrease in model size with good accuracy to full transistor-level simulation. Unfortunately, modelling speed is not comparing with transistor level simulation (TLS). In addition the values of these results may be doubtable because this macromodel is not converted into hardware description language (HDL) for high level modelling (HLM).

Outside a relatively small range of validity, but polynomials are known to be extremely poor for global approximation [Roychowdhury04], so other methods such as piecewise approximation can be used to achieve better solutions. [Rewiński01] developed an approach termed trajectory piecewise-linear (TPWL) using a piecewise-linear (PWL) system. Initially Rewiński et al select a reasonable number of “centre points” along a simulation trajectory in the state space, which is generated by exciting the circuit with a representative training input. Around each centre point, system nonlinearities are

approximated by implicitly defined linearization. A model is generated if the current state point  $x$  is ‘close enough’ to the last linearized point  $x_i$ , i.e.,  $\|x - x_i\| < \varepsilon$ , which means that  $x$  lies within a circle of radius of  $\varepsilon$  and centred at  $x_i$ . Each of the linearized models takes the form shown in Eq. 2-30, with expansions around states  $x_0, \dots, x_{s-1}$ : where  $x_0$  is the initial state of the system and  $A_i$  are the Jacobians of  $f(\cdot)$  evaluated at states  $x_i$ .

$$\frac{dx}{dt} = f(x_i) + A_i(x - x_i) + Bu \quad \text{Eq. 2-30}$$

A Krylov subspace projection method is then used to reduce the complexity of the linear model within each piecewise region. Rewienski et al then combined all  $s$  linear models according to a weighting equation in Eq. 2-31, where  $\tilde{w}_i(x)$  are weights depending on state  $x$ .

$$\frac{dx}{dt} = \sum_{i=0}^{s-1} \tilde{w}_i(x) f(x_i) + \sum_{i=0}^{s-1} \tilde{w}_i(x) A_i(x - x_i) + Bu \quad \text{Eq. 2-31}$$

TPWL is more suitable for circuits with strong nonlinearities such as comparators, and has more advantages than PWL because as the dimension of the state-space in PWL grows one concern with these methods is a potential explosion in the number of regions which may severely limit simplicity of a small macromodel. However, Rewienski et al did not address the criterion of the training stimulus. Moreover, because PWL approximations do not capture higher-order derivative information, the ability of TPWL to reproduce small-signal distortion or intermodulation is limited. Therefore, Krylov-TBR TPWL was developed using TBR projection to obtain further order reduction [Vasilyev03].

The PWP technique [Dong03], which is a combination of polynomial model reduction with the trajectory piecewise linear method, is able to improve TPWL by dividing the nonlinear state-space into different regions, each of which is fitted with a polynomial model around the centre expansion point. These points can be selected either from ‘‘training simulation’’ or from DC sweeps. The resulting macromodel is refined

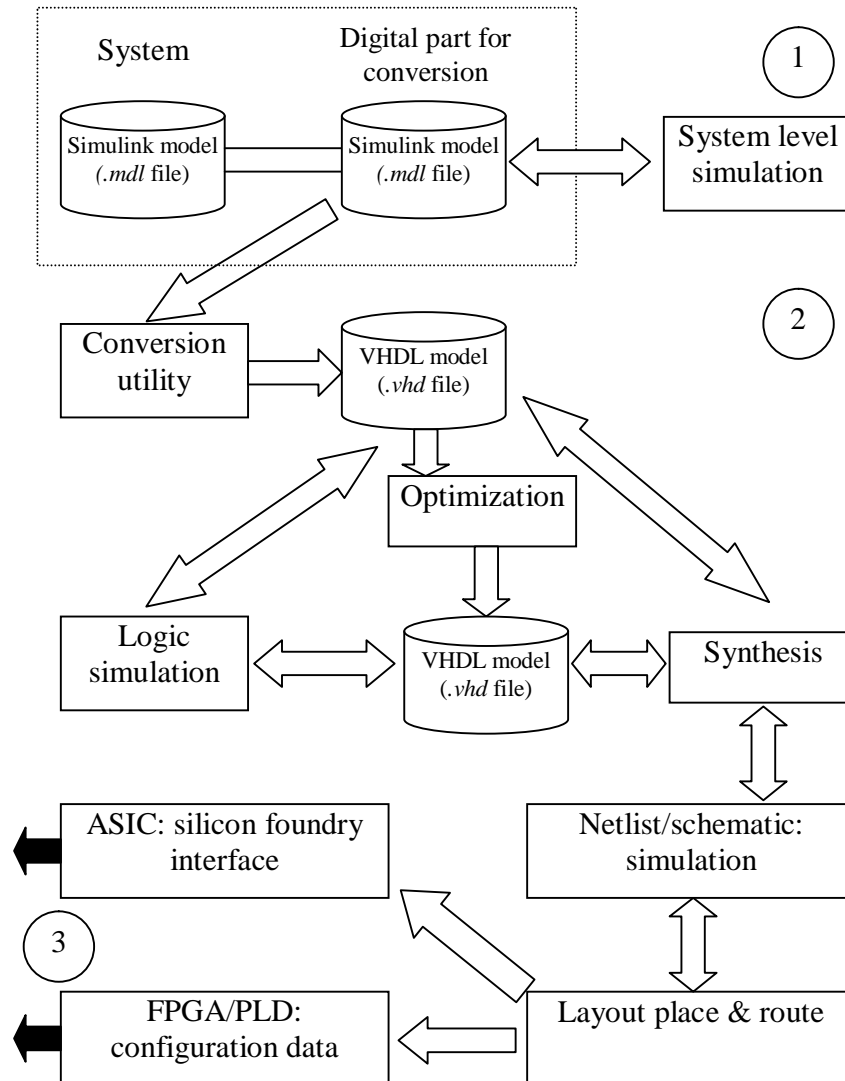
incrementally by new piecewise regions until a desired accuracy is reached. Firstly they expand a polynomial function into many points, each of them is then simplified by approximating the nonlinear function in each piecewise region to obtain much smaller size models. These models are then stitched together. Finally a scalar weight function is used to ensure fast and smooth switching from one region to another. A key advantage of PWP is that a macromodel generated can capture not only linear weakly nonlinear (such as distortion and intermodulation) but also strongly nonlinear (such as clipping and slewing) system dynamics. Moreover, fidelity in large-swing and large-signal analysis can be retained. PWP is further implemented in [Dong04] for extracting broadly applicable general-purpose macromodels from SPICE netlists such that the generated model is able to capture different loading effects, simultaneous switching noise (SSN), crosstalk noise and so on. Furthermore, a speed up of eight times simulation speed is achieved [Dong05]. However, multiple training data is used to cover different operating regions.

## ***2.4 Review of Approaches to Generating a Model in Different Computing Languages***

With the development of the HDLs and application software languages, each designer and programmer is able to design individual systems and implement individual programs and systems more efficiently and conveniently. These languages are divided hierarchically into three levels: at the highest level application languages include C/C++, or MATLAB that is a powerful scientific tool for numerical analysis [MATLAB6.5]; the next level contains HDLs such as VHDL-AMS, MAST, which provide behavioural modelling capability for both digital and analogue systems [Frey98]; the lowest level, i.e., the transistor level, has Spice-like languages such as HSPICE [Watkins]. Each language has individual advantages and disadvantages; HDLs work especially well for circuit structures, application languages such as C are efficient for general purposes. However, as systems become more complicated, particularly in very large scale integration (VLSI), one language may not be sufficient to handle all applications.

Increasingly, the gap between the high-level behavioural descriptions of the required circuit functionality in commonly used mathematical modelling tools, and HDLs can be highlighted by the requirement of seamlessly linking high-level behavioural

descriptions of electronic hardware for purposes of modelling and simulation to the final application hardware [Grout05]. [Grout00] developed a prototype software to analyze and process a *Simulink* block diagram model to produce a VHDL representation of the model. The derived model includes a combination of behavioural, register transfer level (RTL) structural definitions that are mapped directly from the *Simulink* model. This design flow is shown in Figure 2-6.



**Figure 2-6:** Overview of conversion routine within a design process [Grout05]

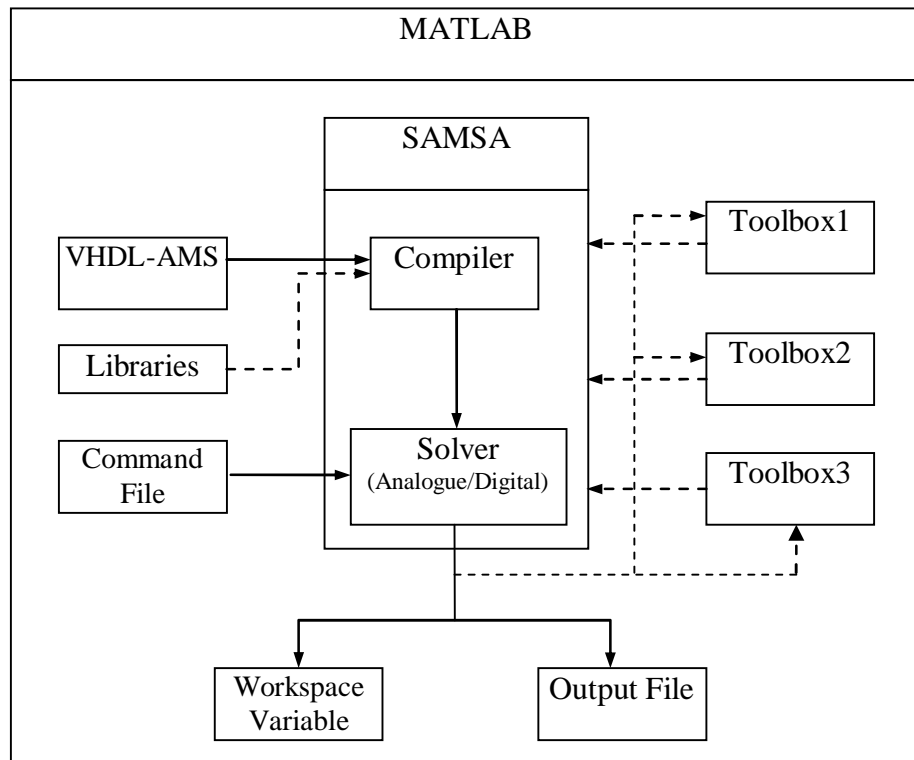
It is seen that it consists of three stages: the first one is to model the behaviour of the overall system and then define a behavioural model of the controller core. There are two model files (.mdl): one is used to store the resulting data for the system, another is to

create VHDL code (entities and architectures). The second stage includes data conversion and synthesis with technology targeting. There are two model files (*.vhd*) in this stage: the first one stores VHDL entities and architectures, and is used to perform an *optimization* routine to map the functions to a predefined architecture. The second *.vhd* file is used within a suitable design flow to compile the entities and architectures into VHDL design units for logical simulation. Synthesis is employed in the third stage to generate a netlist or schematic for further implementation on devices such as FPGAs (field programmable gate arrays) and ASICs (application specific integrated circuits). This concept allows for discrete time algorithms to be modelled [Grout01]. Unfortunately, this process is only available in the digital domain. Moreover, the conversion is implemented at the system level, whereas our research mainly focuses on model conversion at the lower level (e.g., operational amplifier) for analogue or even mixed signal domain.

[Watkins] states that in the mixed-signal domain standard VHDL and Verilog may be employed to model analogue and mixed signals, this approach is referred as Vanilla VHDL mixed signal modelling. It uses the standard arithmetic operators (primarily multiplication and addition), and standard conversion functions, for example, CONV\_INTEGER, CONV\_STD\_LOGIC\_VECTOR, and type casting. This process has several major advantages such as faster simulation, and complete portability to the most widely-used simulation environments [Smith96]. However, the Spice-like analysis (DC operation point, small signal AC) can not be implemented by Vanilla HDL, and filters are hard to be modelled accurately.

[Zorzi02] developed the software named SAMSA for the simulation of analogue and digital systems written in VHDL-AMS in MATLAB. A schematic of the system is shown in Figure 2-7. It consists of a Java compiler and a solver. The former is used to exploit the capability of directly loading Java classes into its workspace. The design unit is analyzed after the file parsing and symbols are loaded from included libraries. The MATLAB default C compiler is employed to compile two C functions generated by the parser. After compilation two dynamically linked functions are available for the VHDL-AMS system. The solver is a function call of the form  $f(y_0, y_0', C_o, F, I_a)$ , where  $y_0$  and  $y_0'$  are the initial condition vectors for the system of differential-algebraic equations

(DAE) being solved,  $C_o$  is an array of control options,  $F$  is the pointer to the run-function and  $I_a$  is used as a temporary array for sharing information. The user can set control options such as the relative tolerance and the max step allowed during transient analysis.



**Figure 2-7:** SAMSA general architecture and dependences with other MATLAB toolboxes [Zorzi02]

Simulation in SAMSA involves three steps: 1. A Spice-like command file, which describes the simulation that should be performed, and variable that should be printed and some other options. 2. The solver calls the setup-function, which initializes the simulated system, for the specified design unit and creates a structure that describes the system to be simulated in the MATLAB workspace. 3. The run-function is called, which updates some workspace vectors and variables, and an output is produced as a workspace variable, or a file in the work directory. Furthermore, the system may be more flexible when output data is post-processed or used within a particular Toolbox. [Zorzi03] also uses SAMSA, but mainly focus on the architecture of digital circuits, and a C++ compiler is utilized instead of C compiler to compile C++ code that is converted

from VHDL-AMS. Unfortunately, simulation time is not compared with transistor level simulation.

Unfortunately, all modelling approaches above are invoked under fault-free conditions, accuracy and speedup of existing models may be doubted when fault simulation is implemented because faulty behaviour may force (non-faulty) subsystems into highly nonlinear regions of operation, which may not be covered by their models.

Therefore, work based on various fault modelling techniques and other important approaches for fault detection are reviewed.

# ***Chapter 3: Literature Review of High Level Fault Modelling and Simulation***

In this chapter we will introduce techniques for analogue fault modelling and simulation based on existing publications.

In section 3.1 techniques for analogue and mixed signal test in the development of modern mixed mode ICs are discussed. Fault analysis and structural test of ICs are also introduced in this section. An overview of existing fault modelling techniques for analogue and mixed mode circuits is introduced in section 3.2 following by IFA techniques in section 3.3. The quality and accuracy of testability measurement is discussed, and improved analogue testing through the use of analogue fault modelling is described in section 3.4. In section 3.5 test coverage and test quality are introduced.

## ***3.1 Different Test Techniques for Analogue and Mixed Mode Circuits***

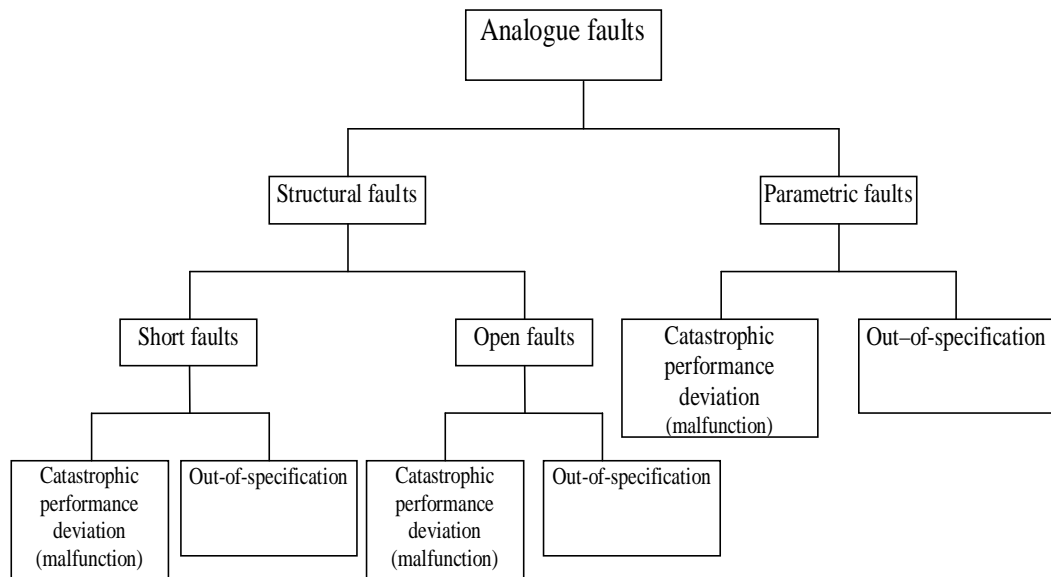
### **3.1.1 Description of IC Failure Mechanism and Defect Analysis**

A fault is defined as the electrical effect of a defect [Wilkins86]. At device level or circuit level, many factors may cause failures in different processing stages. These failures may result from any one of several different defects [Wilkins86]:

1. Manufacturing defects occur at the wafer stage in ICs, for example, short in metal interconnect and defect in gate oxide.
2. Defect during packaging: imperfect bonding and poor encapsulation.
3. Production defects on PCBs. a) components placement b) soldering
4. Operational stress: components are destroyed or dirty.

At the abstract level, faults, mainly analogue faults, are classified in [Maly88] as shown in Figure 3-1.





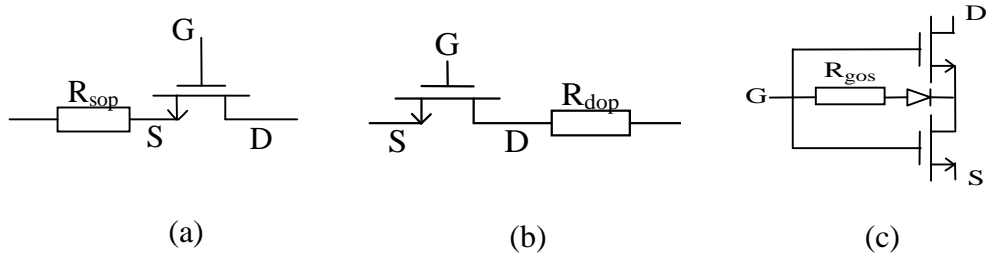
**Figure 3-1:** Analogue fault category [Maly88]

This categorisation shows that analogue faults can be divided into two main types: structural faults and parametric faults. Structural faults are random defects that cause structural deformations like short and open circuits which change the circuit topology, or cause large variations in design parameters (e.g., a change in the  $W/L$  ratio of a transistor caused by a dust particle on a photolithographic mask) [Kalpana04] [Jiang06]. Parametric faults are caused by statistical fluctuations in the manufacturing environment. Changes in process parameters (e.g., oxide thickness and substrate doping) can cause the values of components to vary beyond their tolerance levels (malfunction). Parametric faults are also caused by process gradients which produce device mismatch [Nagi93].

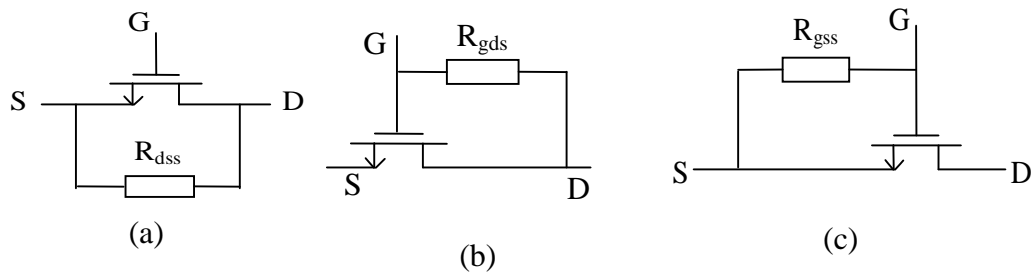
Unfortunately, the definition of this category is less applicable and limited to the modern technology such as 90nm, 65nm and beyond CMOS [Healy05]. However, it still provides useful information, particularly for this research.

It is known that the short fault mechanism is the dominating analogue fault effect, and that open fault is more difficult to model, especially when floating capacitors are produced [Bartsch96]. Short faults can be modelled as one small value resistor (e.g.,  $1\Omega$ ) connected between two nodes [Bell96] [Kalpana04]. Open faults may be modelled by using a large value resistor to connect two nodes serially [Spinks98]. The catastrophic

faults in a MOS transistor include: drain/source opens (DOP, SOP), gate-drain, gate-source, drain-source shorts (GDS, GSS, DSS); and gate-oxide short (GOS) are shown in Figure 3-2, Figure 3-3, respectively [Stopja04].



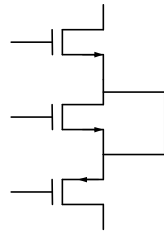
**Figure 3-2:** Source (a)/ Drain (b) open fault models; Gate oxide short model (c)



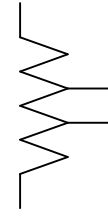
**Figure 3-3:** MOS transistor short fault models: DSS (a), GDS (b) and FSS (c)

The gate open is more difficult to model without knowing gate's dc voltage. The isolated capacitor can hold any voltage because the high impedance and leakage can charge or discharge the node of the open gate. One way to approach the problem is to set up voltage in the real device on this capacitor in order to implement simulation, for example [Caunegre95] sets gate voltage to 0V.

The relationship between out-of-specification and malfunctions are not completely isolated, for example, in a cascode circuit when two transistors are shorted, the circuit may still work, but be out of specification. Moreover, if a resistor is only shorted between two branches of its layout instead of the whole part, the circuit will still work, but the resistance changes. These two situations are shown in Figure 3-4 a), b) respectively.



a) Cascode transistor is shorted



b) Part of the resistor is shorted

**Figure 3-4:** Specification for catastrophic faults

### 3.1.2 Structural Test

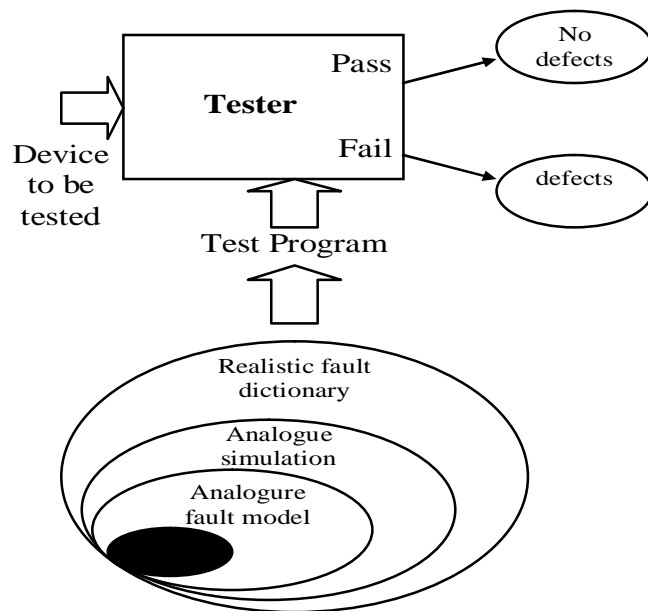
As was mentioned in the introduction, functional test is to assess whether the DUT meets its test specification and does not require detailed knowledge of circuit structure. However, this approach may fail to test all parts of the IC evenly, may offer poor fault coverage, and may be very time consuming. Therefore, structural test or defect-oriented test (DOT) is required especially in the analogue domain because it has already established in digital domain [Voo97] [Xing98] [Fang01] [Kalpana04].

The analogue fault detection and classification can broadly be divided into the following categories:

1. Estimation method: This can be further divided into the analytical (or deterministic) method and the probabilistic method. In the former, the actual values of the parameters of the device are determined analytically or based on the estimation criteria using least square criterion approach, e.g., [Simeu05]. [Simeu05] introduces a parameter optimization algorithm termed Situation-Dependent AutoRegressive with eXogenous (SDARX). It combines the Levenberg-Marquardt method (LMM) for nonlinear parameter optimization with least square method (LSM) for linear parameter estimation. However, this algorithm is only available for single-input single-output (SISO) systems, and the offset parameter is not included, which is the slowest parameter to converge because there is no signal to stimulate it. In probabilistic methods the values are inferred from the tolerance of the parameters, e.g., inverse probability method is a representative of this class. [Elias79] employs statistical simulation techniques to select parameters to be tested and then formulates the test limits on this basis.

2. Topological method: This is also known as *simulation-after-test* (SAT) method. The topology of the circuit is known and SAT method essentially reverse engineers a circuit to determine the values of the circuit component parameters.

3. Taxonomical method: This is known as *simulation-before-test* (SBT) method [Sachdev95]. This structure is shown graphically in Figure 3-5. It is seen that the fault dictionary is a key part; it holds potential faulty and fault-free responses. Inductive fault analysis (IFA) is used to determine realistic faults classes. Analogue fault simulation is implemented using Spice-like simulation over these fault classes. It generates a catastrophic defect list. During the actual test the measured value is compared with the stored response in the dictionary. If the measurements from the actual response are different from the fault-free response by predetermined criteria the fault is regarded as detectable. If the faulty response does not differ from the fault-free response by the threshold, the result is considered as undetected by the stimulus, so another stimulus is tried. The whole process is carried out for all the faults. With this approach most of the faults can be detected, and the test cost and time are reduced compared to the functional test. Some marginal failures can not be detected, which may be removed by using improved process control or detected by limited functional test.



**Figure 3-5:** Graphical representation of the realistic defect based testability methodology for analogue circuit [Sachdev95]

Due to the size of circuits and the number of faults, faults cannot be manually introduced and simulated. It is necessary to run fault simulation automatically without modifying the core simulator [Caunegre95]. [Caunegre95] develop a comprehensive system based on automated fault modelling and simulation. It comprises many parts: fault list generator, components faults catalogue, simulation control file generator and fault coverage analyzer. The fault list generator can generate a list of possible faults from a given circuit schematic. This system provides a simulator format netlist. It searches for appropriate models from the component fault catalogue for each component referenced in the fault list. The component fault catalogue provides a generic description of fault models for each library component. However, these models have to be designed and updated by experts in this domain, so it is not convenient for someone who is unfamiliar with circuit design and structure. The simulation control file generator in [Caunegre95] can create two command files by processing a previously built reduced fault list. One of files contains a command script to control the simulator, another includes parameter modification commands for parametric faults. Fault modelling and simulation can then be implemented after this process. The results are recorded in a result file.

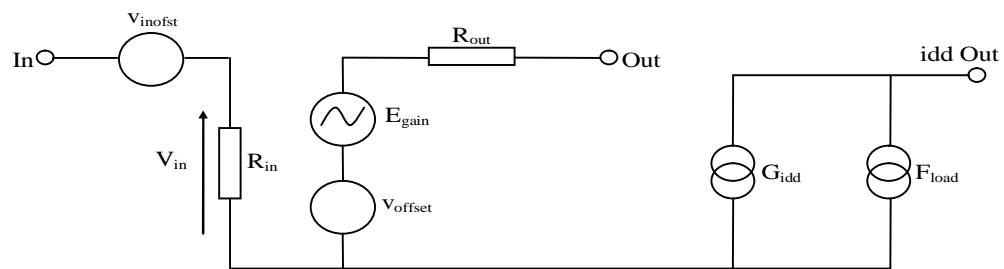
### ***3.2 High Level Fault Modelling and Simulation***

A successful model should provide high simulation speed, high accuracy, robustness and ease of use [Getreu93]. High level fault modelling (HLFM) is one of the best solutions for reducing fault simulation time [Kalpana04] [Nikitin07] [Joannon08]. Generally there are two approaches to fault simulation: 1. Injecting only the chosen fault into the low level cell by using a tool such as ANTICS [Spinks97] and observing the fault effects on the specifications of the modules at higher levels of design hierarchy [Olbrich96] [Olbrich97] [Joannon06]. 2. High level fault models are obtained by abstracting faults from transistor level into a behavioural description of their effects [Pan96] [Zwo96] [Wilson02] [Simeu05] [Joannon06]. In both cases transistor level simulation is an important and necessary step.

Generally high level fault models can be either linear or nonlinear. Both of them can be written in SPICE [Nagi92] or hardware description language (HDL) such as VHDL [Zwo00], MAST [Wilson02] and VHDL-AMS [Nikitin07] [Joannon08]. Linear models

are mainly built with linear sources and passive components [Pan96] [Yang98]. Using linear models can achieve high speed, accuracy and be implemented easily. However, the limitation is obvious. Nonlinear models can be built with nonlinear elements such as diodes, transistors, non-linear controlled sources and have moderate modelling efficiency and a wide range of operation [Zwo96] [Bartsch99]. Sometimes it is the only possibility due to a highly nonlinear behaviour.

In [Zwo96] a behavioural model for a two-stage CMOS op amp is described, as shown in Figure 3-6.



**Figure 3-6:** Macromodel of inverting and non-inverting amplifiers [Zwo96]

It includes input and output stages. The latter consists of two sections: the voltage controlled current source ( $G_{idd}$ ) and current controlled current source ( $F_{load}$ ). The former can model the power supply current of the output transistors; the latter is capable of modelling the power supply current variation due to the output load. Moreover, the output stage can model output offset voltage with  $V_{offset}$ , and fault propagation with the voltage controlled voltage source ( $E_{gain}$ ). The input offset voltage can be modelled with  $V_{inofst}$  in input stage. Furthermore, the input and output impedance can be modelled with  $R_{in}$  and  $R_{out}$ , respectively. This model can be embedded within a large circuit such as a mixer. By comparing the transistor level simulation, this behavioural model shows that not only fault-free but also faulty behaviour (short faults) can be accurately modelled for all faults, and faulty effects propagated correctly. Simulation speed is over 7 times faster than for the transistor level simulation for a transient analysis. In addition fault collapsing is also mentioned in this paper. However, only short faults are modelled and the behavioural model is only implemented with HSPICE, it may be more efficient if a HDL is adopted such as MAST. A similar model is described in [Bartsch99], this faulty

behavioural model is not only able to simulate all faulty behaviours of [Zwo96], but also can model other behaviours such as bias current ( $I_b$ ) and bias voltage ( $V_b$ ), and extra poles and zeros are also added to improve the accuracy. Moreover, the model is implemented in the hardware description language (HDL). The macromodel in [Pan96] can model not only extra poles and zeros, but also phase shift, and DC power dissipation. However, output impedance is modelled with a linear resistor, so it is susceptible to nonlinearities in circuit behaviour and loses accuracy when applied to non-linear behavioural macromodels of analogue subsystems.

Another modelling technique is presented in [Chang00]. This linear macromodel is similar to [Zwo96] and [Bartsch99]. Like [Bartsch99] voltage clipping is modelled with two diodes connected between the output and the positive and negative power supplies, respectively. However, it is not capable of modelling power supply current variation and bias current. This behavioural fault model shown in Eq. 3-1 is based on the fact that the offset voltage in most cases has a linear relationship with the input voltage for the closed-loop op amp, and is composed of two parts, i.e.

$$F_{os} = mV_{in} + k \quad \text{Eq. 3-1}$$

where  $m$  is the gain attenuation part and  $k$  is the output offset part. The values of  $m$  and  $k$  are independent of the input. With this behavioural model faulty behaviours such as stuck-at faults and other nonlinear behaviour such as slew rate can be performed well. This faulty model may be run in both DC and AC domains. Moreover, this model can be inserted into a large circuit such as a benchmark biquad filter. The behavioural model is about 11 times faster than transistor level. However, this faulty model is only implemented with SPICE. [Wilson01] used the same model in the hardware description language – MAST run on the Saber simulator [Saber04], [Kilic04] further developing it with VHDL-AMS. A comparison between transistor and behavioural models at all stages shows the latter simulation speed is faster.

A mapping technique is described in [Pan97]. The aim is to map each performance parameter set  $P$  to the corresponding macro parameter set  $B$ ,  $B = F(P)$ . A sensitivity matrix  $S$ , derived by perturbing the value of each component of the seed point  $B_0$ , along with the faults  $P_1, P_2, \dots, P_N$  is used to estimate the region of interest  $R_b$  in the macro

space, and the least square method used to reduce the model order. After  $R_b$  is obtained, a large number of random faults at the macro level are generated by randomly perturbing the values of the macro parameter set  $B_0$ . The values of the corresponding performance parameter set  $P$  can be generated by macro-level simulation. Then the dimensionality can be reduced by using a cross-correlation based technique, finally the mapping function  $B = F(P)$  is derived by utilising a neural network. By running transistor level simulation to obtain  $P$ , the fault macromodel can be obtained according to the mapping function. The technique is a good step for automating the macromodel synthesis procedure. Unfortunately only a linear op amp macromodel is used to test the neural mapping. Problems occur with large performance variations such as stuck-at faults, because the macromodel structure itself is not able to model such faulty behaviours. Moreover, the parameter mapping can be applied by the system DRAFTS using analytical design equations [Nagi93], but it is doubted whether the system can still be suitable for modern analogue and mixed mode ICs because of their increasing complexity.

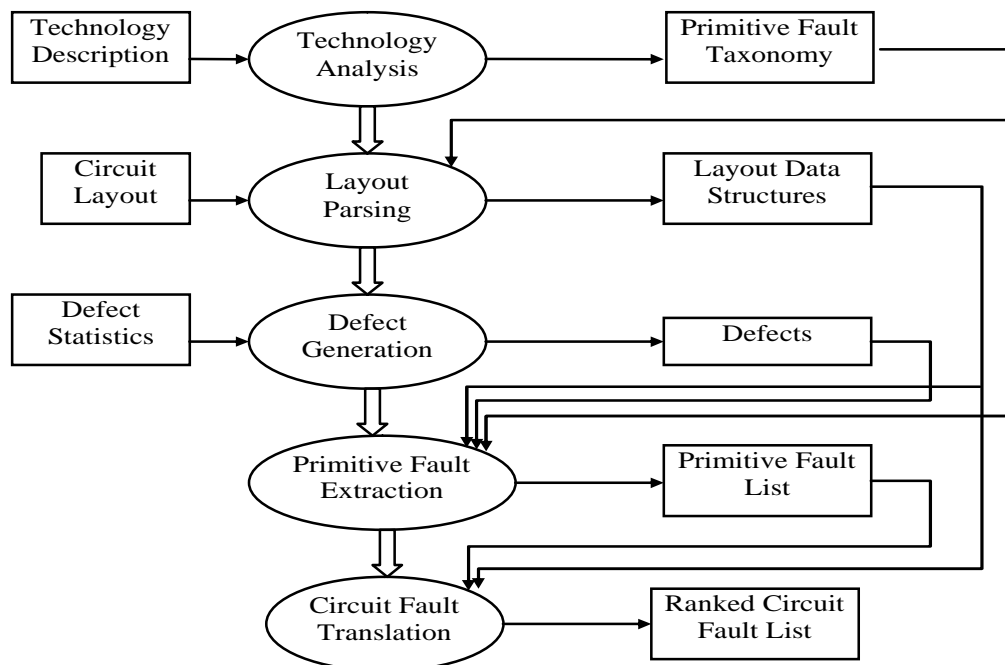
[Bartsch99] developed an algorithm that employs both linear and nonlinear fault models to implement high level fault simulation (HLFS). In this algorithm a transistor level operating point analysis is performed on the whole circuit. These points are collected for the model creation. In the case of a linear fault model the operating point information is used to ensure that the linear model is in the region where the model is parameterized. If this does not hold true a check is made to determine if the faulty behaviour can still be simulated with a linear model. If so, new parameters are derived, otherwise a nonlinear fault model has to be used. The operating point analysis can be further used as a convergence aid for the nonlinear models. After these checks the high-level fault model is injected. Then a check is made to see if the fault-free op amp is either in one of the saturation regions, or if it is in the linear region. If the op amp is in any of these regions, a linear model is inserted, otherwise, a nonlinear one is inserted to handle nonlinearity. After the injection the actual fault simulation is performed. During fault simulation it has to be ensured that the operating region for which the linear model was designed is never violated. A warning mechanism is implemented for this purpose with a simple *if-else* statement. Instead of asserting a warning message, the linear model could also be exchanged with a non-linear model. Then the simulation continues with a nonlinear



model. However, [Bartsch99] did not reach the point where these models can be switched automatically.

### 3.3 Inductive Fault Analysis (IFA)

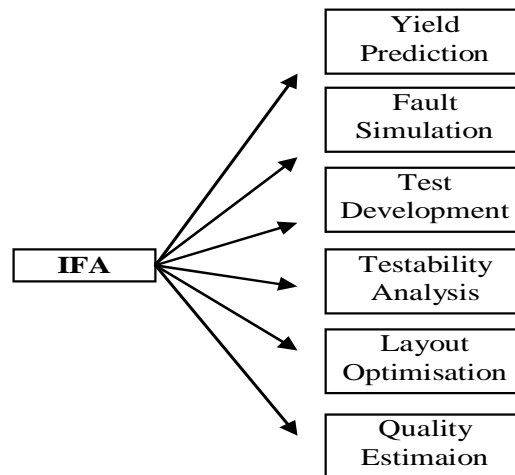
IFA was a major departure from traditional high level fault modelling and simulation because it accounted for an IC's technology, fabrication defect statistics, realistic defect occurrences, and physical layout [Ferguson88] [Jiang99] [Jiang06]. It also provides a connection between circuit level faults and technology level defects. The general structure of IFA is shown in Figure 3-7 [Ferguson88]:



**Figure 3-7:** Structure of IFA [Ferguson88]

The structure includes three columns: the left column shows the three inputs which contribute to the accuracy of the resulting fault list; five ovals represent the phases of IFA; the right column represents the output of 5 phases. The primitive fault taxonomy created by technology analysis, parameterised for the fabrication technology and considered defect types, lists all possible local changes in conductance, which may result from each defect type and the conditions which must exist for the local changes to result in a fault. The layout parsing combines pre-processing of the physical layout information with circuit extraction to facilitate the subsequent fault extraction and

translation phases. Defect generation uses information from defect statistics and layout parsing. Data statistics, gathered from an actual fabrication line is used to generate the correct mixture of defect types and sizes. These defects are transferred to the fourth phase-primitive fault extraction, together with results from the primitive fault taxonomy and layout data structures. The primitive fault extraction extracts the primitive faults by checking the geometric relationships between the defects and the circuit layout using the primitive fault taxonomy. Both the defect generation and primitive fault extraction are required once for every defect. The last phase, circuit fault translation, receives the generated fault list and exacted faults, and adds this information to the primitive faults. The final output, ranked circuit fault list, is used for many applications as shown in Figure 3-8 [Olbrich97].



**Figure 3-8:** Various Applications for IFA [Olbrich97]

IFA techniques can be used for applications such as fault simulation, design for testability and quality estimation. IFA based on modelling and simulation techniques is used to extract weighted fault lists, and then fault simulation and testability analysis take this information, together with the device netlist, to model the complete manufacturing and test process. Limitations of inaccurate fault models have been overcome by using realistic, layout dependent defect modelling techniques. A “weighting” factor may be assigned to each fault to describe how the probability of occurrence is related to quality. Using Eq. 3-2, the relative probability of occurrence of a fault  $n$  (of  $N$  faults) as  $W_n$  can be found.

$$FC = \frac{\sum_{n=1}^N F_n W_n}{\sum_{n=1}^N W_n} \times 100\% \quad \text{Eq. 3-2}$$

$FC$  is the weighted fault coverage,  $W_n$  represents the total number of circuits in the batch affected by fault  $n$  and  $N$  is the total number of fault across all circuits.  $F_n$  is a fault detection figure.  $F_n = 1$  if the fault is detectable, otherwise is 0 [Olbrich97]. This weighted fault coverage figure may be used for analogue and mixed circuits.

IFA can be used for qualifying and optimizing design for test (DFT) schemes [Olbrich96]. The method proposed in Olbrich et al requires a fault list. This can be achieved in two ways. The first one, based on IFA, is performed using a tool such as VLASIC (VLSI LAYout Simulation for Integrated Circuits), which uses either a Monte Carlo algorithm or critical area analysis to obtain the fault lists. Yield information, physical layout and process information are required to perform this. The second one is based on a transistor level fault model (short faults). The model uses the following formula for defect resistance:

$$R_f = \rho \cdot \frac{l}{W \cdot d} \approx \frac{L_{min}}{Diameter \cdot Thickness} \quad \text{Eq. 3-3}$$

where  $\rho$  is resistivity of the materials,  $l, W$  are the length and width of the particle, respectively,  $d$  is the layer thickness.  $l$  can be approximated by the minimum distance rule,  $L_{min}$ , between two lines of the same layer. Diameter and Thickness are chosen depending on the types of fault, for an open fault,  $L_{min} \approx Diameter$ , and for pinhole faults,  $L_{min} \approx Thickness$ . By comparing these two methods for the testability analysis of a self-test function in a high-performance switched-current design, it was shown that IFA generates multiple faults for modelling, the faults are weighted to reflect actual manufacturing statistics, and testability and quality prediction can be performed. However, the IFA route does not supply statistically based information on in-field failures. Moreover, it depends on an appropriate fault-model to describe the physical defects in a form which is simulatable in a circuit level netlist, so it can not adjust simulation inaccuracies resulting from inappropriate fault-models. Furthermore, the

result of this comparison is likely to be inaccurate if high level fault model is used instead of transistor level model.

An approach based on statistical process and device simulation is proposed in [Jaworski97]. It is implemented with two special programs: a statistical circuit extractor EXCESS II and a statistical process and device simulator SYPRUS. The former is used to simulate layout disturbance and performs an extraction which generates a topological netlist. A separate device model for each critical component in each netlist is computed by SYPRUS. These models are generated from the full process simulation and device modelling. A set of netlists is produced. These netlists may include models of the variation of parameters such as temperature and parasitic components. Moreover, they are used as input to a standard circuit simulator performing appropriate simulations. Results from these simulations are used to estimate the fault region for the given circuit. Only transistor level fault simulation is implemented.

Recently IFA has been employed to investigate defects and corresponding behaviour that are caused by particles contaminants introduced into the fabrication of a combdrive surface-micromachined microresonator [Jiang06], which possesses all the primitive elements used in many types of capacitive-based microelectromechanical systems (MEMS) [Jiang99]. Jiang et al used Monte Carlo (MC) analysis to find complete category of the defects. HSPICE simulation was run to evaluate misbehaviors associated with the categories of defective structure. The next step is to incorporate abstracted models of the contaminants into MEMS CAD environments. This will enable the evaluation and optimization of MEMS testability for a range variety of capacitive-based MEMS.

### ***3.4 Design for Testability with Controllability and Observability (Design and quality issue)***

According to the international technology roadmap semiconductors reports that the semiconductor industry has reached a point where testing a chip costs as much as manufacturing it [Aktouf05]. With greater functionality being packed into each design, both the time required to test each integrated circuit (IC) and the cost of the necessary testing equipment keep increasing. Design for test (DFT), which essentially means

constructing designs with easy testability in mind, is key to solving the problems of both time and expense [Aktouf05]. In the digital domain testability is mainly an issue of control and observation of deeply embedded internal nodes. The ability of setting an input condition is named “controllability” and the ability to observe the output is called “observability” [Wilkins86]. In analogue or mixed mode domain, DFT may be largely unsuccessful due to its impact on the circuit performance.

DFT consists of methods such as built-in self-test (BIST) and automatic test pattern generation (ATPG) based methodologies. [Healy05] indicated that pseudorandom BIST is displacing the latter at 90nm and beyond. This is because as the technology shrunk, the need for more and more test patterns and the need for at speed testing drove ATPG developers to various forms of data compaction and double capture timing in an attempt to identify the un-modeled failure mechanisms present in these complex devices. At 90nm and beyond these ATPG models are inaccurate and incomplete in covering all of the ways that parts actually misbehave. This has resulted in high defects per million (DPM) at manufacturing, due to the lower quality of test. “Continued attempts to extend the life of ATPG, by compressing patterns or developing faster than life timing schemes, is akin to rearranging the deck chairs on the Titanic as the un-modeled faults iceberg looms ever larger” [Healy05].

[Bratt95] proposed a DFT structure based on a configurable op amp that allows access to embedded analogue blocks such as phase locked loop (PLL). This implementation of DFT allows injection of control voltages by using this op amp. Both detection and diagnostic capabilities associated with a number of hard and soft faults are improved. [Hsu04] improved the controllability and observability by developing a current-mode control and observation structure (CMCOS) for analogue circuits with current test data. With this approach, all test points can be controlled simultaneously, also no expensive testing equipments are required for measurement. However, simulation speed is not mentioned.

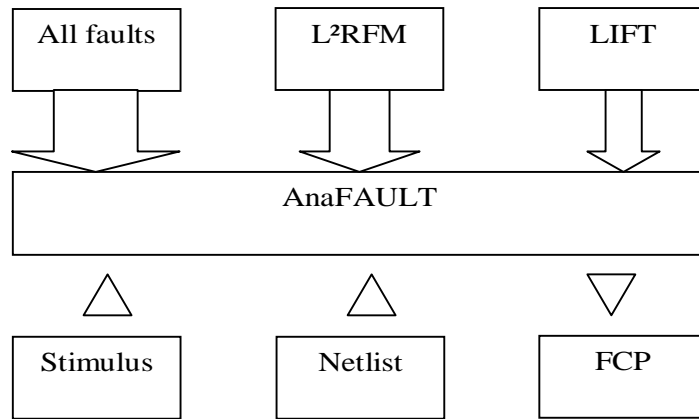
### ***3.5 Test Coverage and Test Quality***

Test coverage and quality can be investigated with fault simulation and fault modelling techniques. A fault coverage analyzer can be used to determine if the test detects each

fault by comparing the simulation measurements with the values from a fault-free simulation [Caunegre95]. A tolerance margin is defined by the designer to verify these detected and undetected faults. Decisions includes: a) if the measurement interval is outside of the allowed interval, the fault is easily detected. b) if the measurement falls inside the allowed interval, the test is passed and a fault will never be detected. c) if the measurement is half outside and half inside the allowed interval, the fault is considered as undetected. With the well chosen tolerance range, high test coverage can be obtained, and test quality can be improved.

Fault coverage ratio is  $\frac{n_d}{n_s}$ , where  $n_d$  is the number of detected faults and  $n_s$  is the number of simulated faults. Unfortunately, there is no further discussion on c) in the paper, even though some of faults can be detected if the range of interval is adjusted. A further investigation based on section c) has been performed by [Spinks97] [Spinks98]. They developed a fault simulator named ANTICS, which is able to inject faults into a transistor level model of the fault-free netlist. The range of allowed interval may be adjusted by using Monte Carlo (MC) sensitivity analysis to obtain the better fault stimuli. Therefore, the accuracy may be improved. However, Monte Carlo simulation has disadvantages: it is hard to accurately model complex circuits by using a simple system and is computationally expensive, even though fast modern computers are used [Johnson03].

[Sebeke95] defines a similar tool to [Spinks97] [Khouas00] and [Grout04] shown in Figure 3-9. It comprises an automatic analogue fault simulator called AnaFAULT and an automatic fault extraction tool, LIFT. The latter can extract sets of faults from a given analogue or mixed mode circuit layout and generate a list of realistic and relevant faults using IFA. This list represents the interface to AnaFAULT, which converts faults into fault models and fault simulation models. By this link, the tool allows a more comprehensive fault simulation, and results that are more realistic and relevant. Moreover, the overall time for the fault simulation decreases significantly compared with the assumption of the complete set of possible faults.



**Figure 3-9:** Analogue fault modelling from concept and schematic to layout. The arrows width represents the size of the fault lists [Sebeke95]

[Spinks98] presents ANAFINS as the fault generator for generation a list of possible faults from a given circuit schematic. The difference between [Spinks98] and [Sebeke95] is: ANTICS uses HSPICE as the modelling kernel, whereas AnaFAULT adopts ELDO. However, both are implemented at the transistor level simulation, so it is very CPU intensive and not suitable for complete modern complex analogue and mixed mode ICs.

The next chapter will re-implement work based on [Bartsch99] in the Hull University as the starting point of author's research and also for comparison.

# ***Chapter 4: High Level Fault Modelling and Simulation based on Other's Fault Models***

## ***4.1 Introduction***

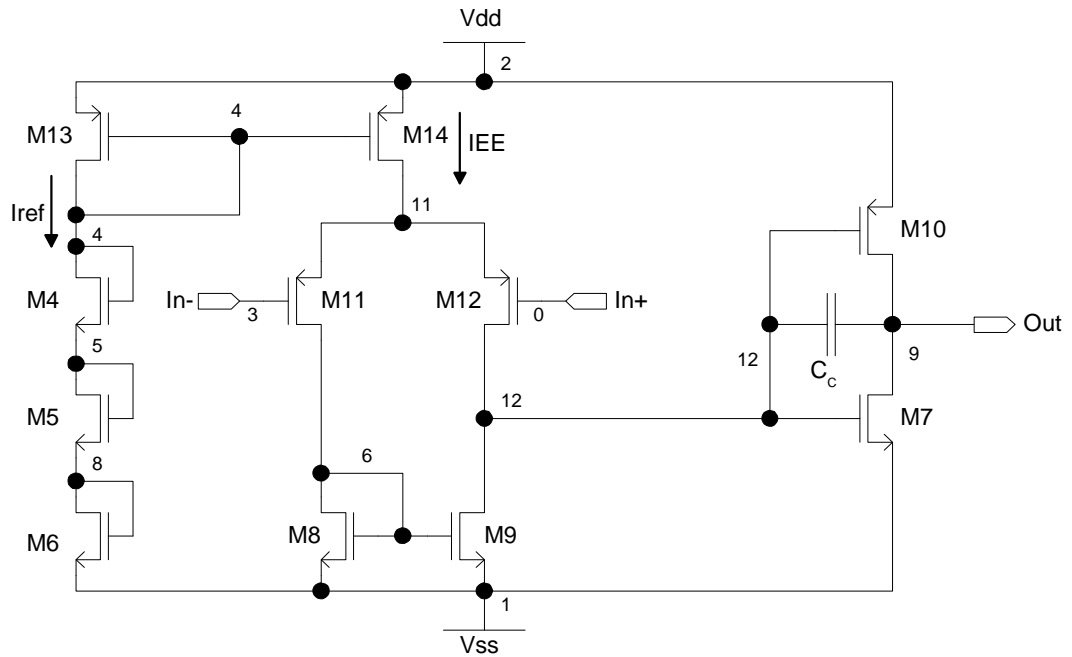
In this chapter we will reproduce some work based on Bartsch's work [Bartsch99] as the starting point of author's research. Moreover, one of the fault models will be employed in chapter 9 and 10 for comparison in terms of accuracy and speed during analogue fault modelling. The difference from Bartsch's work is that the models reproduced in this chapter are rewritten in the hardware description language (HDL) termed MAST [Saber04] instead of SpectreHDL. Simulation speed is not a key issue in this chapter because it has been discussed in [Bartsch99].

The structure of this chapter is as follows: section 4.2 introduces a two-stage CMOS operational amplifier (op amp) used for our investigation. HLFM techniques are summarised in section 4.3. Section 4.4 demonstrates these approaches with different netlist followed by the conclusion in section 4.5.

## ***4.2 Two-stage CMOS Op amp***

In this section the two-stage CMOS op amp from [Bartsch99] is used. Its design is based on [Allen87]. A schematic of this op amp is shown in Figure 4-1. It consists of an input stage and an output stage. The former is realised as a CMOS differential amplifier using p-channel MOSFETs. The differential amplifier is biased with the current mirror M13&M14. Three NMOS diodes (M4, M5 and M6) are used to keep the gate to source voltage of the current mirror small ( $V_{GS} = -1.175V$ ). The output stage (M7 and M10) is a simple CMOS push-pull inverter. Characteristic measurement of the op amp can be found in Appendix A.





**Figure 4-1:** Schematic of the two-stage CMOS op amp

Fault injection is required in order to perform TLFS. This is done using the fault injector named ANAFINS, which is the part of the transistor level fault simulator ANTICS [Spinks98] [Spinks04]. Only shorts are of interest throughout the whole thesis, other faults such as open faults will be investigated in the future work.

There are 11 transistors in this op amp, the maximum number of short faults on one transistor is 3 and therefore the number of short faults in this op amp is 33. However, only 19 of them need to be investigated because ANTICS is able to collapse redundant faults such as gate to source short on M4 ( $m4\_gss^1$ ) and drain to source short on M4 ( $m4\_dss^2$ ). It also recognises that  $m4\_gds^3$  has already been presented in the normal design and thus it is not a fault. Unfortunately ANTICS does not detect the equivalence of faults  $m4\_gss$ ,  $m5\_gss$  and  $m6\_gss$  due to the identical design of M4, M5 and M6. Therefore, only 17 faults are simulated, as summarised in Table 4-1. This shows that about 53% of the faults are stuck-at faults and only 17.6% of the injected shorts result in out-of-specification faults. Others catastrophic faults include failure to inverted gain, so the output signal follows the input one.

<sup>1</sup> short between gate and source on transistor 4

<sup>2</sup> short between drain and source on transistor 4

<sup>3</sup> short between gate and drain on transistor 4

total faults	stuck-at 2.5V	stuck-at -2.5V	stuck-at other voltages	parametric faults	other catastrophic faults <sup>4</sup>
17	8	2	0	3	4

**Table 4-1:** Characterisation of MOS transistor short faults

### 4.3 High Level Fault Models

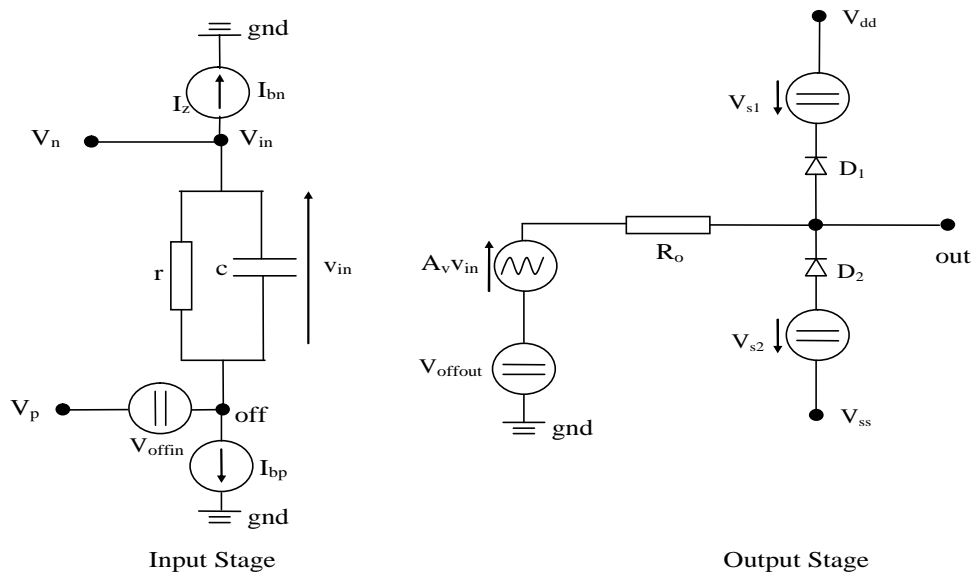
In this section we start by introducing and producing linear and nonlinear HLFMs [Bartsch99] written in MAST in subsection 4.3.1 and 4.3.2, respectively.

#### 4.3.1 Linear HLFMs

They consist of three groups: dc; dc/ac; dc and dc/ac. They are developed written in MAST [Saber04]. More details about the syntax and construction of this language are given in Appendix B.

##### 4.3.1.1 DC op amp model

The DC model shown in Figure 4-2 is a modified version of the ones published by [Boyle74] with the exception of input impedance.



**Figure 4-2:** dc macromodel (see Appendix C: C.1.1)

<sup>4</sup> They include faults such as the output is not inverted to input signal because gain  $A_v$  is not negative.

It is seen that this macromodel consists of two stages, the input stage is a standard input stage for modelling non-ideal op amp behaviour such as the input offset voltage  $V_{offin}$ , input bias current  $I_{bp}$ ,  $I_{bn}$  and dc/ac input impedance with  $r$  and  $c$ . The combination of  $r$  and  $c$  creates one pole in the left half plane as seen in Eq. 4-1.

$$W_{pole} = \frac{1}{r \cdot c} \quad \text{Eq. 4-1}$$

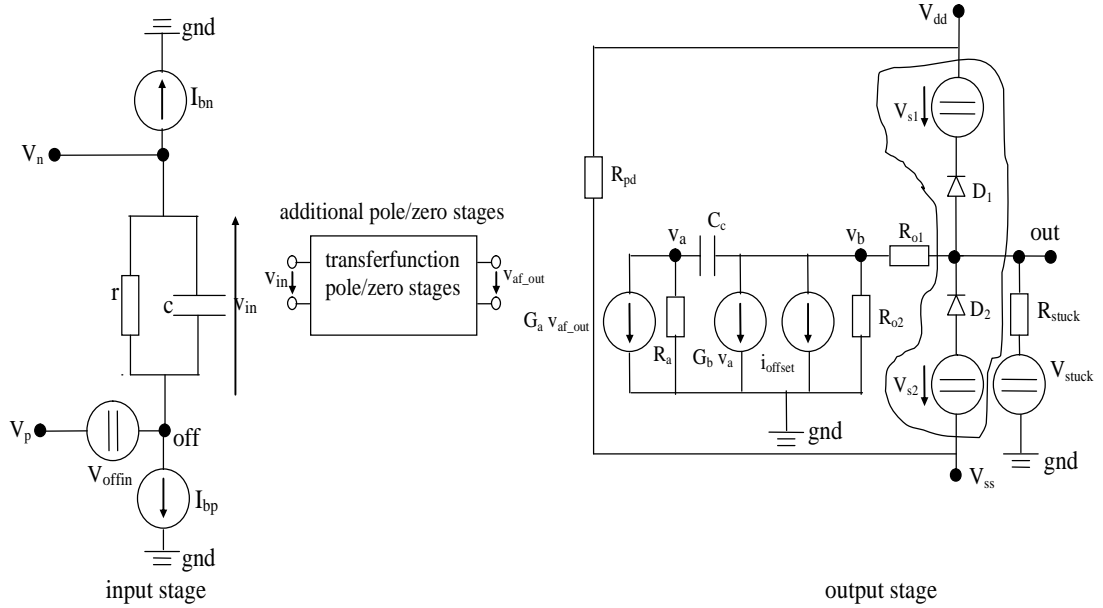
If the input impedance behaves like a capacitor,  $r$  has to be near infinity, e.g.,  $r = 1E^{20}\Omega$ , and then  $c$  can be determined by Eq. 4-2, where  $Z_{in}$  is the magnitude of the input impedance at a certain frequency  $f$  in Hertz.

$$c = \frac{1}{2 \cdot \pi \cdot f \cdot Z_{in}} \quad \text{Eq. 4-2}$$

The output stage is able to model the output impedance  $R_o$ , output offset voltage  $v_{offout}$ , gain, and output voltage clipping using diodes  $D_1$ ,  $D_2$ ,  $v_{s1}$  and  $v_{s2}$ .

#### **4.3.1.2 DC/AC op amp model**

In this section a dc/ac op amp model is designed as shown in Figure 4-3. It is seen that this model includes three stages: the input stage, a transfer function and an output stage. The input stage is the same as the one in the dc op amp model; the output stage is able to model the dominant pole of the input-output transfer function [Boyle74].



**Figure 4-3:** AC macromodel op amp (see Appendix C: C.1.2)

For the model in Figure 4-2, the number of poles and zeros are restricted to the input impedance function and for the output impedance function. With the additional stage an arbitrary number of poles and zeros can be modelled for the input-output transfer function. For certain op amps the dominant pole may not be enough, or not be modelled accurately by Boyle's output stage alone, so an input-output transfer function (as described in Appendix C) is added to obtain additional poles and zeros shown in Eq. 4-3, assuming the output voltage clipping with  $D_1$ ,  $D_2$ ,  $V_{s1}$  and  $V_{s2}$  is not included and  $R_{stuck}$  is set to infinity ( $\approx 1E^{20} \Omega$ ).

$$A_f = \frac{V_{out}}{V_{in}} = -G_a \cdot \frac{R_{o2} \cdot (s \cdot C_c - G_b)}{G_b \cdot (1 + s \cdot C_c \cdot R_{o2}) + s \cdot C_c - G_b + \frac{1}{R_a} \cdot (1 + s \cdot C_c \cdot R_{o2})} \quad \text{Eq. 4-3}$$

It is seen that with the Boyle's output stage one zero in the right half plane and one pole in the left half plane can be realised. From Eq. 4-4 the pole frequency  $f_{p,Af}$ , the zero frequency  $f_{z,Af}$  and the dc gain  $A_f$  can be derived:

$$f_{p,Af} = \frac{1}{2 \cdot \pi \cdot C_c \cdot R_a \cdot \left(1 + \frac{R_{o2}}{R_a} + R_{o2} \cdot G_b\right)} \approx \frac{1}{2 \cdot \pi \cdot C_c \cdot R_a \cdot (1 + R_{o2} \cdot G_b)} \quad \text{Eq. 4-4}$$

$$f_{z,Af} = \frac{G_b}{2 \cdot \pi \cdot C_c} \quad \text{Eq. 4-5}$$

$$A_f(f=0) = G_a \cdot R_a \cdot G_b \cdot R_{o2} \quad \text{Eq. 4-6}$$

A straight forward circuit analysis of the output stage gives the further following equation for the output impedance in the complex frequency s-domain seen in Eq. 4-7:

$$Z_{out} = \frac{v_{out}}{i_{out}} = \frac{R_{o1} + R_{o2} + s \cdot C_c \cdot [R_a \cdot (R_{o1} + R_{o2} + R_{o1} \cdot R_{o2} \cdot G_b) + R_{o1} \cdot R_{o2}]}{1 + s \cdot C_c \cdot [R_a \cdot (1 + G_b \cdot R_{o2}) + R_{o2}]} \quad \text{Eq. 4-7}$$

With this equation the pole frequency  $f_{p,zout}$ , the zero frequency  $f_{z,zout}$  and the dc output impedance can be derived as shown in Eq. 4-8, Eq. 4-9, Eq. 4-10, respectively to realise one *pole* and one *zero* in the left half plane:

$$f_{p,zout} = \frac{1}{2 \cdot \pi \cdot C_c \cdot [R_a \cdot (1 + G_b \cdot R_{o2}) + R_{o2}]} \approx \frac{1}{2 \cdot \pi \cdot C_c \cdot R_{o2} \cdot (1 + G_b \cdot R_a)} \quad \text{Eq. 4-8}$$

$$f_{z,zout} = \frac{R_{o1} + R_{o2}}{2 \cdot \pi \cdot C_c \cdot [R_a \cdot (R_{o1} + R_{o2} + R_{o1} \cdot R_{o2} \cdot G_b) + R_{o1} \cdot R_{o2}]} \quad \text{Eq. 4-9}$$

$$z_{out}(f=0) = R_{o1} + R_{o2} \quad \text{Eq. 4-10}$$

Usually  $G_b \cdot R_{o2} \gg 1$  and  $G_b \cdot R_a \gg 1$ , so  $f_{p,zout} \approx f_{p,Af}$ . This is because the pole frequency of  $Z_{out}$  and  $A_f$  are almost the same without additional *zero/pole* stages. The model is capable of modelling ac behaviour of the output impedance.  $R_{o2}$  represents the output impedance at low frequencies and  $R_{o1}$  represents the output impedance at high frequencies. The above equations can be transformed in the following equation system:

$$\text{I: } z_{out}(f=0) = R_{o1} + R_{o2}$$

$$\text{II: } f_{z,zout} = \frac{R_{o1} + R_{o2}}{2 \cdot \pi \cdot C_c \cdot [R_a \cdot (R_{o1} + R_{o2} + R_{o1} \cdot R_{o2} \cdot G_b) + R_{o1} \cdot R_{o2}]}$$

$$\text{III: } f_{p,zout} = \frac{1}{2 \cdot \pi \cdot C_c \cdot [R_a \cdot (1 + G_b \cdot R_{o2}) + R_{o2}]} \approx \frac{1}{2 \cdot \pi \cdot C_c \cdot R_{o2} \cdot (1 + G_b \cdot R_a)}$$

$$\text{IV: } f_{z,Af} = \frac{G_b}{2 \cdot \pi \cdot C_c}$$

For ‘active region’ considerations, the selection of  $R_a$  is not important. However, this means that the voltage response at node  $v_b$  is linear with  $R_a$ . If  $R_a$  is too large a value of  $v_b$  is developed during a transient excursion through the active region of the op amp, a considerable discharge or recovery time can be encountered after the active region excursion. Therefore, a small value of  $R_a$  is required to prevent these discharge delays. Empirically, Boyle suggested  $R_a$  is set to  $100\text{k}\Omega$  [Boyle74] and then the equation system can be solved for  $R_{o1}$ ,  $R_{o2}$ ,  $C_c$  and  $G_b$ . The advantage of this equation system is that one zero for the gain transfer function can be arbitrary chosen, whilst the pole and zero of the output impedance are exactly modelled [Bartsch99]. As the pole frequency of the gain is nearly the same as the pole frequency of the output impedance and it can not be arbitrarily chosen. However, if it can not be modelled well by the output stage alone additional *pole/zero* stages may be utilised to adjust the location of the first pole. Furthermore, a different equation system is used when the solution results in the negative resistance or capacitance.

$$\text{I: } f_{z,zout} = \frac{R_{o1} + R_{o2}}{2 \cdot \pi \cdot C_c \cdot [R_a \cdot (R_{o1} + R_{o2} + R_{o1} \cdot R_{o2} \cdot G_b) + R_{o1} \cdot R_{o2}]}$$

$$\text{II: } f_{p,zout} = \frac{1}{2 \cdot \pi \cdot C_c \cdot [R_a \cdot (1 + G_b \cdot R_{o2}) + R_{o2}]} \approx \frac{1}{2 \cdot \pi \cdot C_c \cdot R_{o2} \cdot (1 + G_b \cdot R_a)}$$

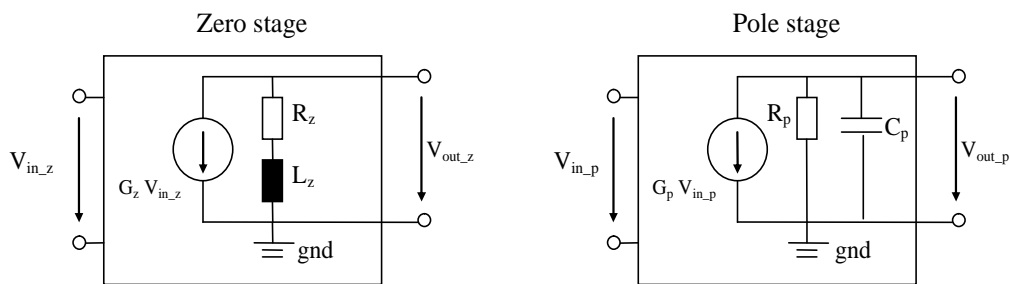
$R_{o1}$  is set to  $R_{o,ac}$ , where  $R_{o,ac}$  = ac output impedance for frequencies well above  $f_{p,zout}$  and  $R_{o2} = R_o - R_{o1}$ , where  $R_o$  = dc output impedance. The equation system is then solved for  $C_c$  and  $G_b$ . However, the zero with the solution of the above equation system is always located at very high frequencies with the investigated faults. When a zero is still required for the input-output transfer function, an additional zero stage has to be added. Parameters for  $R_{o1}$ ,  $R_{o2}$ ,  $G_b$  and  $C_c$  are obtained according to one of methods above. The value of  $G_a$  can be determined with Eq. 4-11:

$$G_a = \frac{A_f(f=0)}{G_b \cdot R_{o2} \cdot R_a} \quad \text{Eq. 4-11}$$

To model stuck-at faults  $V_{stuck}$ ,  $R_{stuck}$  and  $I_{offset}$  are introduced. The value of  $I_{offset}$  can be derived by knowing the output stuck-at voltage  $V_{stuck,out}$  and  $R_{o2}$  ( $R_{o1}$  is set to  $0.001 \Omega$ ):

$$I_{offset} = -\frac{V_{stuck,out}}{R_{o2}} \quad \text{Eq. 4-12}$$

With  $R_{stuck}$  and  $V_{stuck}$  the output can be shorted to any arbitrary potential. The potential is determined with  $V_{stuck}$ . The output impedance is determined by  $R_{o1}$ ,  $R_{o2}$  and  $R_{stuck}$ . Conveniently  $R_{o1}$ ,  $R_{o2}$  should be much greater than  $R_{stuck}$ , thus the output impedance is approximately  $R_{stuck}$ . Moreover, in order to model additional transfer function characteristics, the pole and zero stages seen in Figure 4-4 are used.



**Figure 4-4:** Additional pole/zero stages

The following transfer functions are created with those stages shown in Eq. 4-13 and Eq. 4-14, respectively:

$$\text{Pole stage: } \frac{V_{out\_p}}{V_{in\_p}} = \frac{G_p \cdot R_p}{1 + s \cdot R_p \cdot C_p} \quad \text{Eq. 4-13}$$

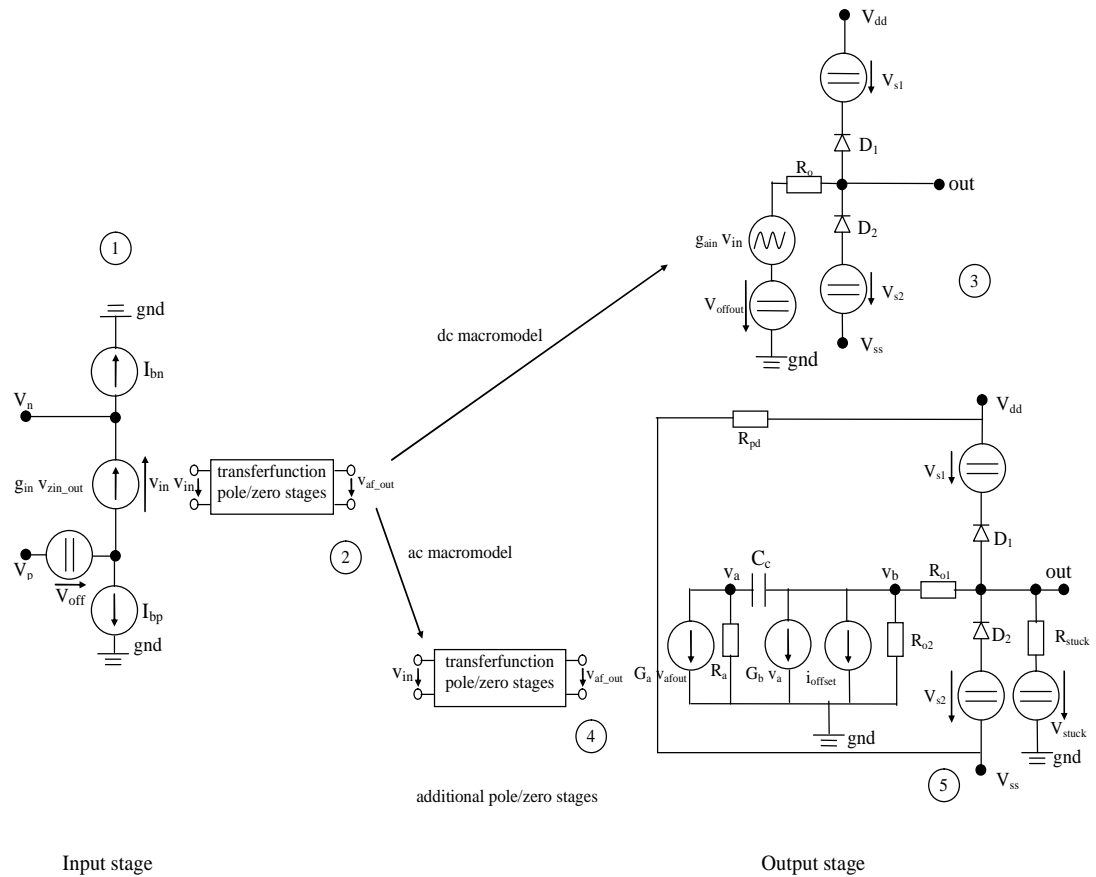
$$\text{Zero stage: } \frac{V_{out\_z}}{V_{in\_z}} = G_z \cdot R_z \cdot \left(1 + s \cdot \frac{L_z}{R_z}\right) \quad \text{Eq. 4-14}$$

Any arbitrary real pole/zero sequence in the left half plane can be achieved with these stages. For convenience  $G_p \cdot R_p = 1$  and  $G_z \cdot R_z = 1$ .

#### 4.3.1.3 DC and dc/ac Macromodel with Complex Input Impedance Function

In order to realise any arbitrary sequence of real poles and zeros for the input impedance function with the suggested pole/zero stages, elements  $r$  and  $c$  in Figure 4-2 are replaced by a voltage controlled current source (VCCS ( $g_{in} \cdot V_{zin}$ )), which is controlled by additional input impedance pole/zero stages. The dc input impedance is set by either

$G_p \cdot R_p$  or  $G_z \cdot R_z$ . The additional input impedance pole/zero stages are fed with the differential input voltage of the op amp. Figure 4-5 illustrates the concept. It is seen that the output stage comprises either block 3 or block 4 and 5 depending on whether only dc parameters are modelled or both dc and ac parameters are included. Except for block 1 and 2 the macromodel has the same architecture as the dc macromodel. All of the equations in section 4.3.1.1 and section 4.3.1.2 apply to this model.



**Figure 4-5:** Linear HLFM with arbitrary number of poles and zeros of the input impedance function  $opdc\_zin$  and  $opac\_zin$  (see Appendix C: C.1.1 and C.1.2)

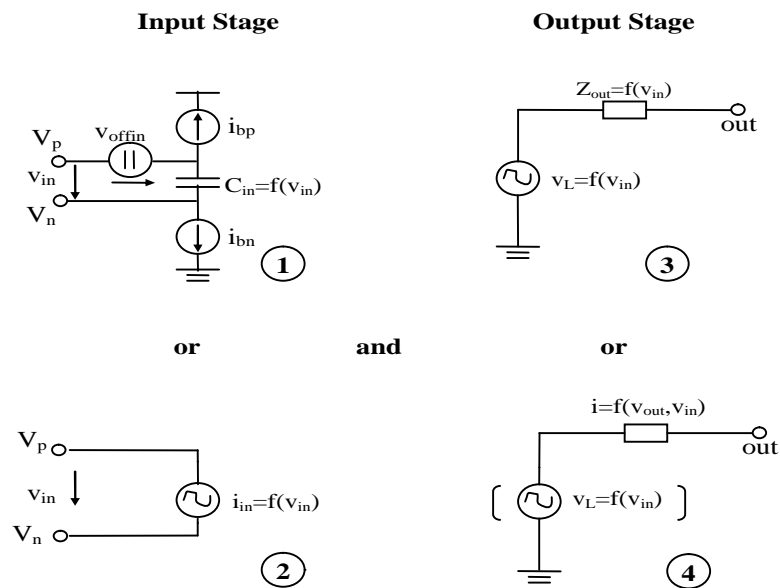
All of macromodels shown in this work have a highly modular structure as many pole/zero stages may be applied as necessary without affecting other models. Moreover, the presented models can be exchanged with more accurate ones. For example the input stage can be modified in order to model the common mode gain.



### 4.3.2 Nonlinear HLFMs

#### 4.3.2.1 Model Architecture

This nonlinear model consists of two parts as seen in Figure 4-6, both are implemented using functions ( $C_{in} = f(V_{in})$ ,  $Z_{out} = f(V_{in})$ ,  $V_L = f(V_{in})$ ,  $i_{in} = f(V_{in})$  and  $i = f(V_{out}, V_{in})$ ). The input stage comprises either block 1 or block 2. The former is used when the input impedance is modelled with a nonlinear controlled capacitor, otherwise block 2 is selected. In block 2 the input impedance is modelled with a nonlinear voltage controlled current source (VCCS) ( $i_{in} = f(V_{in})$ ). The output stage including either block 3 or block 4 determines accuracy and modelling capabilities. In block 3 the output impedance is a one-dimensional function dependent on the input voltage. The nonlinear voltage controlled voltage source (VCVS) ( $V_L = f(V_{in})$ ) models the input-output transfer function ( $V_{out} = f(V_{in})|_{no\_load}$ ). A two-dimensional function can be implemented with block 4 ( $i = f(V_{out}, V_{in})$ ) when the output impedance is dependent on not only the input voltage, but also the output voltage.



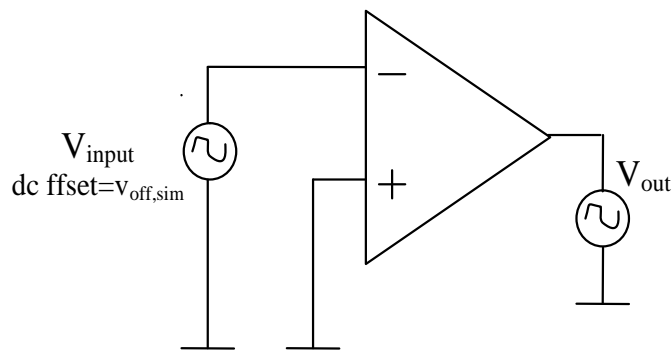
**Figure 4-6:** Nonlinear macromodels (see Appendix C: C.2 (for block 1&4))

#### 4.3.2.2 Implementation in MAST

Block 1 and block 4 are used in this section to build the nonlinear macromodel. A cubic spline interpolation is applied to pre-simulated data obtained from the transistor level

simulation. Interpolation techniques are commonly used when no mathematical relationship can be derived, or if such a relationship is very difficult to find. For a good spline interpolation the number of sample points and the sample point distance is very important [Bartsch99]. Nonlinear parts need to be simulated with more data to achieve accurate signals, but more samples require a large table size that causes low speed. Whereas fewer sample points tends to cause oscillation in regions of high curvature. Therefore, it is necessary to find a compromise between table size and model accuracy.

The op amp in Figure 4-1 is configured as an open loop amplifier as shown in Figure 4-7. This circuit is modelled at transistor level (HSPICE).



**Figure 4-7:** Linear model parameterization

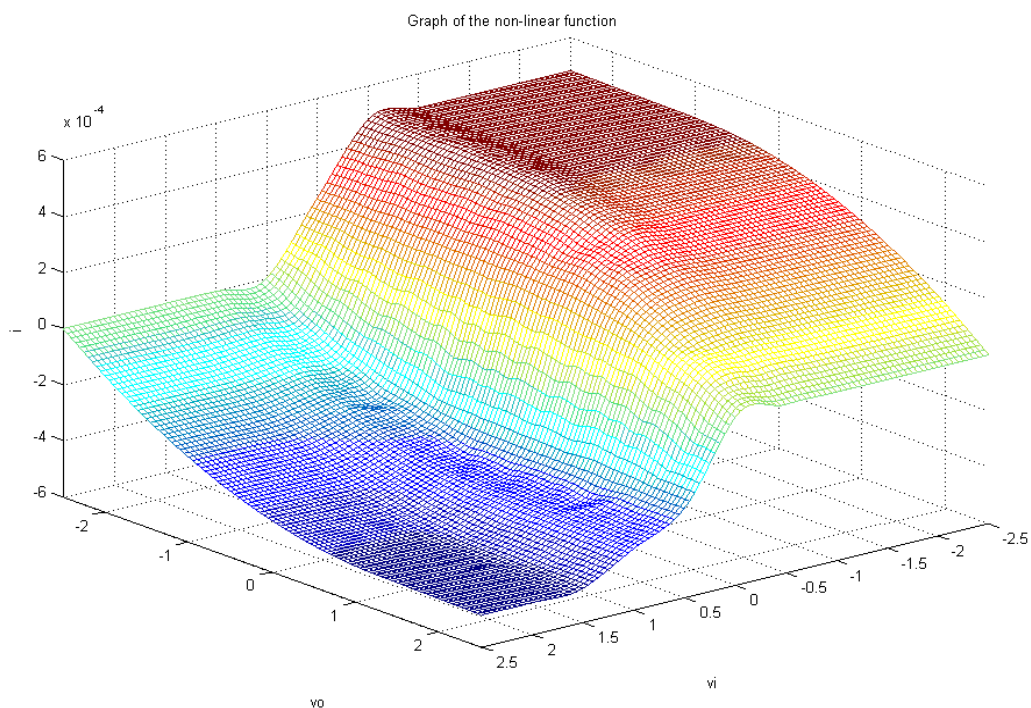
A set of dc analysis run is necessary with the strategy shown in Figure 4-8:

- A dc voltage source is connected to both the negative input and to the output of the op amp to measure the output current.
- The simulation is performed in the following way:
  - FOR dc voltage source at the input node = -2.5v to 2.5v
  - FOR dc voltage source at the output node = -2.5 to 2.5v
  - Perform DC analysis of the op amp block
  - END;
- END;

**Figure 4-8:** Algorithm of obtaining the output current function

Firstly dc analysis is run over the whole input voltage region with a step size of 500mV. It is observed that the linear range is between 0 to 10mV, so the second simulation is only implemented around this linear region with a smaller step size of 10uV, a high accuracy near the linear region is achieved. These two dc analyses gives 1,500 sample

points, which are then stored in a text file. The table look-up (*tlu*) functionality in MAST is then utilized with data interpolations to create the model that goes exactly through all their data points without involving any optimization algorithm [Saber04]. During development of block 4 it was realised that the *Cosmos* simulator does not have the function to generate the 2-D graph. Although *SaberSketch* in *Saber* can implement it, it does not have an *export* function to save signals. Therefore, MATLAB is employed to achieve the nonlinear two-dimensional output current function using the command *griddata* [MATLAB6.5] depicted in Figure 4-9.



**Figure 4-9:** Nonlinear two dimensional output current function  $i = f(V_{out}, V_{in})$

The  $x$  and  $y$  axis are spanned with the differential input voltage  $V_i$  and the output voltage  $V_o$ , respectively, the output current  $i$  forms the  $z$  axis. The nonlinear VCVS  $V_L = f(V_{in})$  is not required since this function is already included in the highly nonlinear function  $i = f(V_{out}, V_{in})$ . Such a nonlinear function can be very powerful because it is able to model the nonlinear input-output transfer function as well as the nonlinear output impedance. Such a non-linear function is very powerful. It models the non-linear input-output transfer function as well as the non-linear output impedance.

#### ***4.4 Conclusion***

In this chapter various types of high level fault models are introduced. They are divided into two categories: linear and nonlinear. The former can only be used for certain types of faults. However, the linear and nonlinear models can be used together. Those models are written in MAST and run on the Cosmos simulator. The netlist used is the open-loop amplifier, inverting amplifier and state-variable band-pass filter, respectively. Results have shown that HLFM can model faulty behaviour correctly compared with TLFS. Simulation speed is not focused in the chapter because it has been discussed and shown significantly in [Bartsch99].

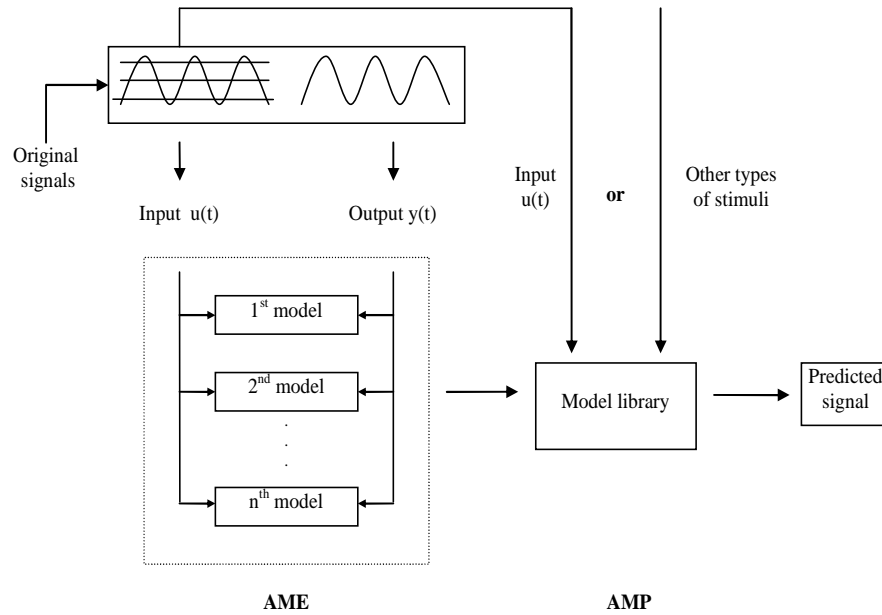
In the next chapter a novel automated model generation (AMG) approach will be introduced.

# ***Chapter 5: The Multiple Model Generation System (MMGS) for Automated Model Generation***

## ***5.1 Introduction***

Automatic generation of circuit models for handling strong nonlinearity has received great interest over the last few years. It is essential for realistic exploration of the design space in current and future mixed-signal SoCs (system-on-chips) and SiPs (system-in-packages). Generally such techniques take a detailed description of a block such as SPICE level netlist and then generate a much smaller macromodel via an automated computational procedure. The advantage of this approach is its generality. As long as the equations of the original system are available numerically, knowledge of circuit structure, operating principles and so on are not very important [Roychowdhury03].

In this chapter a novel automated model generation (AMG) approach named multiple model gradation system (MMGS) is developed for single-input single-output (SISO) macromodels capable of coping with a larger range of conditions than more conventional macromodels. The process is shown in Figure 5-1. The MMGS generates macromodels by observing the variation in output voltage error against input range. The advantage is that the estimated signal can be adjusted recursively in time to handle nonlinearity. It consists of two parts: the automated model estimator (AME) and automated model predictor (AMP). The AME implements the model generation algorithm, and the AMP uses these models to predict signals in the simulation with different types of stimuli. The system is based on a set of models  $n$ . The location of each model is decided by the thresholds seen in  $u(t)$ .



**Figure 5-1:** Schematics for the procedure of MMGS

Individual models in the n-model set are based on the RARMAX (Recursive AutoRegressive Moving Average eXogenous variables model) system [Ljung99] in the system identification toolbox in MATLAB [MATLAB6.5]. It is a single-input single-output (SISO) system and used to compute recursively for an ARMAX (AutoRegressive Moving Average eXogenous variables model) [Ljung99]. ARMAX is a robustified quadratic prediction error criterion that is minimized using an iterative Gauss-Newton algorithm that is a modification of Newton's method that does not use second derivatives. The algorithm is due to Carl Friedrich Gauss [Broyden65].

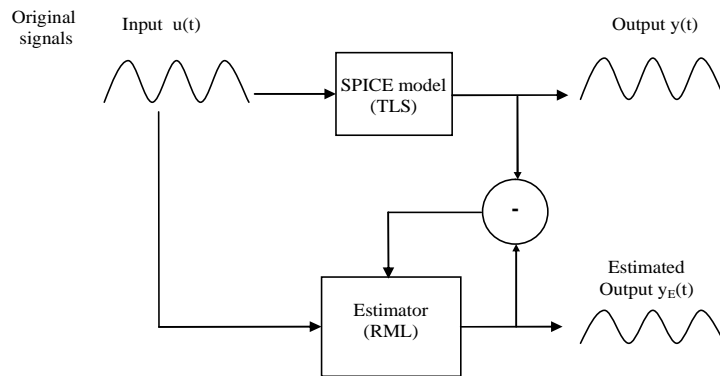
The structure can be expressed in Eq. 5-1 [Ljung99], where  $u(t)$  and  $y(t)$  represent input and output signals, respectively;  $e(t)$  is known as the noise parameter or prediction error;  $a$ ,  $b$  and  $c$  are coefficients.

$$y(t) + a_1 y(t-1) + \dots + a_{na} y(t-na) = b_1 u(t-1) + \dots + b_{nb} u(t-nb) + e(t) + c_1 e(t-1) \dots \quad \text{Eq. 5-1}$$

The method used to obtain the coefficients ( $a$ ,  $b$ ,  $c \dots$ ) is the recursive maximum likelihood (RML) method, which is an improved version of the extended least squares (ELS) technique [Ljung99] to distinguish measurement errors from modelling errors by

properly weighting and balancing the two error sources [Yeredor00]. The RML algorithm is summarised in Appendix L. Ljung was the first person to prove mathematically that RML is able to converge more reliably than ELS [Ljung75]. The key difference from ELS is that the RML does not include the prefilter, which can filter noise properties in a  $c$  polynomial that controls whether or not the estimator converges properly. The recursive method processes all samples one at a time and iterates the algorithm to obtain the estimation result during one sampling interval. Compared with non-recursive models it allows use of relatively small arrays, and is able to find the neighbourhood of a reasonably acceptable working model even when a unique solution is not available, whereas for a non-recursive model all results are processed simultaneously, which produces large matrices of stored information.

The estimation process is described in Figure 5-2, where  $u(t)$  is the input stimulus, which is used to connect both the SPICE op amp model and the estimator;  $y(t)$  is the output response from the transistor level simulation (TLS);  $y_E(t)$  is the output response using RML.

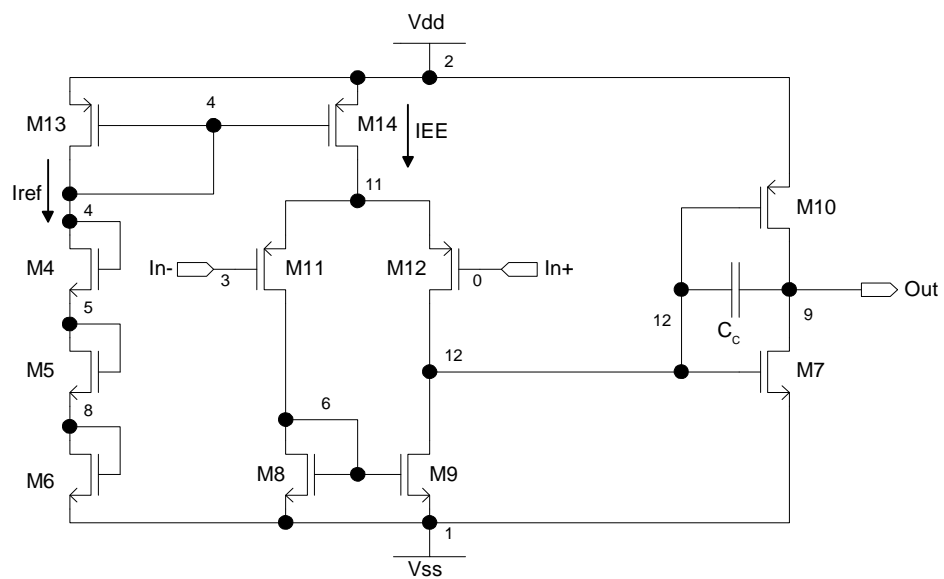


**Figure 5-2:** The general process of the estimation

The RML estimates the data obtained from TLS to produce coefficients for the RARMAX model and the estimated output signal. We then compare the estimated output signal with the one from the original SPICE model, if it is not good enough the condition of the estimator will be changed in order to achieve better results.

These generated models comprise bilinear equations [Ashenden03] that reproduce the input-output relationships of the original circuit, and can be easily converted into formats used by system-level simulation tools, e.g., VHDL-AMS (used in this work), MAST, and even SPICE subcircuits. Model order reduction (MOR) techniques may be used to reduce the order of model and improve the simulation speed.

In this thesis the SPICE model used as an example to evaluate the MMGS is the same two-stage CMOS operational amplifier (op amp) as shown in Figure 4-1, shown again in Figure 5-3. The op amp is used in open-loop configuration.



**Figure 5-3:** Schematic of the two-stage CMOS operational amplifier

The following chapter is outlined: the quality measurement based on an mathematical equation is introduced in section 5.2; section 5.3 introduces the training data for estimation; the MMGS is presented in section 5.4; section 5.5 overviews the some key factors to improve the quality of estimation; illustrative results are given in section 5.6 followed by the conclusion in section 5.7.

## 5.2 Algorithm Evaluation based on an Mathematical Equation

We need to investigate the quality of estimation by comparing the output signals. The determination of ‘closeness’ between two signals is based on the normalized evaluation range seen in Eq. 5-2. Where  $Average\_dif$  is the percentage of average difference,  $y(i)$ ,  $y_P(i)$  indicate the difference between the original signal and predicted signal at  $i$ th point.



$N$  represents the number of samples.  $y\_peak-to-peak$  is the peak- to-peak amplitude of the original signal  $y$ .

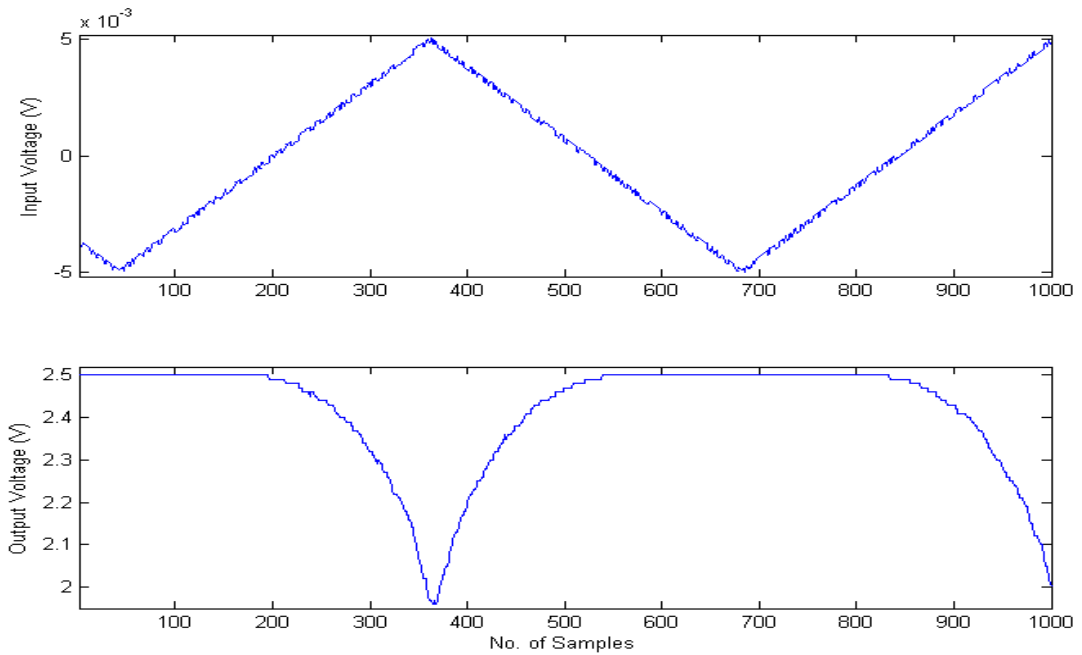
$$Average\_dif = \frac{\sum_{i=1}^N |y(i) - y_p(i)|}{y\_peak-to-peak} \times 100 \quad \text{Eq. 5-2}$$

Furthermore, the simulation speed measurement was also investigated but will be introduced in chapter 9.

### ***5.3 Training Data for Estimation using PseudoRandom Binary Sequence Generator (PRBSG)***

It is necessary to use a robust training data for estimation so that the model(s) generated can handle a wide input spectrum and nonlinearity. In this work a triangle waveform with pseudorandom binary sequence (PRBS) superimposed on it is used as the training data for estimation. The training data is generated by a pseudorandom binary sequence generator (PRBSG). The PRBSG is written in C++ and run in the visual C++ environment. A command *srand* is used to make sure that signals generated by PRBSG start from the different initial conditions, otherwise their sequences will be correlated. Another way to achieve PRBS is to build a linear feedback shift register (LFSR).

An example of the signal is shown in Figure 5-4, where  $x$  axis indicates the number of samples,  $y$  axis represents the amplitude in voltage (V). It is a 155Hz, 5.1mV triangle waveform with a 51uV PRBS superimposed on it. The PRBS has a time interval of 10us. 20,000 samples are used but only the last 1,000 samples are displayed. Connecting the signal to the negative input (In-) of the open-loop op amp, the positive input (In+) is grounded. The output signal is obtained in Figure 5-4. It is seen that the output signal is inverted compared with the input signal, and it does not cover full output range (-2.5V to 2.5V) because of the input offset voltage (5.94mV). It has only saturated at 2.5V but not cover the negative range. The full coverage data as the training data will be used in chapter 6.



**Figure 5-4:** The training data from PRBSG and output response using the open-loop amplifier

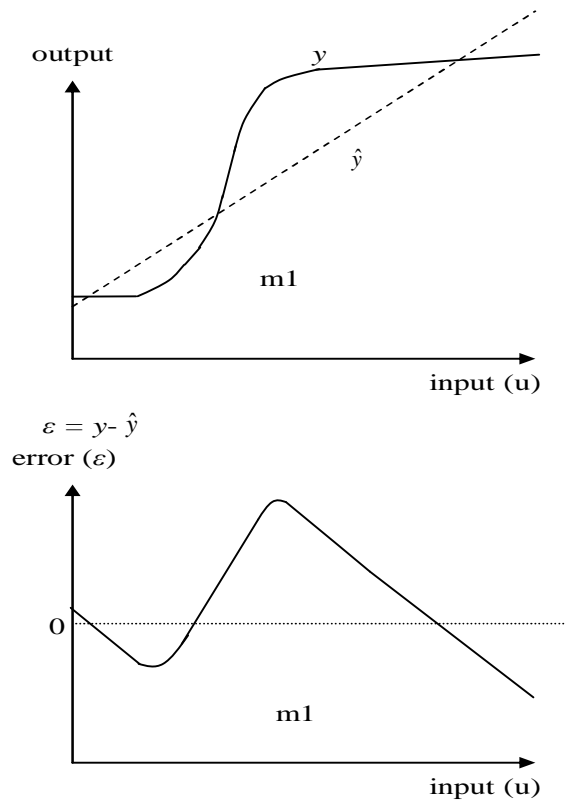
## 5.4 Multiple Model Generation System (MMGS)

In this section the multiple model generation system (MMGS) is introduced. It includes two sections: in subsection 5.4.1 we illustrate how multiple linear models are used to represent a nonlinear characteristic, and criteria used to select the family of models; in subsection 5.4.2 the automatic approach for the MMGS is introduced.

### 5.4.1 Manual Implementation

This section describes two issues: one is the selection of models to be used, and the other is the investigation of the quality of estimation.

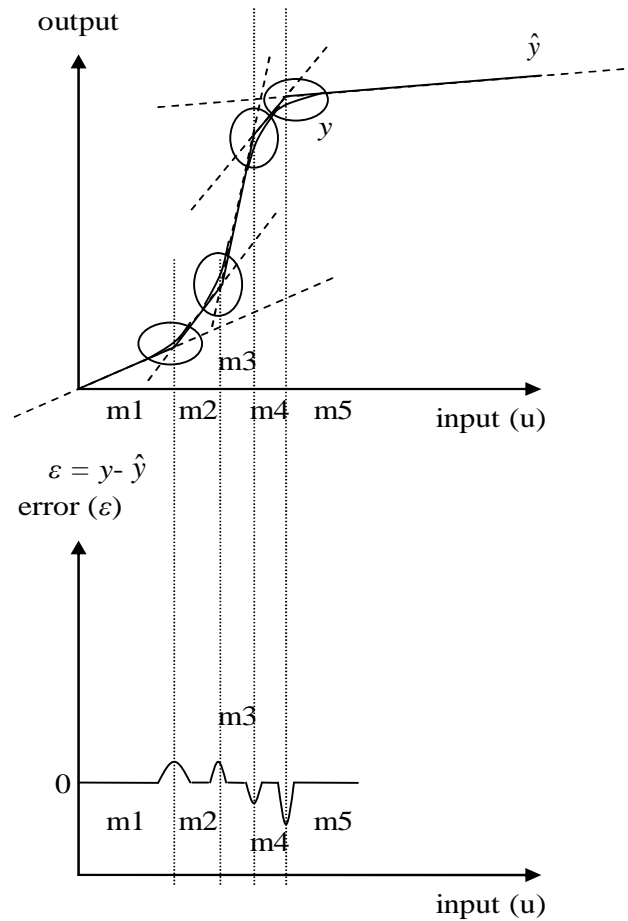
Imagine a typical input and output circuit transfer characteristic and a modelling estimate using a linear model, and corresponding error against input with one iteration. The following illustrates the theory of the issues shown in Figure 5-5. Where  $y$  is the original signal,  $\hat{y}$  is the estimated signal using the estimator based on a linear model  $mI$ , error  $\varepsilon$  is the difference between  $y$  and  $\hat{y}$ , that is,  $\varepsilon = y - \hat{y}$ . Ideally  $\varepsilon$  should be zero for all values of input.



**Figure 5-5:** Input and output circuit transfer characteristic (one linear model)

Comparing the two outputs  $y$  and  $\hat{y}$  indicates that the linear model struggles to model the nonlinear characteristics, this can be determined by observing variation of the output error.

Imagine the same circuit transfer characteristic and the modelling estimate based on five linear models  $m1$ ,  $m2$ ,  $m3$ ,  $m4$  and  $m5$  instead of one (top figure), and corresponding error for one cycle (bottom figure) shown in Figure 5-6. Where  $y$  is the original signal,  $\hat{y}$  is the estimated one by the estimator, error  $\varepsilon$  is the difference between  $y$  and  $\hat{y}$ , that is,  $\varepsilon = y - \hat{y}$ .



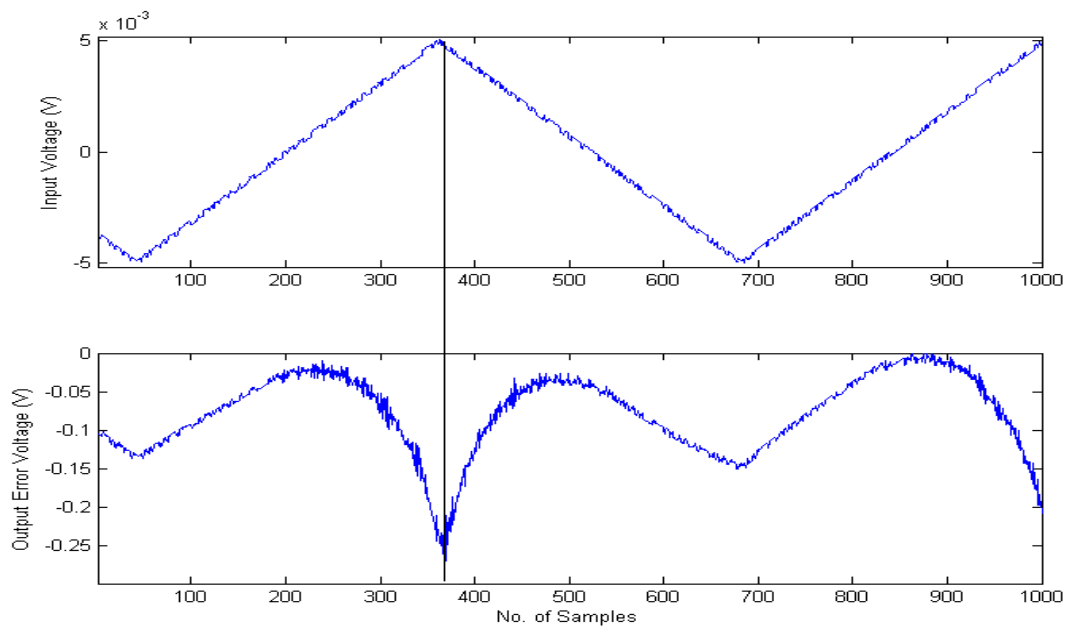
**Figure 5-6:** Input and output circuit transfer characteristic ( $m1$ - $m5$  and input threshold selected linear models)

It is shown that  $\hat{y}$  is constructed from five linear functions to form a piecewise nonlinear model. Each model is only valid for certain ranges of input indicated by the vertical lines on Figure 5-6, i.e., input thresholds. The linear parts of the signal  $y$  can be approximated reasonably well, that is,  $\epsilon \cong 0$ . The modelling error tends to be larger at the change over input thresholds between models. These have been circled in Figure 5-6 and are highlighted in the plot of error  $\epsilon$  figure. It can be seen that the amplitude of the error  $\epsilon$  is much smaller than Figure 5-5.

Therefore, in theory estimation accuracy can be improved by using multiple models, and the output error  $\epsilon$  vs input can be used to assess performance.

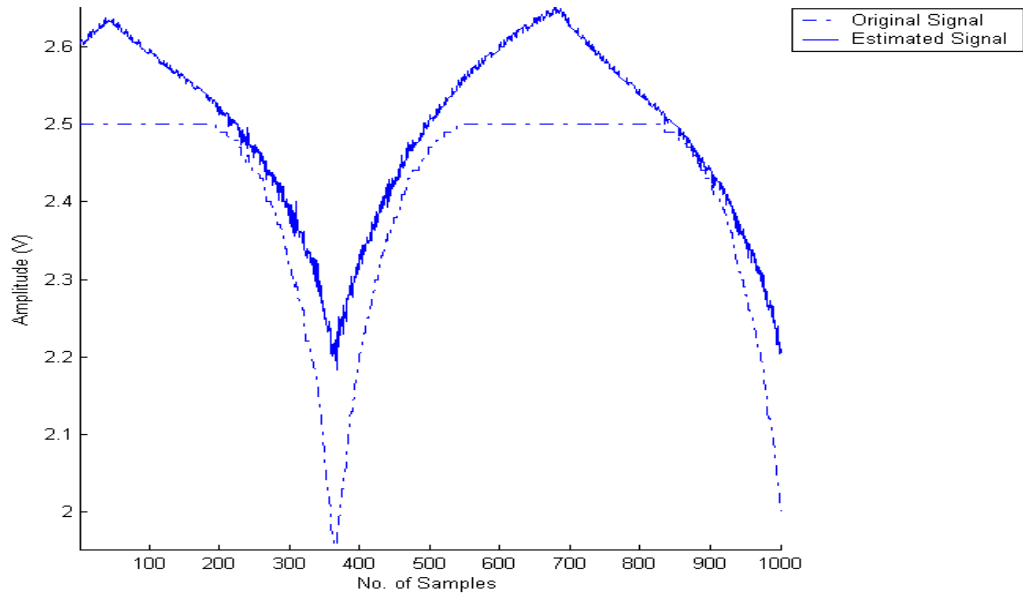
Based on the theory, the rest of the subsection 5.4.1 shows the actual simulation using the op amp in Figure 5-3. The input and output signals from TLS are shown in Figure 5-4. This process starts by observing the error using a linear model to fit the whole range of operation. In RML there are two error parameters: the innovation error *epsi* and residual error *epsilon*, both are the difference between the original signal and the estimated one. However, *epsi* is not only related to the value at current time but also the one at the previous time, which is difficult to observe. Therefore, *epsilon* is the criterion adopted for threshold creation. In this stage the number of samples does not affect the shape of the error.

We plot *epsilon* against the input voltage as shown in Figure 5-7, only last 1,000 samples are displayed. It is seen that the error varies significantly during estimation. The vertical line indicates where the largest error is and how it corresponds to the input signal.



**Figure 5-7:** The input voltage and the output error voltage

The estimated output voltage and the original signal are plotted in **Figure 5-8**, only the last 1,000 samples are displayed. It is seen that the estimated signal does not match the original signal well. This indicates that the linear model is unable to model the saturation and nonlinear parts. Thus, more models are required to handle nonlinearity.



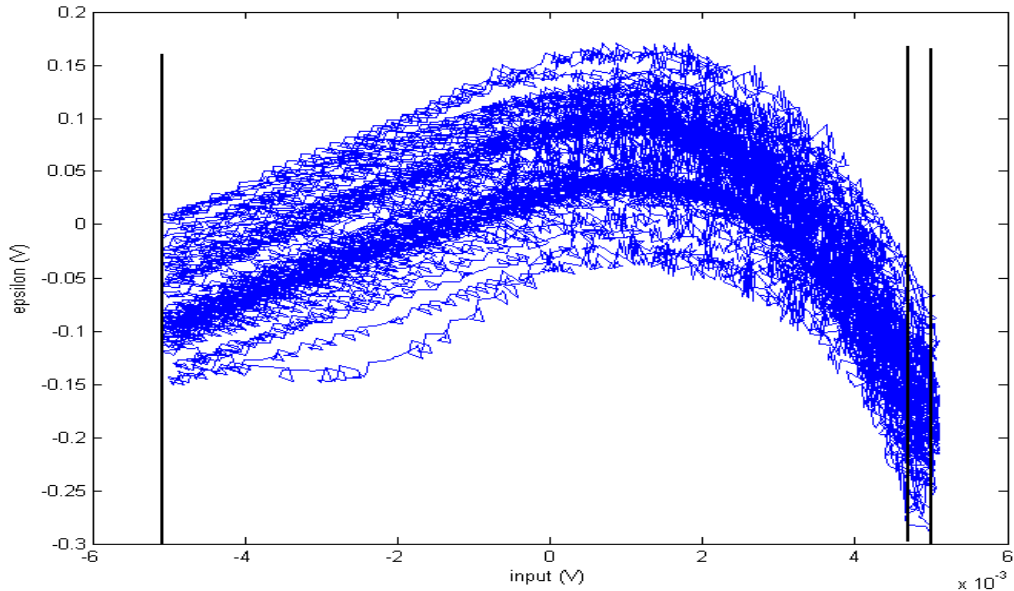
**Figure 5-8:** Comparison of output signals based on one model during estimation process

We decide to split the region up into two regions at the vertical line shown in Figure 5-7 because this is where the maximum error voltage appears. Now each new region will contain one model. Both models are then estimated, and then we observe if the error is too large to tolerate, if this is the case we split the regions again and then run the whole estimation. It is important to know that the models generated for new regions may be different from ones that have been obtained over the same range previously. This is because they are only generated using the training data in individual regions as it gradually learns over the operating time.

To observe the whole history of error behaviour over the whole input range of the system, we plot them shown in Figure 5-9. This method will be used in the rest of thesis. It is seen that the error does not appear as a uniform bar instead it is varying with the input at the time. It has the negative bias in the negative area of the input, increasing to a peak at approximate an input voltage of 1mV, and becoming more negative as the input voltage reaches its maximum (5mV).

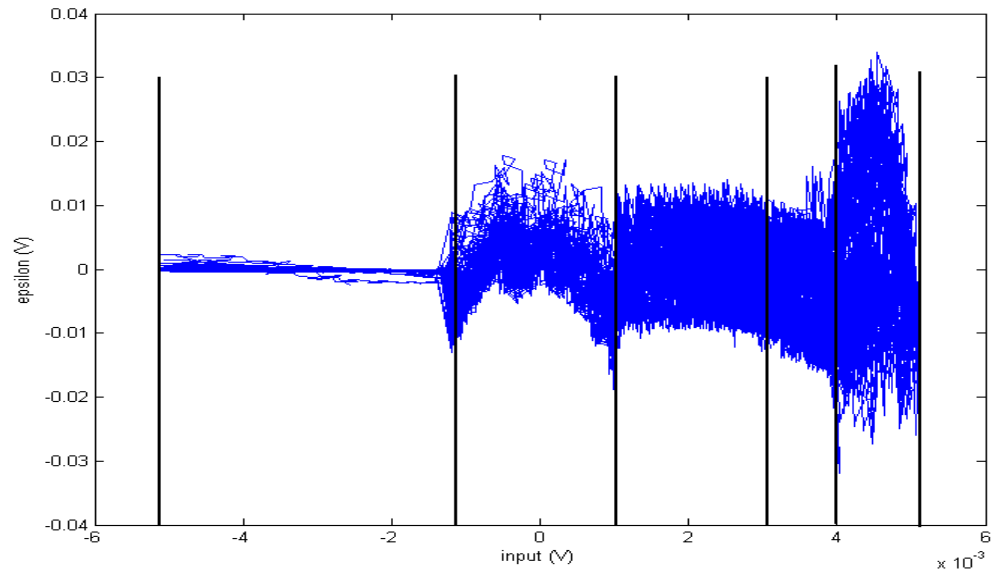
This proves that the system can not be represented by a linear model. Therefore, we search where the largest error (maximum amplitude) is, which is displayed by the

second vertical line for inputs voltage at 4.7mV (the first and last vertical lines indicate the boundary of the input). The line divides the input range into two parts, each of them contains a model.

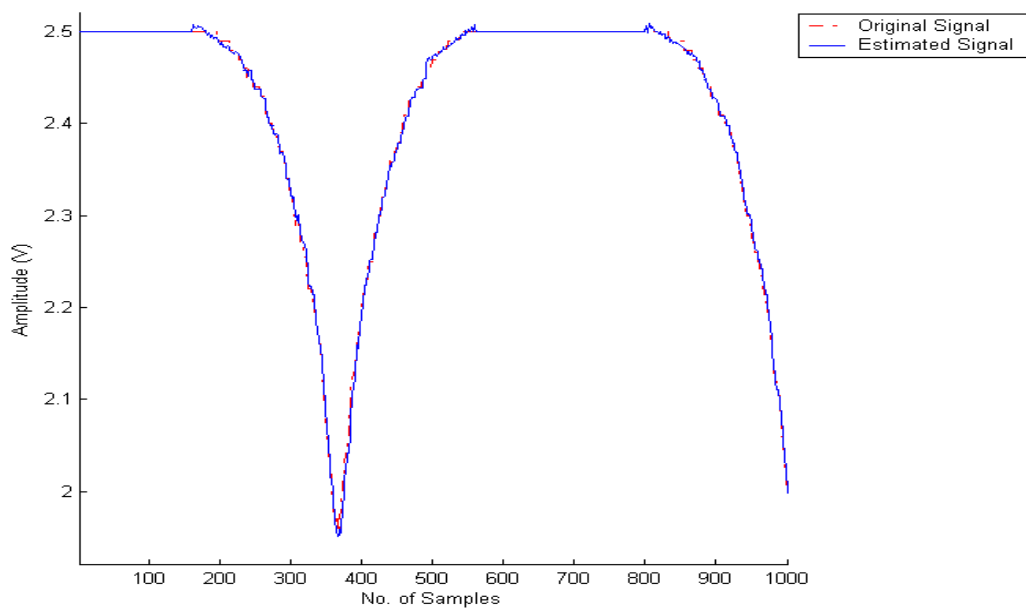


**Figure 5-9:** The variation in *epsilon* vs input range based on one model

Instead of two models, five models (within six vertical lines) are employed. The value of *epsilon* against input range is plotted in Figure 5-10, the ranges over which these models apply are indicated by the vertical divisions. It is seen that the first model between the first vertical line (-5.1mV) and second vertical line models nonlinear behaviour with good accuracy due to the straight and narrow error band. The second error voltage amplitude between the second and third vertical lines is larger than the first one. The error amplitude is similar in the third and fourth models. The error between the fifth and the last vertical lines (5.1mV) has the largest amplitude, because here we find the stronger nonlinearity and there are fewer samples in this region, but this is the best that can be achieved by the last model. However, the amplitude of error has been significantly reduced compared with Figure 5-9. The quality of the predicted signal is therefore greatly improved shown in Figure 5-11 compared with Figure 5-8. According to Eq. 5-2 the average difference between them is 0.0013%.



**Figure 5-10:** The variation in *epsilon* vs input range based on five models



**Figure 5-11:** Comparison of output signals based on five models

In this section we have demonstrated that using multiple linear models improves estimation quality. The operating ranges where the models are needed are obtained by observing the output voltage error variation against input voltage.



There is a potential problem when one model is moved to another, there may be discontinuity. More details about it will be discussed in Chapter 7.

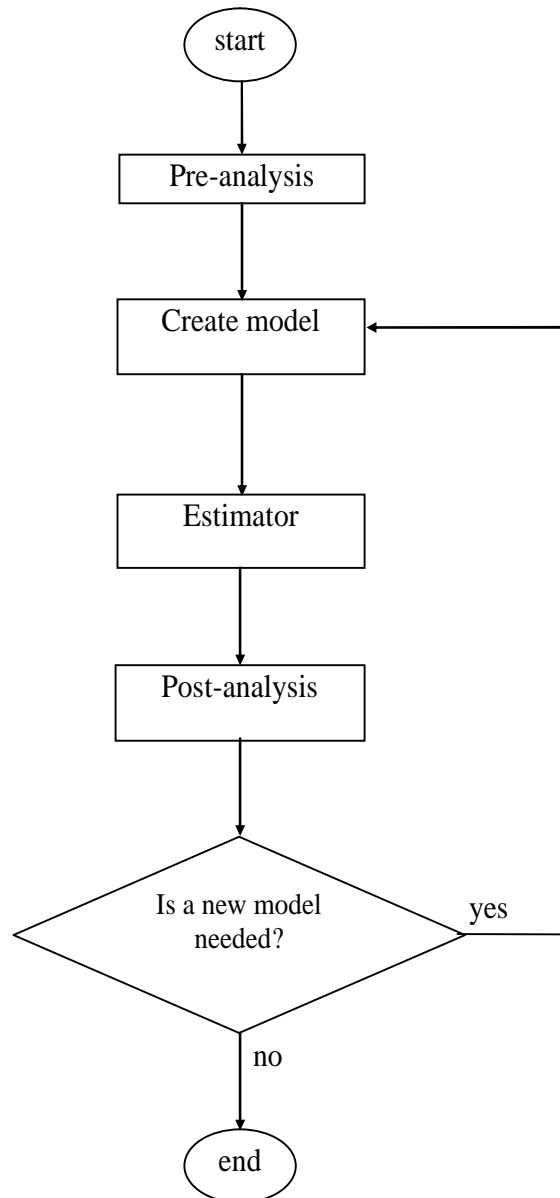
The automatic approach of the MMGS is developed in the following section.

## **5.4.2 The MMGS**

In this section we are going to introduce the automatic model generation algorithm of the MMGS. It includes the automated model estimator (AME) and automated model predictor (AMP). The former automatically generates models by observing output error voltage variation through input voltage range. The AMP uses the models to predict circuit response.

### ***5.4.2.1 The Automated Model Estimator (AME)***

The AME comprises three stages: the pre-analysis, estimator and post-analysis. Pre-analysis is mainly to set up conditions such as input range and the number of intervals for model creation and is only performed once; the estimator is used to determine the quality of output data; post-analysis is the critical step because procedures for creating models are implemented here. This process terminates when no new model is created. The general structure is shown in Figure 5-12. Its MATLAB codes can be found in Appendix H.1.1.



**Figure 5-12:** The flowchart for the AME

#### 5.4.2.1.1 The Pre-analysis

The pre-measurement is used to determine the input range and set the maximum number of sub-models. Initially, maximum and minimum values of the input signal are measured. It is then divided into a number of intervals. The number has to be even so that the middle interval is centred in the full range of input voltage. There may be a model within one of the intervals, but it is not necessary because the final decision to adding a model is made in the post-analysis.

#### 5.4.2.1.2 The Estimator

The RML estimator provides output responses and the residual error *epsilon* [Ljung99]. Its mathematical process can be found in appendix L.

The process is implemented in MATLAB [MATLAB6.5]. It starts by running through all samples using a *for* loop. The indices for creating the threshold are found with a *find* statement. A statement *min* is used to guarantee that only the smallest index is selected, and then the new model pointed by this index is generated. Parameters (*th*) and the covariance matrix (*p*) in each model need to be created and updated. The innovation error (*epsi*) and residual error (*epsilon*) are all calculated. Moreover, the prefilter needs also to be updated. The estimation is not over until all samples finish [Ljung99].

#### 5.4.2.1.3 Post-analysis

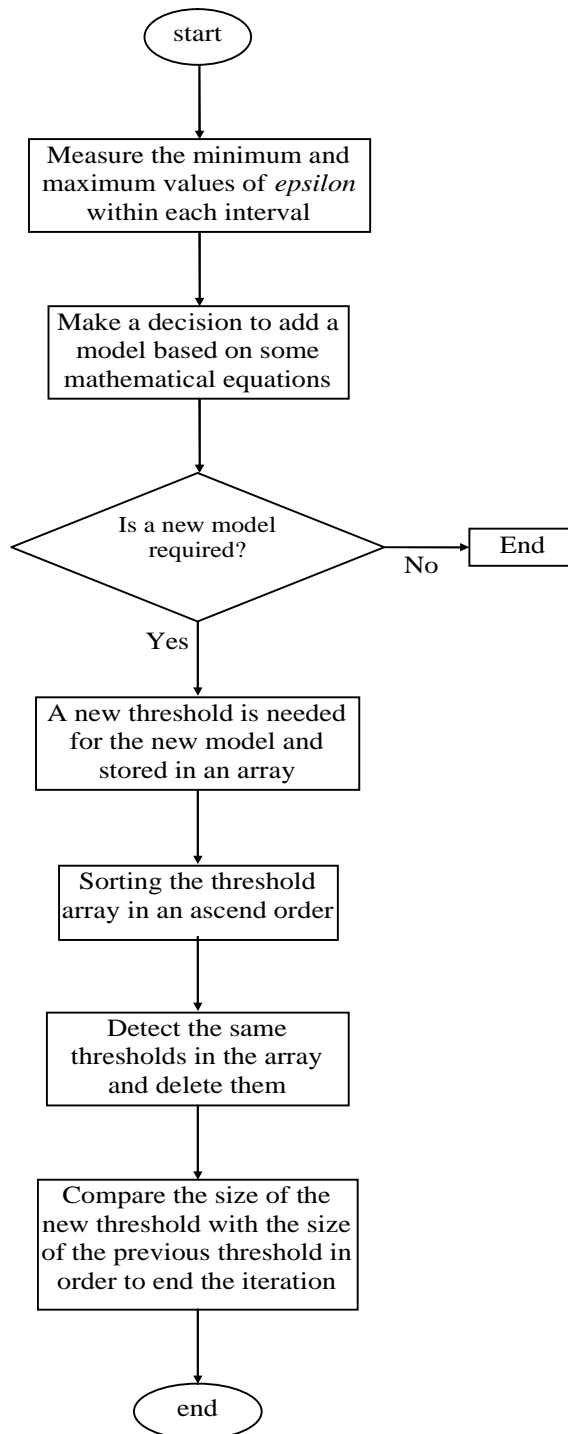
Post-analysis is a critical step because the model generation process is implemented here. The workflow is described in Figure 5-13.

The decision to add a new model to an interval of input voltage is based on Eq. 5-3, where *mediumRange* is half of the difference between the maximum amplitude of the error (*highInterval*) and the minimum amplitude of the error (*lowInterval*) for the interval. *criticalRange* is the equivalent summation. *criteria* calculated for the interval results from the comparison of these measures and that of the central interval of the simulation (*mediumRange(central)*).

$$\begin{aligned} \text{mediumRange} &= (\text{highInterval} - \text{lowInterval}) / 2 \\ \text{criticalRange} &= (\text{highInterval} + \text{lowInterval}) / 2 \\ \text{criteria} &= [\text{mediumRange} - \text{mediumRange}(\text{central})] - \text{criticalRange} \end{aligned} \quad \text{Eq. 5-3}$$

If the difference between two *mediumRange* is greater than the *criticalRange*, one model is added within the  $j^{\text{th}}$  interval (if there are *j* intervals), otherwise no action is taken. If *j* is greater than a central point, the threshold will be set at the lower range, otherwise it is set at the higher range in order to obtain the position close to the central point. In order to increase simulation speed a shift mechanism is used to delete equivalent models. Finally the new threshold array is sorted into monotonic order. Only

one model is created per iteration, because the error profile is recalculated whenever a model is added.



**Figure 5-13:** The algorithm for post-analysis

#### 5.4.2.2 The Automated Model Predictor (AMP)

The AMP is used to verify the AME system. It loads models generated by the AME to predict output responses. Its process is similar to the estimator in section 5.4.2.1.2. Unlike the estimator, the prefilter is not required. The MATLAB codes can be found in Appendix H.1.2.

### 5.5 Key Factors to Improvement of Estimation Quality

In this section some key factors that can improve the quality of estimation are discussed. They include the offset parameter, the number of samples and the training data.

The op amp already shown in Figure 5-3 is configured as an open-loop amplifier in these experiments. The estimator and predictor each comprise three programs written in MATLAB (see Appendix E).

#### 5.5.1 The Offset Parameter Related to Model Operating Points

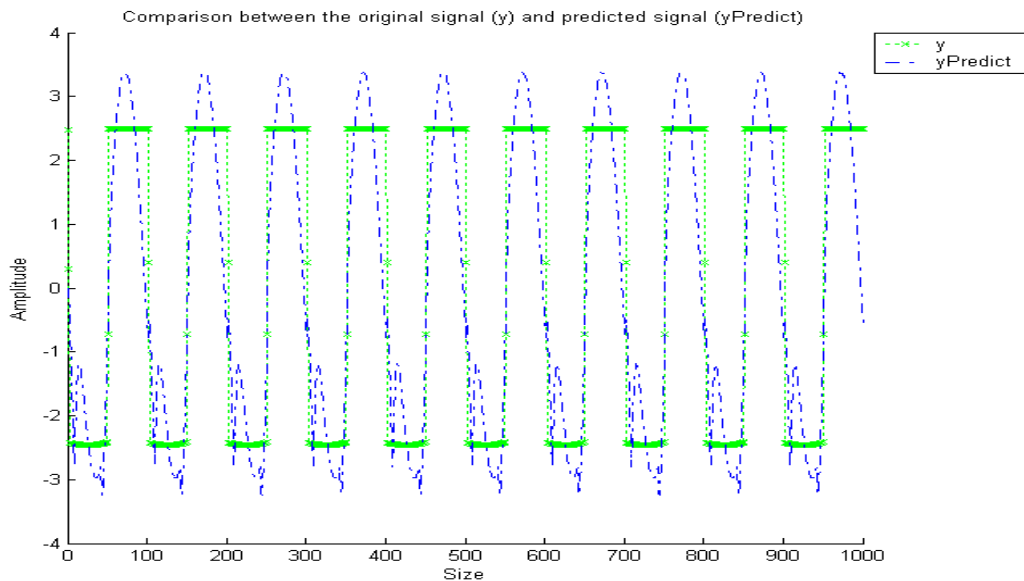
Within the multiple model generation process, each individual model is allocated in a unique range of operating input values given by the thresholds. It is known that the model created for each of the ranges has a linear relationship between the input and output seen in Eq. 5-1. As a result it must have an additional parameter termed *offset* to place its operation in the middle of its allocated input operating range. The transfer function in Eq. 5-1 is modified seen in Eq. 5-4, where  $v_{offset}$  is the offset vector and  $d$  is the coefficient for the offset.

$$y(t) + a_1 y(t-1) + \dots + a_{na} y(t-na) = b_1 u(t-1) + \dots + b_{nb} u(t-nb) + e(t) + c_1 e(t-1) + \dots + d \cdot v_{offset}$$

Eq. 5-4

The following will show how important this parameter is in the system to improve the accuracy of estimation. Two results are compared: one is with the offset, another without it. Full source code can be found in Appendix E.1 and Appendix E.2, respectively. The input signal is a sine waveform with an amplitude of 0.2V at 100Hz. It is connected to the inverting input (In-), the non-inverting input (In+) is grounded. 1,000 samples are used. Initially simulation results without the offset parameter show

that the predicted signal  $y_{Predict}$  has failed to model saturation compared with the original signal  $y$  in Figure 5-14.

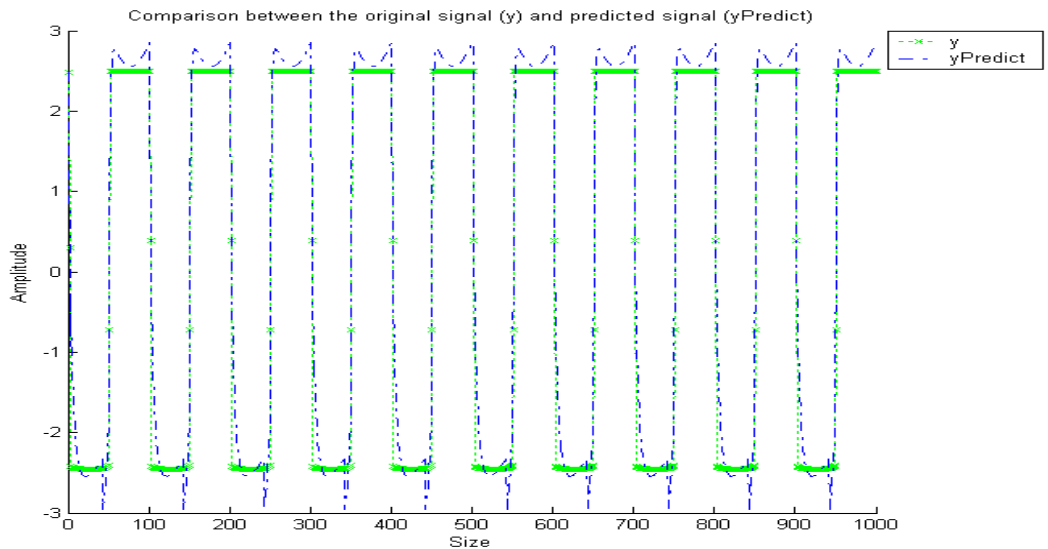


**Figure 5-14:** The predicted signal without the offset parameter

The determination of 'closeness' between two signals is based on the normalized evaluation range seen in Eq. 5-2. In this case  $Average\_dif$  is 4.88%.

After that simulation using the offset parameter was run. During simulation the predicted signal is obtained in Figure 5-15, the quality of the predicted signal  $y_{Predict}$  has been improved compared with the one without the offset parameter. According to Eq. 5-2 the average difference is 3.283%, which is less than 4.88%.

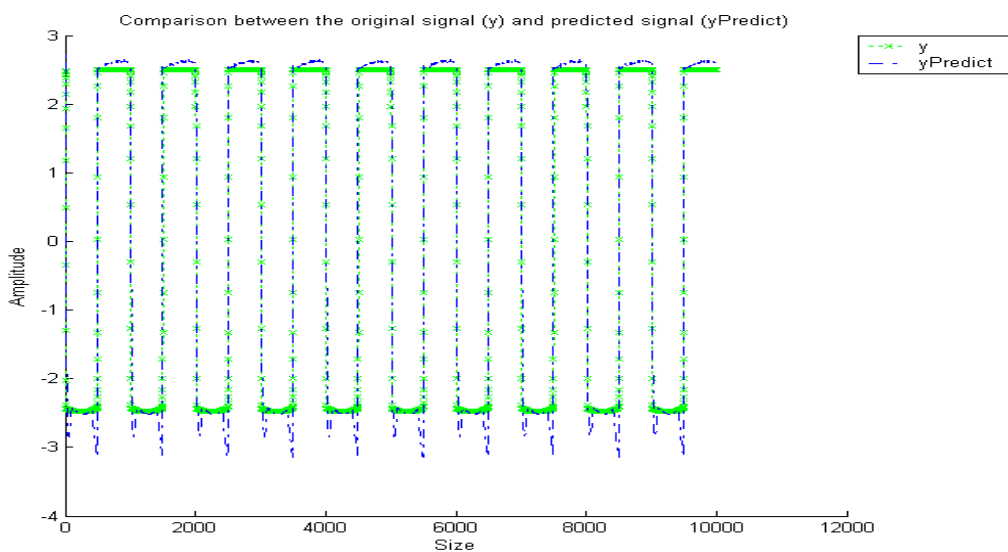
There is some noise on the nonlinear parts because the estimator can not obtain enough information on that region; it has to generate information itself. This can be improved by using a saturation detector to delete the saturated samples, thus the estimator will focus on available information. However, it has been proved that with the offset parameter the quality of predicted signal can be improved.



**Figure 5-15:** The predicted signal with the offset parameter

### 5.5.2 Quality Improvement based on the Number of Samples

In this subsection a longer simulation is performed using 10,000 samples instead of 1,000 in order to investigate if better quality estimation can be achieved. The same sine waveform signal as above is used with the open-loop amplifier. The predicted output signals  $yPredict$  is shown in Figure 5-16. Illustrative results show that with 10,000 samples spikes have been reduced. According to Eq. 5-2 the average difference is 1.8946%, which is less than 3.283%. It has proved that sometimes quality of predicted signal can be improved with more samples.



**Figure 5-16:** Signals from the predictor with 10,000 samples

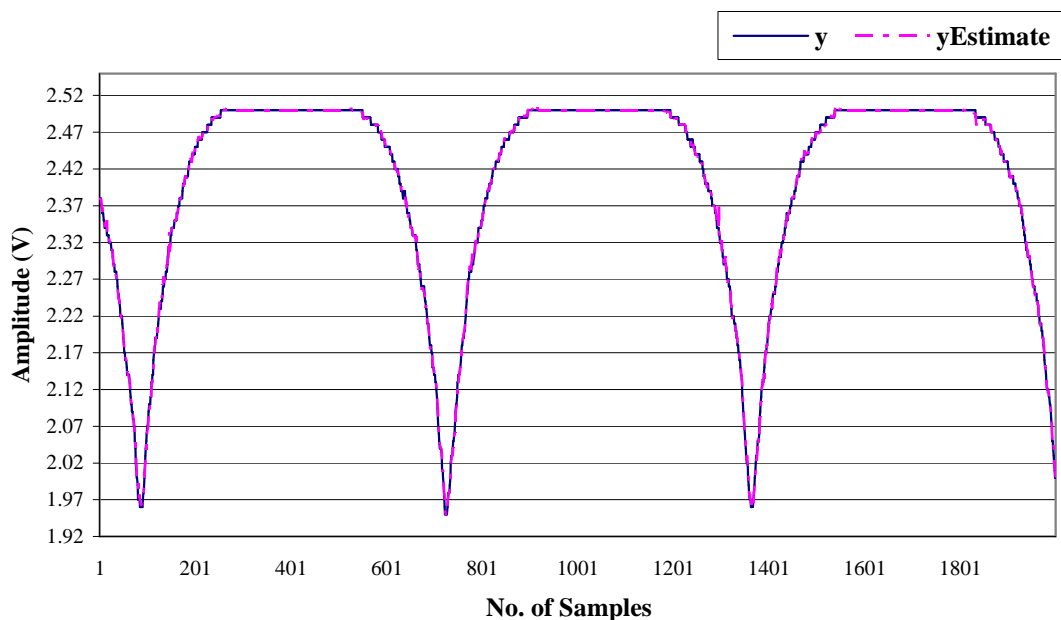
However, this is not always true, for example models on saturation regions, because samples in the models may not be informative. Illustrative results can be found in Appendix F.

Moreover, many methodologies for improving quality of the MMGS have been developed and described in Appendix G.

## 5.6 Experimental Results

### 5.6.1 Simulation for Nonlinearity

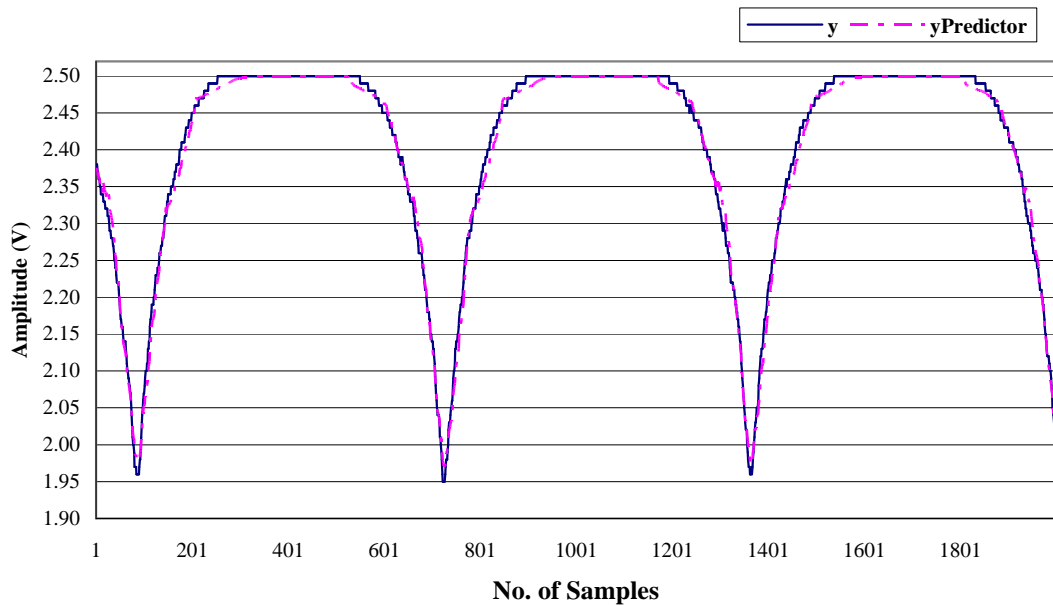
In this subsection the transistor-level open-loop amplifier shown in Figure 5-3 is modelled with inclusion of strong nonlinearity. Five models are generated, the stimulus for both the AME and AMP is the same PRBS in Figure 5-4 (20,000 samples). The estimated signal  $y_{Estimate}$  and the original signal  $y$  are plotted in Figure 5-17 (last 2,000 samples). The  $x$  axis is the number of samples, and the  $y$  axis is the output voltage (V). It is seen that  $y_{Estimate}$  matches  $y$ . The average difference measurement is 0.3746%. This proves that this algorithm for automatic model estimation has worked successfully.



**Figure 5-17:** The estimated signal from the AME system



The AMP system then uses these models to predict the nonlinear system with the same input stimulus in Figure 5-4, signals are shown in Figure 5-18 (last 2,000 samples).



**Figure 5-18:** The signal from the AMP system

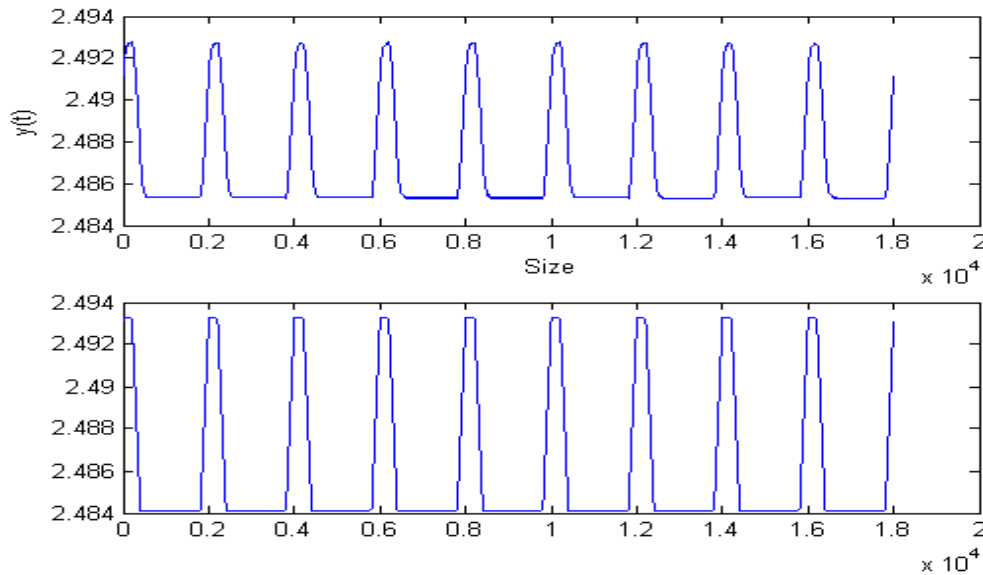
It is shown that both the predicted signal  $y_{Predictor}$  and the original signal  $y$  are very close. The average difference measurement is 0.572%.

Although there is some difference on the saturation part in the predicted signal, because the process of prediction is “dead reckoning”, that is, in the predictor we derive the output response from the input directly without any information on output to work on, so we accumulate errors, whereas in the estimator we use genuine data from the real process.

### 5.6.2 Validation Test for MMGS Generated Models via Time-domain (transient) Simulations

In this subsection another stimulus is used in order to validate the models from section 5.6.1 (five models). The circuit used is the open-loop amplifier. A square waveform stimulus was used based on the pulse source function in HSPICE: PULSE ( $v1$   $v2$   $td$   $tr$   $tf$   $pw$   $per$ ), where  $v1$  is the initial value of the voltage or current before the pulse onset,  $v2$  is the plateau value,  $td$ ,  $tr$  and  $tf$  represent the delay time, rising time and falling time, respectively,  $pw$  is the pulse width,  $per$  is the pulse repetition period in seconds. In this

case these parameters are chosen as follows: PULSE (-0.2mV 0.2mV 100us 100us 100us 700us 1000us). All conditions in the system remain the same as above, the predicted signal is seen in Figure 5-19.



**Figure 5-19:** Predicted square waveform based on models from the MMGS

It is seen that the shape of the predicted signal (bottom) is close to the original one (top). The average difference between them is 0.524%.

### 5.7 Conclusion

In this chapter an AMG based approach termed multiple model generation system (MMGS) has been developed for SISO models. It consists of the automated model estimator (AME) and automated model predictor (AMP). A PRBS-signal generator is used to generate robust training data for the AME. Results show that these generated models are able to model nonlinear behaviours, and model circuits subjected to various stimuli such as the pulse waveform with good accuracy.

In the next chapter the multiple model generation system (MMGS) for multiple-input single-output (MISO) models based on recursive maximum likelihood (RML) is implemented.

# ***Chapter 6: The Multiple Model Generation System (MMGS) for Multiple-Input Single-Output (MISO) Systems***

## ***6.1 Introduction***

In this chapter the multiple model generation system (MMGS) for multiple-input single-output (MISO) models based on recursive maximum likelihood (RML) is implemented in MATLAB [MATLAB6.5]. The reason for developing a MISO model is that the system of interest is a two-input operational amplifier (op amp). Existing approaches for MISO models such as the recursive prediction error (rpem) method or the recursive pseudo-linear regression (rplr) method [MATLAB6.5] can also be used. The difference between them is that different gradient approximations are utilized [Ljung99].

Currently, the model generation process is dependent on threshold measurement of one input, the other input is not responsible for model generation and selection, because this reduces the complexity of the process for our investigation. The generation method based on both inputs will be investigated in the future work. The input for the model generation has the most nonlinear relationship, so the input and output transfer characteristic is the one with the worst nonlinearity, which is the one used to generate models, the other input is more linear in the relationship with the output.

This chapter is outlined as following: section 6.2 overviews the MMGS for MISO models. Experimental results are given in section 6.3 to verify the system works. In section 6.4 the conclusion is given.

## ***6.2 Algorithm on the MMGS for MISO Models***

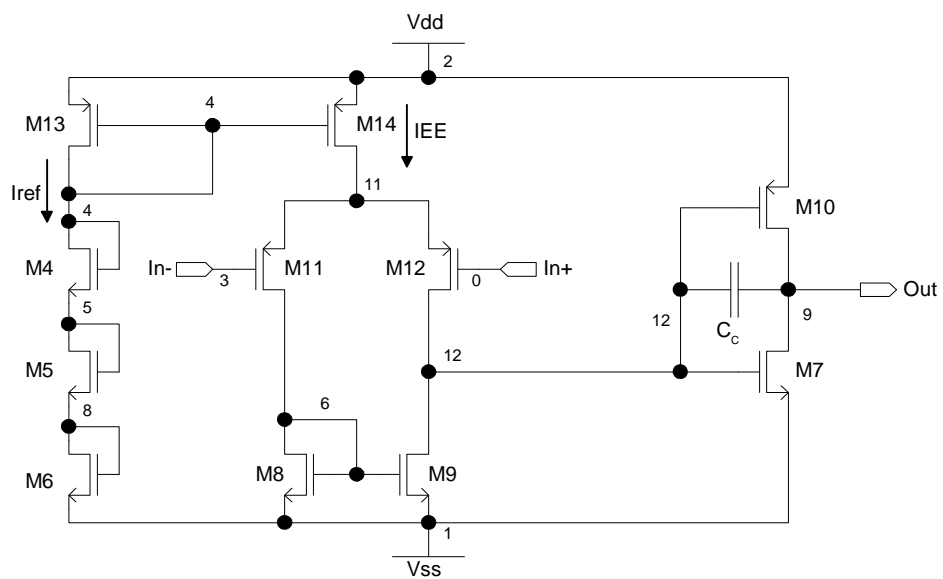
The MMGS comprises two parts: the AME and AMP. The former generates models based on the model generation algorithm, the latter uses these models to predict signals. It uses the same approach as the SISO models to generate multiple models, that is, by observing the variation in output error against input range. Moreover, the criteria in Eq.

5-4 are employed to decide when a new model is added. The difference is that the structure of RARMAX system is modified seen in Eq. 6-1, i.e., a second input vector  $v$  is added compared with Eq. 5-3.

$$y(t) + a_1 y(t-1) + \dots + a_{na} y(t-na) = b_1 u(t-1) + \dots + b_{nb} u(t-nb) + f_1 v(t-1) + \dots + f_{nf} v(t-nf) + e(t) + c_1 e(t-1) + \dots + d \cdot v_{offset} \quad \text{Eq. 6-1}$$

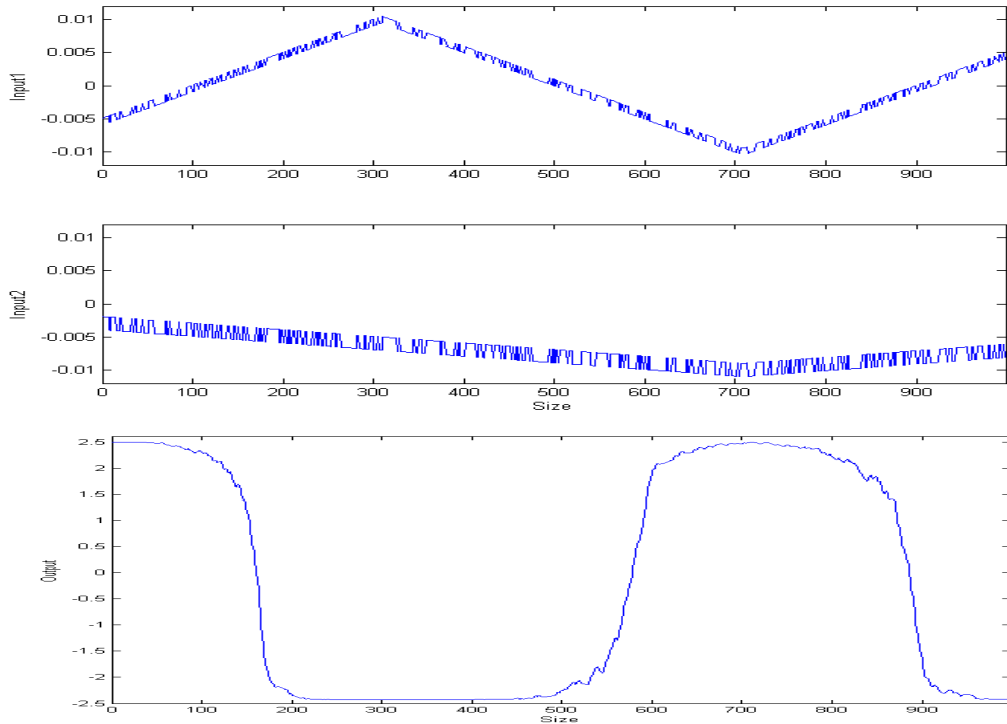
where  $u(t)$ ,  $v(t)$  represent first and second inputs, respectively;  $y(t)$  is the output signal;  $e(t)$  is the noise parameter or prediction error;  $v_{offset}$  is the offset vector;  $a$ ,  $b$ ,  $c$ ,  $d$  and  $f$  are their coefficients.

The same two-stage CMOS operational amplifier (op amp) as Figure 4-1 is employed also shown in Figure 6-1 as an open-loop amplifier in these experiments.



**Figure 6-1:** Schematic of the two-stage CMOS operational amplifier

In addition, the pseudorandom binary sequence generator (PRBSG) developed in chapter 5 is used to generate two PRBS signals for the MMGS as shown in Figure 6-2, in which 30,000 samples, only last 1,000 are displayed.



**Figure 6-2:** Two inputs and one output signals from TLS

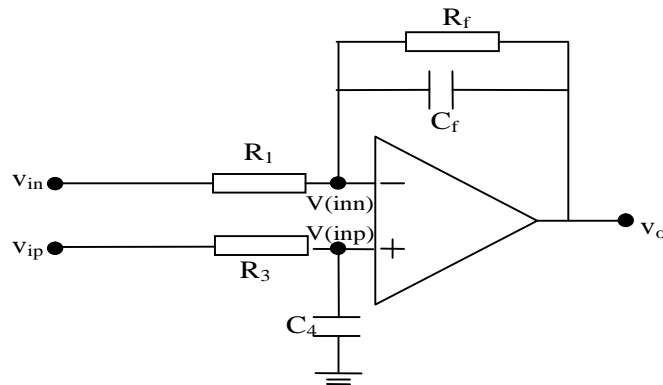
It is seen that two input signals have different shapes with approximately the same amplitude but different frequencies. The first input signal for the estimator is a 125Hz, 11mV triangle waveform with a 0.5mV pseudorandom binary sequence (PRBS) superimposed on it. The second input signal for the estimator is a 25Hz, 10mV triangle waveform with a 1mV pseudorandom binary sequence (PRBS) superimposed on it. The PRBS has a time interval of 10us. The first signal (higher frequency) is connected to the inverting input (In-), another PRBS (lower frequency) is connected to the non-inverting input (In+). The output signal saturates at  $\pm 2.5V$ .

### **6.3 Experimental Results**

#### **6.3.1 Analysis of MMGS**

The aim of the section is to investigate if the MISO MMGS system is able to model a linear system. A linear circuit as shown in Figure 6-3 is used. This circuit is able to simultaneously achieve small steady-state error, large phase margin, and large gain crossover frequency [Chirlian82]. This circuit is based on a differential amplifier, which is a type of an electronic amplifier that multiplies the difference between two inputs by the differential gain. It consists of two low-pass filters with the frequencies of 100Hz and

10Hz, respectively. The transfer function is given in Eq. 6-2, where  $R_I = 1\text{k}\Omega$ ,  $R_f = 10\text{k}\Omega$ ,  $C_f = 0.15915\mu\text{F}$ ,  $R_3 = 10\text{k}\Omega$  and  $C_4 = 1.5915\mu\text{F}$ .



**Figure 6-3:** A linear circuit with two low-pass filters

$$v_o = -\frac{R_f \cdot R_3 \cdot C_4 \cdot s + R_f}{R_1 \cdot C_f \cdot R_f \cdot R_3 \cdot C_4 \cdot s^2 + (R_1 \cdot C_f \cdot R_f + R_1 \cdot R_3 \cdot C_4) \cdot s + R_1} \cdot v_{in} + \frac{R_1 \cdot C_f \cdot R_f \cdot s + R_1 + R_f}{R_1 \cdot C_f \cdot R_f \cdot R_3 \cdot C_4 \cdot s^2 + (R_1 \cdot C_f \cdot R_f + R_1 \cdot R_3 \cdot C_4) \cdot s + R_1} \cdot v_{ip} \quad \text{Eq. 6-2}$$

Its transfer function under discrete-time is shown in Figure 6-4 produced in the system identification toolbox in MATLAB.

```
>> Discrete-time IDPOLY model: y(t)=[B(q)/F(q)]u(t)+ e(t)
B1(q) = -0.8421 q^-1 + 0.8347 q^-2
B2(q) = 0.0125 q^-1 - 0.004393 q^-2
F1(q) = 1 - 1.907 q^-1 + 0.9078 q^-2
F2(q) = 1 - 1.907 q^-1 + 0.9078 q^-2

This model was not estimated from data.
Sampling interval: 0.00014
```

**Figure 6-4:** The coefficients from the system identification toolbox

The investigation uses two steps:

- 1) The analytical simulation is used for analyzing the circuit, both input and output data are then stored in a text file.
- 2) The MMGS generates a model based on these data.

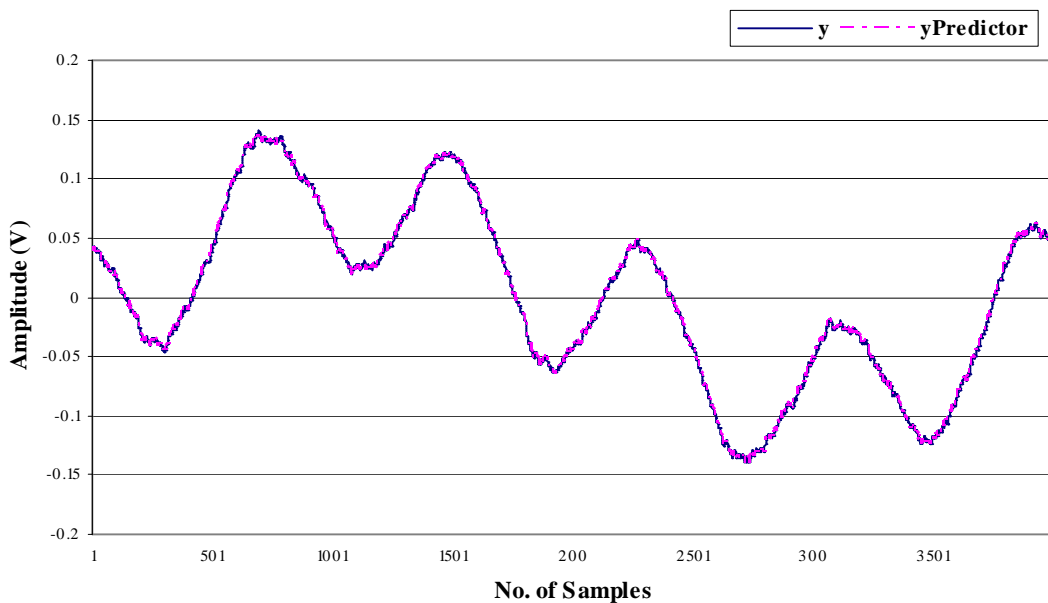
The circuits analysed can be found Appendix I-1 and Appendix I-2, respectively.

The same training data shown in Figure 6-2 is used again, in which the higher frequency signal is connected to the inverting input  $v_{in}$ , and the other one is applied to the non-inverting input  $v_{ip}$ . All (i.e., two inputs and output) signals are then passed to the MMGS to generate a model. Its transfer function is shown in Figure 6-5. Comparing this model with the one from the circuit in Figure 6-4, their coefficients are seen to match.

$$V_o = \frac{- (0.8421 q^{-1} - 0.8347 q^{-2})}{1 - 1.907 q^{-1} + 0.9078 q^{-2}} V_{in} + \frac{(0.0125 q^{-1} - 0.0044 q^{-2})}{1 - 1.907 q^{-1} + 0.9078 q^{-2}} V_{ip}$$

**Figure 6-5:** The model under discrete-time from MMGS

The MMGS was then used to predict the behaviour of the same circuit. Results are shown in Figure 6-6, only the last 4,000 samples are plotted, where the  $x$  axis is the number of samples, and the  $y$  axis shows the amplitude voltage (V).



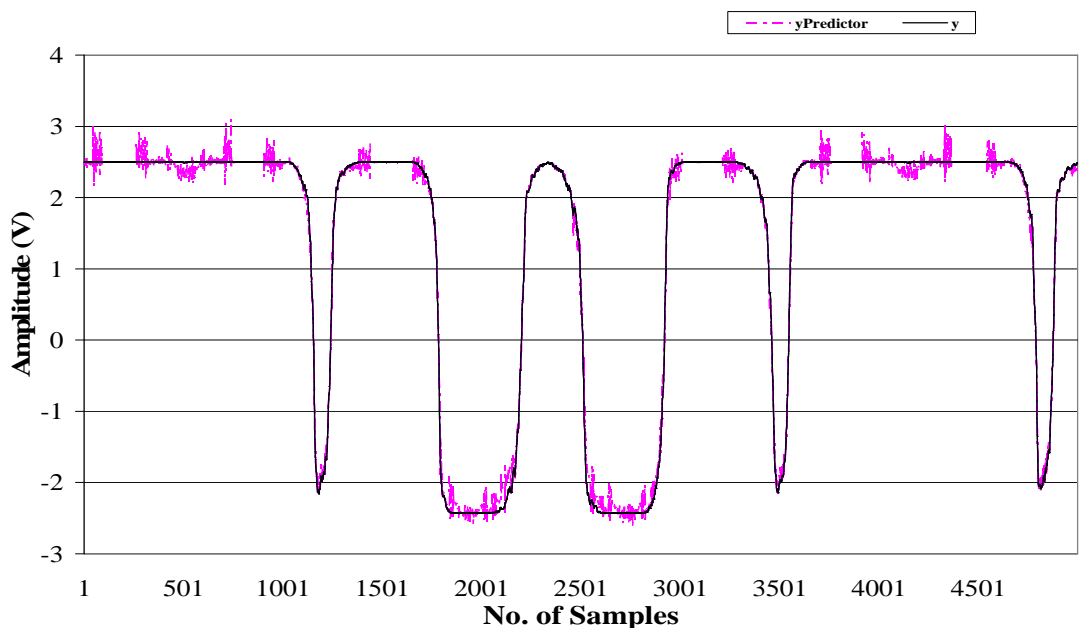
**Figure 6-6:** The predicted signal from the analytical system

The difference between the predicted signal  $y_P$  and the original one  $y$  is measured using the average difference measurement, which can be found in Eq. 5-2. In this case *Average\_dif* is 0.000143%.

Further analysis of various low-pass filters with different cut-off frequency showed that reasonable results are achieved using the procedure detailed above. Unfortunately, this form of MMGS is unable to model high-pass filters accurately due to aliasing. An anti-aliasing filter can be employed to make sure there are no signals beyond the Nyquist sampling frequency. However, this may cause too much phase shift or other discrepancies.

### 6.3.2 Simulation for Nonlinearity

The aim of the section is to investigate whether the MMGS is able to model strong nonlinear behaviour. In this subsection the training data is obtained from the transistor-level open-loop op amp SPICE model from Figure 6-1 is simulated with inclusion of strong nonlinearity. Three stable models are used by the MMGS, and the stimuli are the same as in Figure 6-2, for both the AME and AMP. The signals are shown in Figure 6-7, only last 5,000 samples are displayed, where the  $x$  axis is the number of samples,  $y$  axis is the amplitude (V).



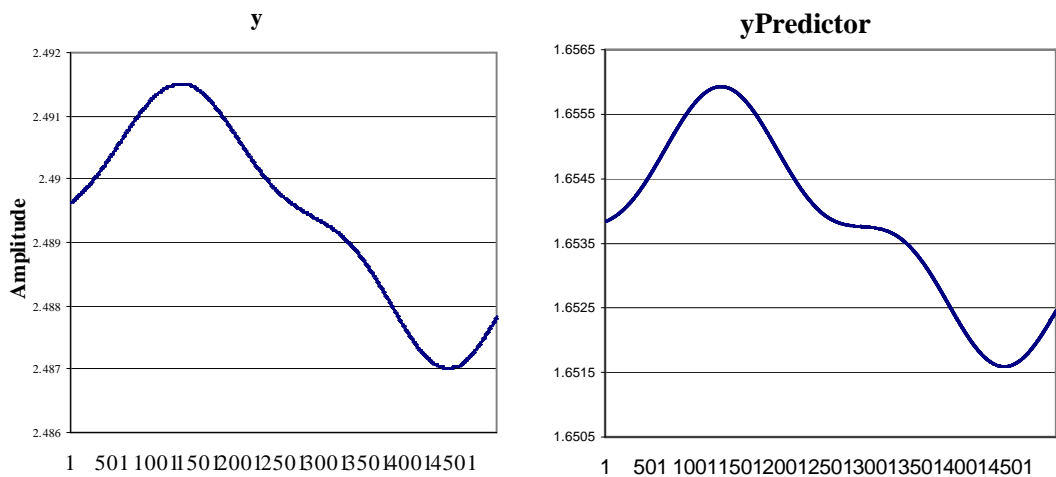
**Figure 6-7:** The predicted signal from MMGS



It is seen that the predicted signal  $y_{Predictor}$  is able to match the original signal  $y$ . The percentage of average difference between the two signals is 0.1593%. There is some noise in the saturation regions because the estimator struggles to obtain enough information where there is no relationship between input and output, that is, the output is not dependant on input. Here it has to generate excitation itself to make the best guess. This can be improved by using a saturation detector to delete the samples in the saturation region, so the estimator will focus on available information. These results indicate that the MMGS is able to handle nonlinearity with good accuracy.

### 6.3.3 Validation for Test for MMGS Generated Models via Time-domain (Transient) Simulations

Various stimuli including sine and square waveforms were used in order to validate the models in subsection 6.3.2. However, only sine waveform is shown in this section. The two inputs are 0.6mV at 500Hz, and 0.3mV at 100Hz. Signals are plotted in Figure 6-8. It is seen that the predicted signal  $y_{Predictor}$  is close to the original  $y$  in term of the amplitude, but the offset is not as expected. The result may be improved by adding an offset parameter, which will be discussed in next chapter.



**Figure 6-8:** The predicted signal with multiple models generated from MMGS

## 6.4 Conclusion

In this chapter the multiple model generation system (MMGS) is developed for MISO models from transistor level SPICE simulations. It has been shown that these generated

models are able to model nonlinear behaviours, handle low-pass filters accurately, and predict various circuit responses reasonably, except for their offset. Offset can be handled by adding a parameter to the model, and problems concerning automation of determining offset will be improved by implementing high level modelling (HLM) in chapter 7.

# ***Chapter 7: High Level Modelling based on Models from the MMGS***

## ***7.1 Introduction***

As was discussed in chapter 2 high level modelling (HLM) is an important part of modern design flows.

In this chapter HLM based on models generated from the multiple model generation system (MMGS) is implemented using both manual and automatic approaches. The following sections are outlined: section 7.2 introduces how to implement this conversion manually. The automatic algorithm is discussed in section 7.3 followed by the result in section 7.4. Section 7.5 supplies the conclusion.

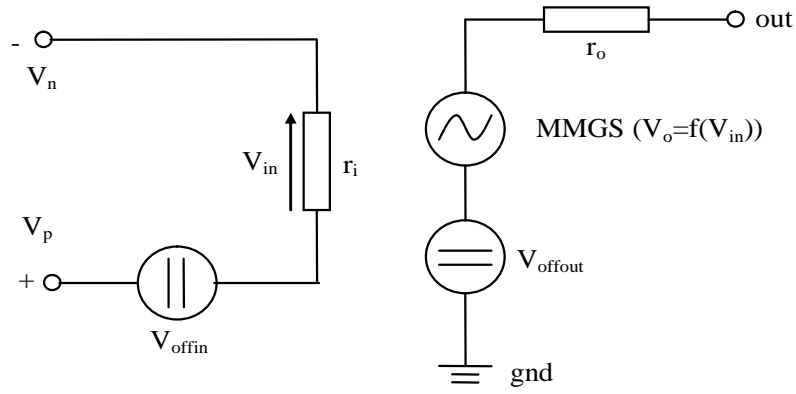
## ***7.2 Manual Conversion***

The aim of this section is to introduce the structure of the behavioural model and prove that there is no discontinuity caused during model switching.

### ***7.2.1 Structure of the Behavioural Model***

The structure of the behavioural model used in this work is shown in Figure 7-1, which is similar to the linear model in chapter 3. However, this behavioural model takes nonlinearity into account, that is, multiple models from the MMGS are included in the voltage controlled voltage source (VCVS) (implementing  $V_o = f(V_{in})$ ) to handle nonlinearity. In this case the same models from the multiple model generation system (MMGS) are used in chapter 5, and the input voltage range is  $\pm 0.01$  V.

Two linear resistors  $r_i$  and  $r_o$  provide the input impedance and output impedance, respectively, and  $v_{offin}$  and  $v_{offout}$  are parameters for modelling the input offset and output offset, respectively.



**Figure 7-1:** Structure of the behavioural op amp model

Each model in the MMGS can be expressed as a multiple-input single-output (MISO) system. The high level model is similar to this, but without the noise vector as shown in Eq. 7-1, where  $u(t)$ ,  $v(t)$  represent the first and second inputs, respectively;  $y(t)$  is the output signal;  $v_{offset}$  is the offset vector;  $a$ ,  $b$ ,  $d$  and  $f$  are their coefficients.

$$y(t) + a_1 y(t-1) + \dots + a_{na} y(t-na) = b_1 u(t-1) + \dots + b_{nb} u(t-nb) + f_1 v(t-1) + \dots + f_{nf} v(t-nf) + d \cdot v_{offset} \quad \text{Eq. 7-1}$$

The equation above is then modified using the  $z$  transform as shown in Eq. 7-2 in order to facilitate implementation in the VHDL-AMS language.

$$v_{out} = v_n \cdot (b_1 z^{-1} + b_2 z^{-2} + \dots + b_{nb} z^{-nb}) + v_p \cdot (f_1 z^{-1} + f_2 z^{-2} + \dots + f_{nf} z^{-nf}) - v_{out} \cdot (a_1 z^{-1} + a_2 z^{-2} + \dots + a_{na} z^{-na}) + d \cdot z^{-1} \quad \text{Eq. 7-2}$$

where  $v_n$ ,  $v_p$  and  $v_{out}$  represent inputs and output signals, respectively;  $z^{-1}$ ,  $z^{-2}$ ... are the first and second delay operations, and so on;  $b_1 \dots b_{nb}$  and  $f_1 \dots f_{nf}$  are coefficients of the inputs,  $a_0$ ,  $a_1 \dots a_{na}$  are coefficients for the outputs, and  $d$  is the offset coefficient.

Each behavioural model written in VHDL-AMS is at the form shown in Eq. 7-3.

$$\begin{aligned}
vin\_zm1 &= v_n' \text{delayed}(T); \\
vin\_zm2 &= vin\_zm1' \text{delayed}(T); \\
&\vdots \\
vip\_zm1 &= v_p' \text{delayed}(T); \\
vip\_zm2 &= vip\_zm1' \text{delayed}(T); \\
&\vdots \\
vout\_zm1 &= vout' \text{delayed}(T); \\
vout\_zm2 &= vout\_zm1' \text{delayed}(T); \\
&\vdots \\
vout &= b_1 \cdot vin\_zm1 + b_2 \cdot vin\_zm2 + \dots + f_1 \cdot vip\_zm1 + f_2 \cdot vip\_zm2 + \dots \\
&\quad + d_1 - a_1 vout\_zm1 - a_2 vout\_zm2 - \dots;
\end{aligned}
\tag{Eq. 7-3}$$

where the attribute ‘*delayed*’ is employed to perform a  $z$  domain delay by the sampling interval  $T$ ; voltage sources  $v_n$ ,  $v_p$  and  $v_{out}$  are regarded as quantities at discrete-time in VHDL-AMS using the zero-order hold attribute (‘*zoh*’) that acts as a sample-hold function to allow for the periodic sampling of the quantity, in which the value is held between samples.

In VHDL-AMS there are three types of quantity: free, branch and source quantities. A free quantity is an analogue-valued object that can be used in signal-flow modelling; a branch quantity is similar, but is specifically used to model conservative energy systems; and a source quantity is used for frequency and noise modelling [Ashenden03].

Free quantities  $vin\_zm1$ ,  $vin\_zm2$ ,  $vip\_zm1, \dots$ ,  $vout\_zm2$  represent first and second delays of the sampling interval  $T_s$ .  $T_s$  is derived from the sampling frequency, i.e.,

$$f_s = \frac{1}{T_s},$$

which has to be at least two times higher than the input signal frequency (using

Nyquist’s law) in order to extract all of the information from the bandwidth [Bissell94]. Failure to do so can cause aliasing, which produces unwanted frequencies within the range expected [Bissell94]. These unwanted frequency components occur at the difference between the input and sampling frequencies, and produce erroneous sampled waveforms. To overcome aliasing, anti-aliasing filters can be employed by sampling systems.

As previous discussed the model selection process during the MMGS is based on input ranges shown in Figure 7-2.

```

If the input signal is within the range for the first model use
    The first model is selected
Else if the input signal is within the range for the second model use
    The second model is selected
    .
    .
    .

Else the input signal is not included in these ranges
    Either the first or the last model is selected

```

**Figure 7-2:** Model selection algorithm based on input range

However, during high level modelling (HLM) we decided to focus on the model selection process based on the output values instead of inputs in order to achieve bumpless transfer, but input information is fed to all the models all the time. During model switching, two models either side of a boundary (interface) giving the same input will approach each other, because they have been trained by the estimator to be consistent at the interface point.

This process is shown in Figure 7-3 using an *if-else* statement. Initially we displayed all transfer functions of models in HLM, we then run all the models in parallel, meanwhile input information are fed to all the models all the time. We only switch from one output to another when they reach the particular area of the input signal.

```

Display the transfer function for the first model
Display the transfer function for the second model
    .
    .
    .
Display the transfer function for the last model

If the input signal is within the range for the first model use
    The output voltage of the first model is selected
Else if the input signal is within the range for the second model use
    The output voltage of the second model is selected
    .
    .
    .

Else the input signal is not included in these ranges
    Either the output voltage of the first or the last model is selected

```

**Figure 7-3:** The model selection process for the predictor

By doing this way we can achieve reasonable bumpless transfer. Although it is not guaranteed that complete bumpless transfer is achieved, discontinuities will be small because the two models are estimated to have the same value for the input because they have both been subjected to the same boundary zone in the training data. However, higher simulation speed may not be achieved. This can be improved by feeding only the neighbouring models instead of all models.

This behavioural model is used to implement various systems such as a differential amplifier and a low-pass filter in order to verify that it works well. Simulations were run using two simulators: SMASH from Dolphin [Dolphin] and SystemVision from Mentor Graphics [Mentor], respectively. The difference between them has been discussed in Appendix K.

### **7.2.2 Investigation to Bumpless Transfer using SMASH Simulator**

The aim of the subsection is to prove that bumpless transfer between models can be achieved using an example. This is realised in four steps:

1. TLS in HSPIICE is performed. The circuit used is an open-loop amplifier based on the op amp in Figure 4-1. The stimulus containing 20,000 samples is connected to the inverting input port (In-). It is a 82.5Hz, 0.5mV triangle waveform with a 0.1mV, 100kHz pseudorandom binary sequence (PRBS) superimposed on it. The MMGS generates three models by estimating these data from TLS. These models correspond to input voltage thresholds [-0.6mV -0.4909mV -0.3818mV 0.6mV], respectively.
2. The AMP in the MMGS uses the first model in step 1 to predict the output signal. A sine waveform with the amplitude of 0.6mV at 1MHz is used as the stimulus connected to the inverting-port of the open-loop amplifier.
3. The same process in step 2 is implemented for the VHDL-AMS model using the SMASH simulator [SMASHR05] [SMASHU05].
4. Results from step 2 and step 3 are compared.

In step 1 a data loading process is run to load data from the pseudorandom binary sequence generator (PRBSG) as a stimulus for estimation. This is implemented by a subprogram *function* written in VHDL-AMS shown in Figure 7-4.

The file is open in *read\_mode*, and then data is read line by line with a procedure *readline*, which creates a string object in the host computer's memory and returns a pointer to the string. The *read* procedure is then used to read this string into an array. A warning message is given if the size of the returned data is different from the data in the file.

```

-- define a function for data loading
impure function read_array(file_name:string;array_length:natural)
  return real_vector is
    --type real_file is file of real;
    file vsoce:text;

    variable result:real_vector(0 to array_length-1);
    variable iline : Line;
    variable index:natural;
begin
  --load all the data from the file
  index:=0;          --initialization

  --open the file for reading
  file_open(vsoce, file_name, READ_MODE);

  while not endfile(vsoce) and index <= array_length loop
    readline(vsoce, iline);
    read(iline,result(index));

    index:=index+1;
    if array_length>result'length then
      report"the store is not large enough!" severity warning;
    end if;
  end loop;
  return result;
end function read_array;

```

**Figure 7-4:** Data writing based on the sampling interval

A real vector is then used to store these data. After that the data are assigned to a *quantity* as the input signal. This is translated into VHDL-AMS in Figure 7-5:

```

real_convertor: process is
begin
  for i in solutions'range loop
    sigvalue <= solutions(i);
    wait for 1.0us;
  end loop;
end process real_convertor;

sig_break:process (sigvalue) is
begin
  break;
end process sig_break;

vinsource==sigvalue'ramp;

```

**Figure 7-5:** Assigning data as stimuli



where *'range* is an attribute used to determine the range of the *real\_vector*; a *wait for* statement is necessary to allow time to elapse between applying new signals, otherwise the new signal may overwrite the previous one, the analogue solver will restart every 1.0us and wait for the new data load. Use of the *'ramp* attribute without any parameters allows the *quantity* to follow the signal exactly [Ashenden03]. Furthermore, a *break* statement can be used immediately after the signal assignment process to restart the analogue solver effectively.

As has been discussed, models from the MMGS are developed under discrete-time, so in step 3 the sine waveform has to be sampled using the sampling interval (0.01us) before it can be used for prediction. The discrete-time data is obtained by using some functions written in VHDL-AMS and shown in Figure 7-6.

```

vin_sampled==v_in'zoh(Tsmp);
-- Process to generate sample clock
sample_tick: process (tick) is
begin
    tick <= not tick after Tsmp * 0.02 Sec;
end process sample_tick;

-- Process to sample vin and write to the log file
sample_vin: process (tick) is
variable iline:Line;
variable open_status : file_open_status;
variable index : natural := 1;
begin
    if not file_is_open then
        file_open(open_status, vsoce, filename, WRITE_MODE);
        file_is_open <= true;
    end if;
    if open_status /= open_ok then
        report file_open_status'image(open_status) & " while opening file "
        severity warning;
        file_is_open <= false;
    end if;
    --report "sampled vin data point is " & real'image(vin_sampled) severity note;
    if index <= index_max then
        write (iline,vin_sampled);
        writeline(vsoce, iline);

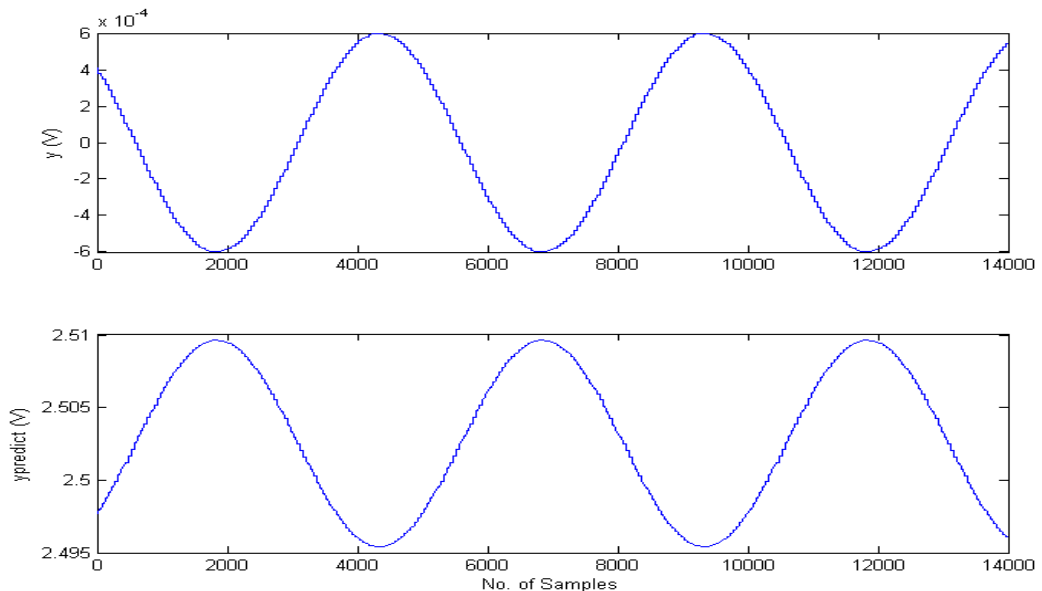
        index := index + 1;
        if index > index_max then
            report "index can not be over the maximum size!"
            severity error;
            file_close(vsoce);      --close the file
        end if;
    end if;
end process sample_vin;

```

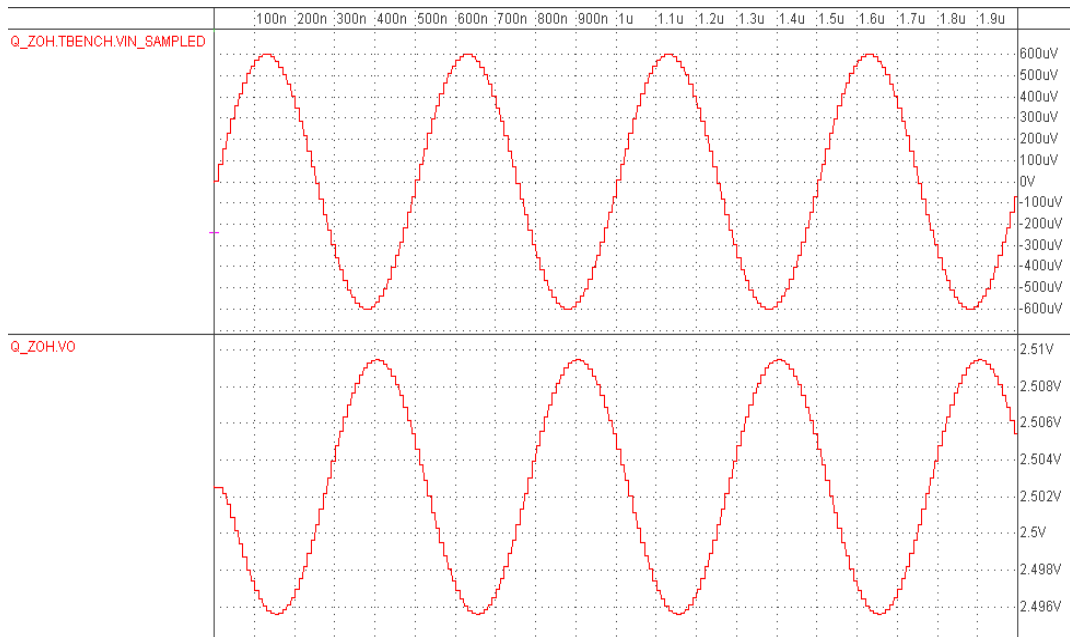
**Figure 7-6:** Data writing based on the sampling interval

The zero-order hold attribute ('*zoh*') samples the continuous quantity  $v_{in}$  at discrete points in time  $T_{smp}$  (0.01us). The first *process* statement with the signal *tick* monitors when  $T_{smp}$  changes. The second *process* statement is to implement data writing. In this statement initially a file open check is employed to determine if the file can be opened, the parameter *status* is used to return information about the success or failure of the operation. This parameter is predefined in the package standard as *file\_open\_status*, others include *open\_ok*, *status\_error* and so on [Ashenden03]. If the file is open successfully, the value *open\_ok* is returned, and further processes can be run. After the file is opened the procedures *write* and *writeline* are used for data writing. A *file\_close* operation is provided paired with the *file\_open* operation, so that we do not inadvertently write the wrong data to the wrong file.

The results from step 2 and 3 are compared using all model in [-0.6mV 0.6mV] to prove model switches without bumps, signals from the AMP and VHDL-AMS model are illustrated in Figure 7-7 and Figure 7-8, respectively. It has been mentioned previously that the voltage thresholds are set at [-0.6mV -0.4909mV -0.3818mV 0.6mV]. It is seen that the output signals (bottom) from both figures can be matched. Therefore, the model conversion for the model is successful.



**Figure 7-7:** Predicted signal from the AMP based on all model



**Figure 7-8:** High level modelling based on all model

It is seen that amplitudes and shapes of output signals (bottom) can be matched in terms of shape and amplitude. There is not any discontinuity on the outputs when the system is operating. Therefore, bumpless transfer can be achieved during model switching by using the procedure mentioned in Figure 7-3.

### 7.3 The Multiple Model Conversion System (MMCS)

In this section an automatic multiple model conversion system (MMCS) is developed to convert models from the MMGS into VHDL-AMS models once these models have been validated. The MMCS is written in MATLAB and located after the MMGS. The structure of the VHDL-AMS model is based on Figure 7-1.

This system is defined as a function in MATLAB shown in Eq. 7-4:

$$\text{function MMCS}(thm, threshold, nn, filename) \quad \text{Eq. 7-4}$$

where *MMCS* is the function name, *thm*, *threshold* and *nn* are parameters that are obtained from the AME, *filename* is used to create a HDL file with *.vhd* extension for VHDL-AMS models, the directory is required so that the user is able to decide where

the file can be created. This function does not return anything because only the file saved in the directory of *filename* is required.

Initially the size of *thm*, *threshold* and *nn* are defined in order to obtain the number of models, the ranges for these models and the orders of each vector of the model seen Eq. 7-2, respectively. *thm* contains a two-dimension array. Its rows and columns contain the models and the parameters, respectively. The system dynamically loads coefficients in *thm* for each model seen Eq. 7-2 and its corresponding *threshold*.

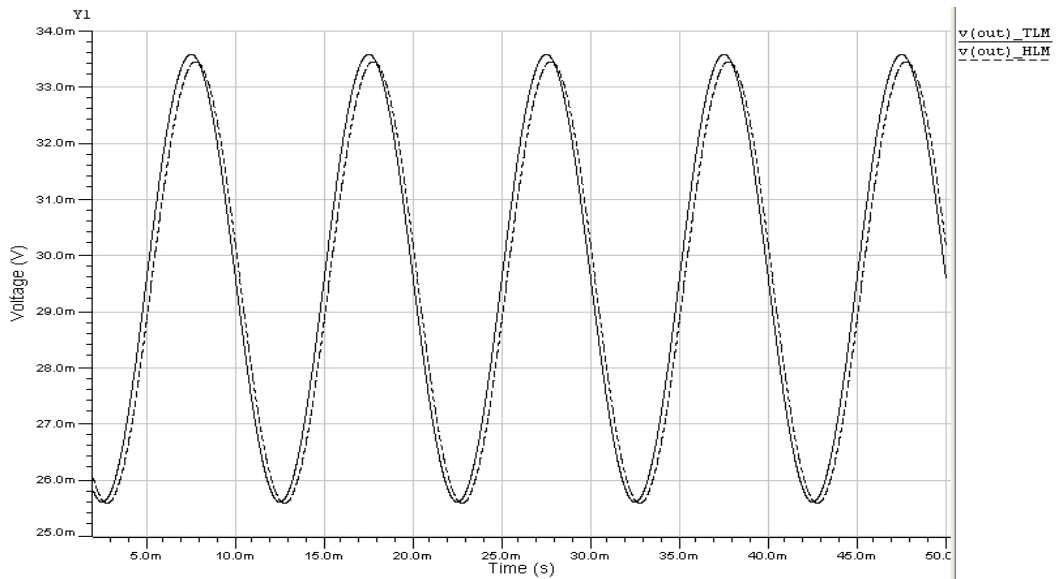
The general structure of a VHDL-AMS model consists of three parts: *entity*, *architecture* and *testbench* [Ashenden03], which are obtained by employing multiple *fprintf* statement in MATLAB. Other statements such as *fopen* are also used to open a file and then write the key words in the file. It is noted that the file directory needs to be set up before the creation is implemented because *fopen* does not create a directory, but accesses it and generates new files. At the end of the system a *fclose* statement is used to close the file.

## **7.4 Experimental Results**

In this section the transistor level simulation (TLS) and high level modelling (HLM) based on the models from the MMGS and MMCS are compared. The aim is to observe whether our models are able to achieve higher speed and reasonable accuracy. The models used in this case have been generated in section 6.3.2. Both TFS and HLM are run in SystemVision [SystemVision], facilitating comparison.

### **7.4.1 The Inverting Amplifier**

An inverting amplifier with a gain of -4 is used for simulation. The stimulus is a sine waveform with the amplitude of 1mV and the frequency of 100Hz. The transient analysis is performed using  $t(start) = 2ms$ ,  $t(end) = 50ms$  with a step of 0.001ms. The output signals from the transistor level  $V(out)_{TLM}$  and high level modelling  $V(out)_{HLM}$  are shown in Figure 7-9.



**Figure 7-9:** Signals between the transistor level and the high level modelling

It is seen that both signals can be matched in terms of shape and amplitude.

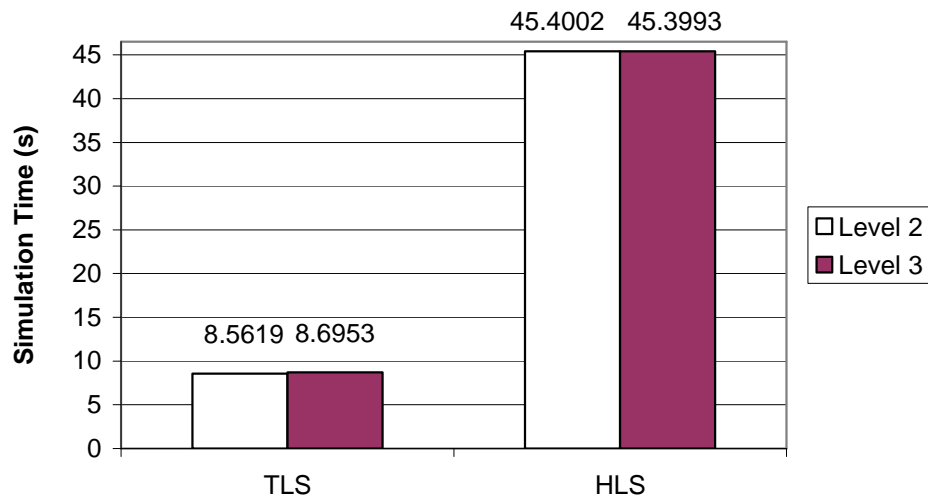
Comparing the simulation time under the same condition, the TLS takes 8.53s of CPU time, the HLS needs about 45.135s, which is about 5 times slower than the TLS. This is because the simulator has not been optimised to use this kind of approach, and so the computational overhead is high.

To investigate simulation speed, two types of experiments are implemented using the same inverting amplifier: 1) during HLM we replace multiple models by a single model to observe simulation time. 2) we consider the SPICE level of the transistor model and prove that the level of transistor model can affect the simulation results.

Firstly we run HLS using a single model instead of multiple models under the same circumstance, the simulation time is 40.234s, which is faster than HLS using multiple models (45.135s). This is because the simulator has not been optimised to use this kind of approach, and so the computational overhead is high.

Secondly we use different level transistor models in the same netlist of the op amp in Figure 4-1. It is known that the transistor that has been used through the thesis is at level 2. In this experiment we use a new transistor, which is a 1.2 micron CMOS model

(Level 3 instead of level 2) [Spiegel95]. Each type of simulation has been run 10 times, the average value is then chosen in order to reduce affects of interaction in computer. The comparison is shown in Figure 7-10.



**Figure 7-10:** Simulation Speed Comparison between level 2 transistors and level 3 transistors

It is seen that during TLS simulation speed based on the level 3 transistor (8.6953s) becomes slower than the one using the level 2 transistor (8.5619s). This is because the former is more complex than the level 2 transistor, it takes more time to complete simulation.

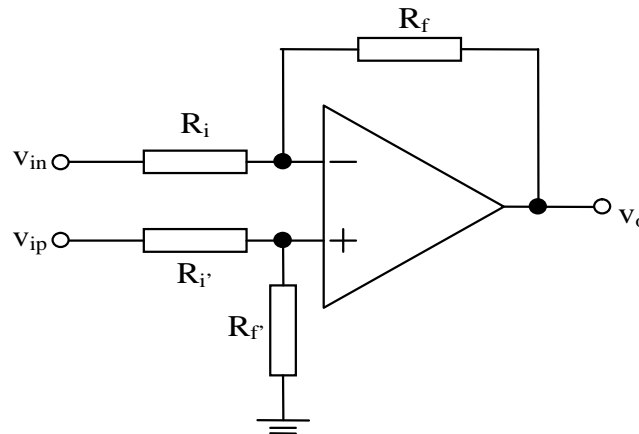
Although the difference between TLS time is not very significant, it has indicated that as the transistor is getting more complex (the parameter level in the transistor is higher), TLS becomes slower, whereas simulation time from HLS almost does not change. Therefore, simulation speed-up during HLS may be observed.

In the future work, we will use more complex CMOS transistors (e.g., IBM 0.13 micron level 49 [Mosis]) to investigate the improvement of simulation speed.

### 7.4.2 The Differential Amplifier

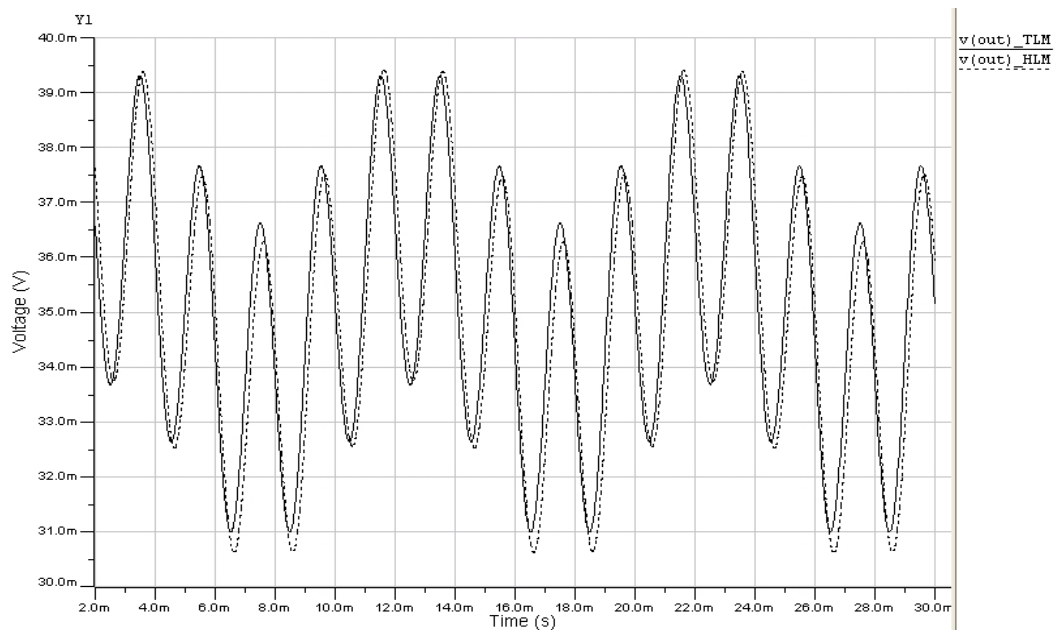
Initially we consider a differential amplifier. The differential amplifier is useful for handling signals referred not to the circuit common, but to other signals, known as floating signal sources. Its capability of rejecting common signals makes it particularly

valuable for amplifying small voltage differences contaminated with the same amount of noise. Its typical circuit is shown in Figure 7-11.  $R_i = 2k\Omega$ ,  $R_f = 10k\Omega$ ,  $R_i' = 1k\Omega$ ,  $R_f' = 5k\Omega$ . The stimuli are two sine waveforms: one has the amplitude of 0.6mV at 500Hz for  $v_{in}$ , another has the amplitude of 0.3mV at 100Hz for  $v_{ip}$ . Transient analysis is performed using  $t(start) = 20ms$ ,  $t(end) = 60ms$ , and the time step is 0.001ms.



**Figure 7-11:** The differential amplifier

The output signals from the transistor level and high level are plotted in Figure 7-12.



**Figure 7-12:** Signals between the transistor level and the high level modelling

Compared with the transistor level simulation (TLS), the shapes and amplitudes of the signals from the model are very close.

Under the same conditions TLS takes 14.72s of CPU time, and the behavioural model takes about 80.093s. To investigate why HLS is slower than TLS the behaviour model using multiple models is replaced by a single model. Under the same circumstance with only one model, HLS needs 70.234s, which is faster than HLM using multiple models. This is because the simulator has not been optimised to use this kind of approach, and so the computational overhead is high.

Moreover, the offset issue in section 5.5.1 has been improved by adding an offset parameter in the model.

## ***7.5 Conclusion***

In this chapter HLM is implemented using a behavioural model. The model is produced using the MMCS, which converts models generated by the MMGS into VHDL-AMS models. This MMCS can dynamically load parameters and thresholds for each model. The model switching process has been validated using a manual experiment in the SMASH simulator.

Results show that the behavioural model can model various systems including an inverting amplifier and a differential amplifier. Speed-up is not achieved because the simulator is not optimised to deal with a lot of computational overhead present when the high level model structure is used. Moreover, speed-up can be improved by feeding the neighbouring models instead of all models during model selection process.

In the next chapter a similar system to the MMGS that generates continuous-time models will be implemented in order to address the speed-up issue.



# ***Chapter 8: Multiple Model Generation System using Delta Operator***

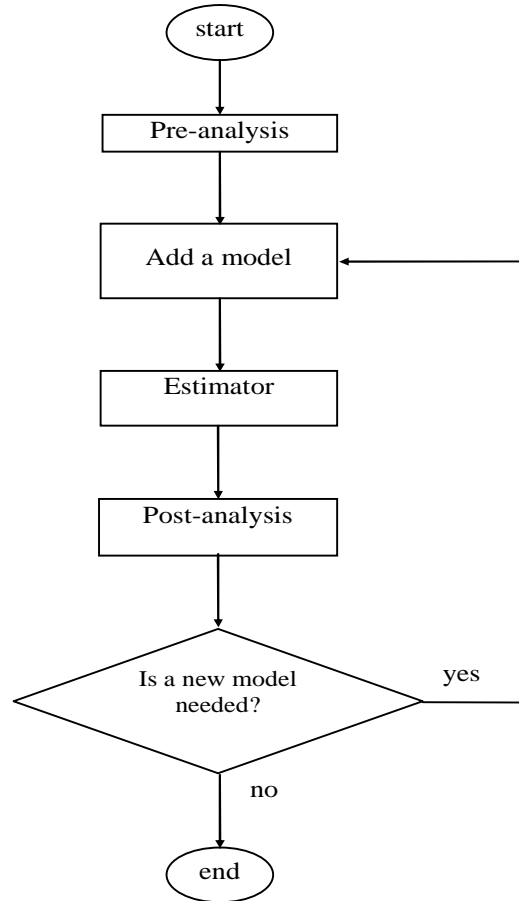
## ***8.1 Introduction***

The objective of the chapter is to obtain a novel behavioural model in order to perform high level fault modelling (HLFM) and high level fault-free modelling. The ideal model should have higher simulation speed with reasonable accuracy compared with transistor level simulation (TLS). We develop a system named multiple model generation system using delta operator (MMGSD) for generating either single-input single-output (SISO) or multiple-input single-output (MISO) macromodels, so that the simulation speed may be improved compared with the MMGS in chapter 5. The MMGSD employs a similar approach to the MMGS, i.e., this model generation process still detects nonlinearity through variations in output error. The difference is that the delta transform is employed instead of the discrete-time transform. By using the delta operator the coefficients produced relate to physical quantities as in the continuous-time domain model and are less susceptible to the choice of sampling interval, provided it is chosen appropriately [Wilkinson91].

This chapter is outlined as follows: section 8.2 overviews the MMGSD; illustrative results for verifying the system are given in section 8.3; section 8.4 supplies the conclusion.

## ***8.2 Overview of MMGSD***

Similar to the MMGS the MMGSD includes an automated model estimator (AME) and an automated model predictor (AMP). The former implements the model generation algorithm. The AMP is used to implement these generated models. The AME includes three stages as illustrated in Figure 8-1: pre-analysis, estimator and post-analysis. Pre-analysis is mainly to set up conditions such as input range measurement and the number of intervals for model location. In the whole algorithm, this stage is only run once. Post-analysis is the critical step because procedures for creating models are run here.



**Figure 8-1:** The algorithm for the AME system

The estimator is based on modified recursively maximum likelihood (RML) estimation [Middleton90], more details can be found in Appendix L.2. The model structure is related to the Laplace transfer function of a process as follows. Initially a continuous time transfer function is considered, as shown in Eq. 8-1.

$$G(s) = \frac{b_0 s^n + b_1 s^{n-1} + \dots + b_n s^0}{s^m + a_1 s^{m-1} + \dots + a_m s^0} \quad \text{Eq. 8-1}$$

As it has been mentioned in the introduction when the sampling interval is sufficiently short, the continuous time transfer function  $G(s)$  is equal to the delta transfer function  $G(\delta)$  [Middleton90] shown in Eq. 8-2.

$$G(\delta) = \frac{y(t)}{u(t)} = \frac{b_0 \delta^n + b_1 \delta^{n-1} + \dots + b_n \delta^0}{\delta^m + a_1 \delta^{m-1} + \dots + a_m \delta^0} \quad \text{Eq. 8-2}$$

After rearranging this equation, Eq. 8-3 is obtained:

$$\delta^m y(t) = -(a_1 \delta^{m-1} + \dots + a_m) y(t) + (b_0 \delta^n + \dots + b_n) u(t) \quad \text{Eq. 8-3}$$

It is known that error is related to the quality of estimation, i.e., a smaller error indicates that a better estimated signal has been achieved. Thus the variation in output error against the input amplitude is analyzed in the MMGSD to decide if a new model needs to be generated. As with the MMGS, residual error is used for observing model errors. The difference is that deltarised error *depsilon* is observed instead of *epsilon*. *depsilon* has been discussed in relation to Eq. 2-16.

Initially the number of intervals to be used on the input voltage is set up. The decision to add a new model on one of the intervals is based on three equations shown in Eq. 8-4:

$$\begin{aligned} \text{mediumRange} &= (\text{highInterval} - \text{lowInterval}) / 2 \\ \text{criticalRange} &= (\text{highInterval} + \text{lowInterval}) / 2 \\ \text{criteria} &= [\text{mediumRange} - \text{mediumRange}(\text{smallestindex})] - \text{criticalRange} \end{aligned} \quad \text{Eq. 8-4}$$

where the medium range *mediumRange* is the half of difference between the maximum amplitude of error *maxInterval* and the minimum amplitude of error *minInterval* in the same interval; *criticalRange* is equivalent to the half of summation to *maxInterval* and *minInterval*; the variable *criteria* is the difference between the *mediumRange* and *criticalRange* at the same interval and then subtracts *mediumRange*; *smallestindex* is the index appointing to the interval where minimum range of *epsilon* is.

The difference from Eq. 5-4 in the MMGS is the reference index. In the MMGS the central index, based on the shape of the error (full value of errors), is used because it has the minimum magnitude of error. However, for the delta transform the central index may not be the index of the smallest error, and so the one relating to the smallest error value is selected.

A new model is required in an interval when *criteria* is greater or equal to zero, otherwise no action is taken. Only one model is created per iteration (Figure 8-1), which

is necessary because the shape of the error changes when a model is added. This process is complete when the number of models does not increase any more.

The same two-stage CMOS operational amplifier (op amp) shown in Figure 4-1 is used to illustrate our methodology. The training data used for the estimator is a 93.34Hz, 0.25V triangle waveform with a 0.04V, 10us PRBS superimposed on it. A similar signal but with lower amplitude and frequency is applied to the non-inverting input, because as it has been explained before the model generation process is only based on one input.

The model structure is for the MMGSD still based on the RARMAX system [Ljung99] but with modification because it is based on the discrete-time transform, whereas the MMGSD is based on delta transform. Therefore, during simulation (estimation) some quantities in the system need to be either deltarised or undeltarised, for example, *epsilon* in the AME and AMP is already deltarised, but during the vector update the undeltarised value is required. Therefore, we create two functions in the MMGSD: the *Deltarise* function and *Undeltarise* function. The former is to generate derivative vectors based on original vectors. The undeltarise function requires original data during the estimation. These two functions are used in different places in the MMGSD.

### 8.2.1 The Deltarise Function

The deltarise function is used to find the deltarised value using the delta operator given in Eq. 8-5, where delta ( $\delta$ ) is related to both the present and future values,  $T_s$  is the sampling rate,  $q$  is the forward shift operator used to describe discrete models, which is shown in Eq. 8-6.

$$\delta = \frac{q - 1}{T_s} \cong \frac{d}{dt} \quad \text{Eq. 8-5}$$

$$qx_k = x_{k+1} \quad \text{Eq. 8-6}$$

The equivalent form of Eq. 8-6 is obtained in Eq. 8-7, the relationship between  $\delta$  and  $q$  is a simple linear function, so  $\delta$  can offer the same flexibility in the modelling of discrete-time systems as  $q$  does.

$$\delta x_k = \frac{x_{k+1} - x_k}{T_s} = \frac{x(kT_s + T_s) - x(kT_s)}{T_s} \cong \frac{dx}{dt} \quad \text{Eq. 8-7}$$

The use of delta operator and its relationship is illustrated in the following example. It is a discrete-time model, but only output vectors are displayed in Eq. 8-8. Initially each vector is subtracted from the one next to it, as seen in Eq. 8-9, and is then divided by  $T_s$ , so deltarised value is obtained, as seen in Eq. 8-10. However, the last one highlighted by the rectangle is not involved in the calculation.

$$\begin{matrix} y(t) & y(t-1) & y(t-2) & \boxed{y(t-3)} \end{matrix} \quad \text{Eq. 8-8}$$

$$\begin{matrix} y(t-1) & y(t-2) & y(t-3) \end{matrix} \quad \text{Eq. 8-9}$$

$$\begin{matrix} \delta y(t-1) & \delta y(t-2) & \boxed{\delta y(t-3)} \end{matrix} \quad \text{Eq. 8-10}$$

$$\begin{matrix} \delta y(t-2) & \delta y(t-3) \end{matrix} \quad \text{Eq. 8-11}$$

To achieve  $\delta^2 y(t-3)$ , Eq. 8-10 is subtracted from Eq. 8-11, and then divided by  $T_s$ . The same procedure is used to obtain  $\delta^3 y(t-3)$ .

$$\begin{matrix} \delta^2 y(t-2) & \boxed{\delta^2 y(t-3)} \end{matrix} \quad \text{Eq. 8-12}$$

$$\delta^2 y(t-3) \quad \text{Eq. 8-13}$$

$$\delta^3 y(t-3) \quad \text{Eq. 8-14}$$

Thus, the deltarised version of Eq. 8-8 is obtained shown in Eq. 8-15.

$$\delta^3 y(t-3) \quad \delta^2 y(t-3) \quad \delta^1 y(t-3) \quad \delta^0 y(t-3) \quad \text{Eq. 8-15}$$

The same procedure is also used for other vectors such as the inputs vectors  $u$ ,  $e$  and the noise vector  $c$ . Delay is not included here. However, there is some difference such that in the input vector the current deltarised values ( $u(t)$ ,  $v(t)$ ) are not required. More details about the modification will be discussed in section 8.2.1.3.

## 8.2.2 The Undeltarise Function

This function is based on Eq. 7-5 but with the modification,  $q = \delta T_s + 1$ , in order to model at the current time. An example is also used to demonstrate how this reverse

algorithm works. It is a model in delta transform, but only the output vectors  $y$  are shown in Eq. 8-16. Firstly each vector, except for the last one, highlighted by the rectangle because it is already undeltarised, is multiplied by  $T_s$  in Eq. 8-17. We then add the output vectors as shown in Eq. 8-17 and Eq. 8-18, so undeltarised vectors are obtained in Eq. 8-19, i.e.,  $y(t-2)$  is obtained.

$$\delta^3 y(t-3) \quad \delta^2 y(t-3) \quad \delta^1 y(t-3) \quad \boxed{\delta^0 y(t-3)} \quad \text{Eq. 8-16}$$

$$T_s \delta^3 y(t-3) \quad T_s \delta^2 y(t-3) \quad T_s \delta^1 y(t-3) \quad \text{Eq. 8-17}$$

$$+ \quad + \quad +$$

$$\delta^2 y(t-3) \quad \delta^1 y(t-3) \quad \delta^0 y(t-3) \quad \text{Eq. 8-18}$$

$$\parallel \quad \parallel \quad \parallel$$

$$\delta^2 y(t-2) \quad \delta^1 y(t-2) \quad \boxed{y(t-2)} \quad \text{Eq. 8-19}$$

To achieve  $y(t-1)$ , Eq. 8-19 is multiplied by  $T_s$ , and then we add the vectors shown in Eq. 8-21

$$T_s \delta^2 y(t-2) \quad T_s \delta^1 y(t-2) \quad \text{Eq. 8-20}$$

$$+ \quad +$$

$$\delta^1 y(t-2) \quad \delta^0 y(t-2) \quad \text{Eq. 8-21}$$

$$\parallel \quad \parallel$$

$$\delta^1 y(t-1) \quad \boxed{y(t-1)} \quad \text{Eq. 8-22}$$

Finally  $y(t)$  is obtained using the same procedure as above.

$$T_s \delta^1 y(t-1) \quad \text{Eq. 8-23}$$

+

$$y(t-1) \quad \text{Eq. 8-24}$$

||

$$\boxed{y(t)} \quad \text{Eq. 8-25}$$

Therefore, the undeltarised version of Eq. 8-16 is achieved shown in Eq. 8-26.

$$y(t) \quad y(t-1) \quad y(t-2) \quad y(t-3) \quad \text{Eq. 8-26}$$

The number of iterations depends on a variable *numb*, the reason to use the variable is that during undeltarising, a vector such as output vector needs to be undeltarised once to obtain the value at next time, but during the prefilter update, it needs to be fully undeltarsied. If a full undeltarisation is required, *numb* is set to 0, otherwise an integer is selected. If the number is greater than the size of the vector array an error message is produced.

### 8.2.3 Two Functions Utility in MMGSD

It is known that the delta operator is a very high gain system because of the sampling interval  $T_s$  (10us in this case), so it is important not to put a vector or a variable in the wrong place during the manipulation, otherwise, the whole process may numerically explode very quickly.

In this subsection some key modifications in the MMGSD based on the functions defined above are described in section 8.2.3.1 and section 8.2.3.2, respectively.

#### 8.2.3.1 The AME

In order to obtain the deltarised output data  $dy$  at current time and the deltarised vector array  $dphi$ , the vector array  $phi$  ( $\varphi$ ) and the original output data  $y$  at current time are needed. The deltarise function is employed in Eq. 8-27.

$$dphi4y = deltarise([y phi(iia)], T_s) \quad \text{Eq. 8-27}$$

where *iia* indexes the array for the output vector in  $phi$ .  $T_s$  is the sampling interval,  $dphi4y$  is the deltarised vector array for output, in which the first element is  $dy$ , and all other elements are assigned to  $dphi(iia)$ .

Similarly input vectors  $u$  and  $e$ , and the noise vector  $c$  are deltarised values for  $dphi$ . However, their deltarised values at the current time are not required.

Secondly the prefilter  $ztil$  in RML is modified as seen in appendix L.2. It is already known that the relationship between  $psi$  ( $\psi$ ) and  $phi$  ( $\varphi$ ) in  $z$  transform is expressed as:  $phi(t) = c(z)*psi(t)$ , or  $phi(t) = psi(t)+c_1psi(t-1)+ \dots +c_{nc}psi(t-n_c)$ , where  $c$  is the polynomial coefficients  $[1, c_1, \dots, c_{nc}]$  for noises to improve the property of  $psi$  so that

the estimator converges more reliable. It is seen that  $\phi(t)$  is related to  $\psi$  at both current and previous time. The relationship between  $\psi$  and  $\phi$  in delta ( $\delta$ ) transform is expressed as in Eq. 8-28, where the  $c$  polynomial is a deltarised version of the coefficients,

$$\phi(t) = c(\delta) \cdot \psi(t) \quad \text{Eq. 8-28}$$

or its full expression in Eq. 8-29.

$$\delta^{nc-1}\phi(t-nc) = \delta^{nc-1}\psi(t-nc) + c_1\delta^{nc-2}\psi(t-nc) + \dots + c_{nc}\psi(t-nc) \quad \text{Eq. 8-29}$$

To achieve deltarised  $\psi$  at current time, this equation is manipulated as shown in Eq. 8-30. It is a two-dimensional array, the number of rows is equal to the size of vectors in  $\phi$  and the number of columns is equal to the number of terms in the  $c$  polynomial.

$$\delta^{nc-1}\psi(t-nc) = \delta^{nc-1}\phi(t-nc) - c_1\delta^{nc-2}\psi(t-nc) - \dots - c_{nc}\psi(t-nc) \quad \text{Eq. 8-30}$$

When using the  $z$  transform, [Ljung99] makes use of the fact that past values of  $\psi$  and  $\phi$  are readily available in the estimator, so that  $\psi(t)$  can be obtained easily from available data vectors in the estimator. This is because the nature of the data does not change with storage position in the data vector. However, when using the delta transform  $\delta^{nc-1}\psi(t-nc)$  can not be obtained using the same procedure, because samples in the data vector are different orders of  $\delta$ . All these data vectors have to be refilled at each sampling interval.

The vectors in  $\delta^{nc-1}\psi(t-nc)$  are shown in Eq. 8-31 if, for example, the coefficients array  $nn$  is [3 4 2 1 4].

$$-\delta^2 y(t-3) \dots -\delta^0 y(t-3), \delta^3 u(t-4) \dots \delta^0 u(t-4), \delta^1 \bar{\epsilon}(t-2) \quad \delta^0 \bar{\epsilon}(t-2), 1, \delta^3 v(t-4) \dots \delta^0 v(t-4) \quad \text{Eq. 8-31}$$

$-\delta^2 y(t-3) \dots -\delta^0 y(t-3)$  are obtained by deltarising  $-y(t-1) \dots -y(t-3)$  using deltarise function given in 8.2.1. The undeltarise function in 8.2.2 is also required to



firstly fully undeltarise each row of  $dpsi$  at previous time to achieve the current time  $psi(t)$ , e.g.,  $-y(t-1)\dots-y(t-3)$  is achieved by fully undeltarising  $-\delta^2y(t-3)\dots-\delta^0y(t-3)$ . The undeltarise function is employed again but only for a single iteration ( $numb = 1$ ) to obtain  $dpsi$  the next time, so this matrix is shifted forward once. The last term ( $\delta^0psi$ ) in the array is then thrown away, so  $\delta^1psi$  becomes  $\delta^0psi$  and so on in order to add the new array in front and keep the algorithm consistent.

Finally the vector array  $phi$  is updated with the new estimation including the noise vector that is updated by residual error  $epsilon$ . We must keep in mind that  $depsilon$  is the deltarised version of  $epsilon$ , in this case we only have  $depsilon$  at current time, thus the undeltarise function is needed for  $epsilon$ , as shown in Eq. 8-32.

$$epsilon = undeltarise([depsilon dphi(iioc)], T_s, 0) \quad \text{Eq. 8-32}$$

where  $dphi(iioc)$  includes noise vectors at previous time,  $iioc$  is the index array for noise vectors in  $dphi$ ,  $T_s$  is the sampling rate, 0 indicates the full undeltarisation as has been discussed above.

The complete MATLAB codes for the AME system can be found in Appendix H.2.1.

### 8.2.3.2 The AMP

Similar to the AME both the deltarise and undeltarise functions are required through the system. Unlike the AME, the predicted value  $y$  is used for updating the vector array  $phi$ , whereas in the AME inputs  $u$ ,  $e$  and output  $y$  are obtained from the training data.

To obtain the output data  $y$ ,  $dy$  is fully undeltarised by employing the undeltarise function shown in Eq. 8-33:

$$y = undeltarise([dy -dphi(iiaa)], T_s, 0) \quad \text{Eq. 8-33}$$

where  $dphi(iiaa)$  includes the previous deltarised output vector,  $iiaa$  is the array for the outputs in  $dphi$ ,  $T_s$  is the sampling rate, 0 indicates the full undeltarisation is utilized.

The full MATLAB codes for the AMP system can be found in Appendix H.2.2.

### 8.3 *Experimental Results*

In this subsection the system is investigated in order to prove that it is able to hunt for known models and converges well.

#### 8.3.1 A Single Model Detection

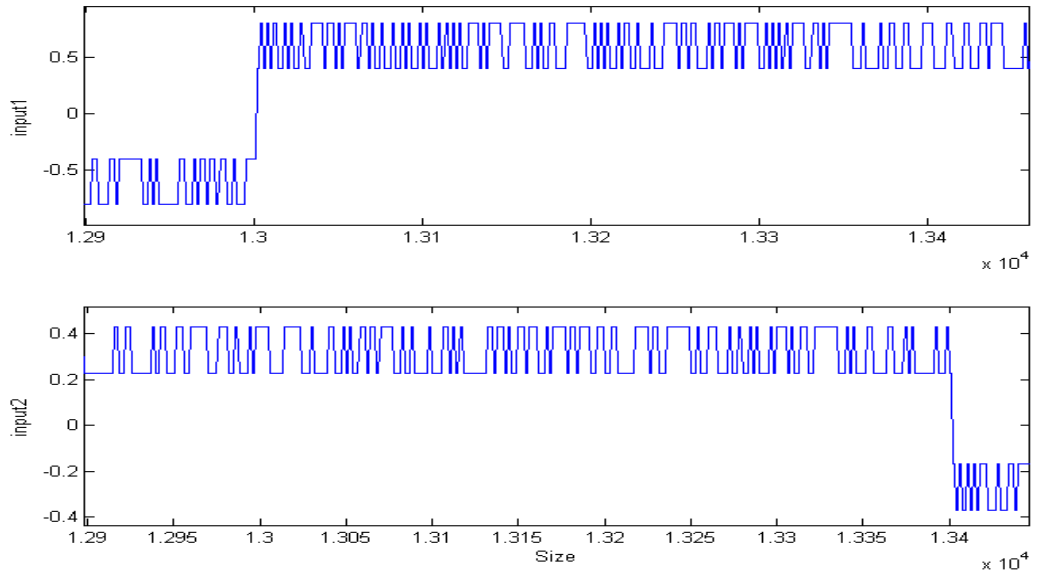
The process follows two steps:

1. *The AMP system is applied to a known linear model. Both input data and output data are stored in a text file.*
2. *The AME generates the model based on these data*

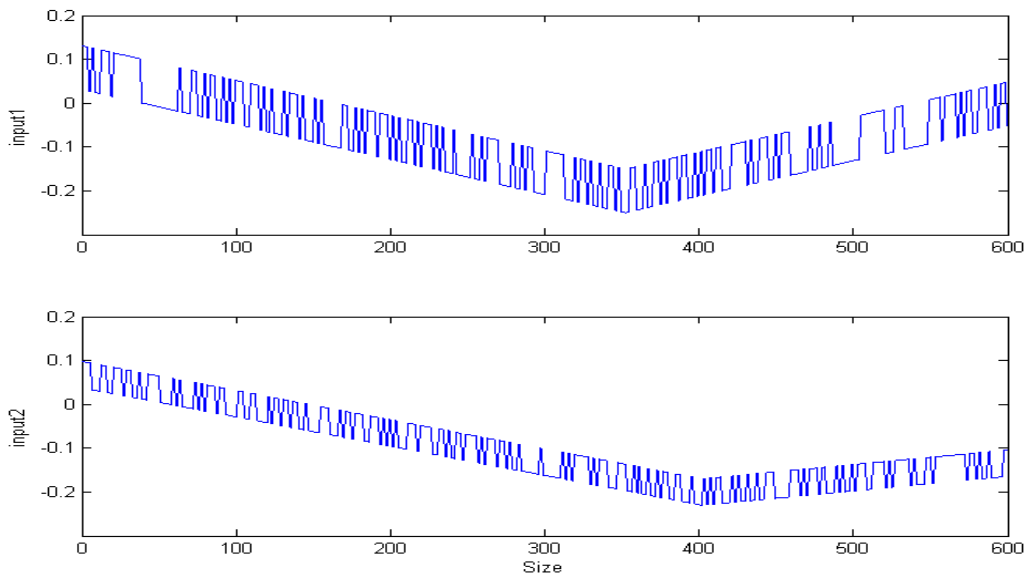
The reason to work in the opposite way is that the AMP is less complicated than the AME and it is easier to find out whether or not the delta operator works well in the MMGSD. The system used in this example is a linear model given in Eq. 8-34.

$$V_o = \frac{-(20s + 500)V_{in} + (10s + 250)V_{ip} + 250V_{offset}}{s^2 + 20s + 500} \quad \text{Eq. 8-34}$$

Two types of training data are generated from the PRBSG for the MISO AMP: one is a 0.6V, 50Hz square waveform with a 0.12V, 100kHz PRBS superimposed on it for the inverting input, a similar signal but with lower amplitude and frequency is applied to the non-inverting input as shown in Figure 8-2 with 14,000 samples. Another training waveform is a 0.2V, 100Hz triangle waveform with a 0.05V, 100kHz PRBS superimposed on it for the inverting input, the second input is a similar signal but with lower amplitude and frequency for the non-inverting input displayed in Figure 8-3 with 14,000 samples.

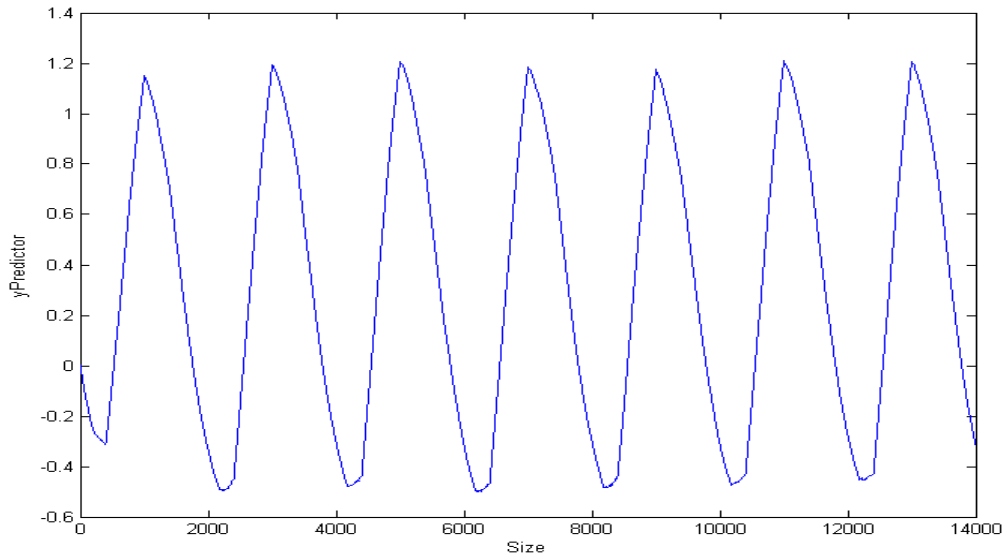


**Figure 8-2:** The square PRBS signal



**Figure 8-3:** The triangle PRBS signal

The output signal from the AMP using the square PRBS is plotted in Figure 8-4.

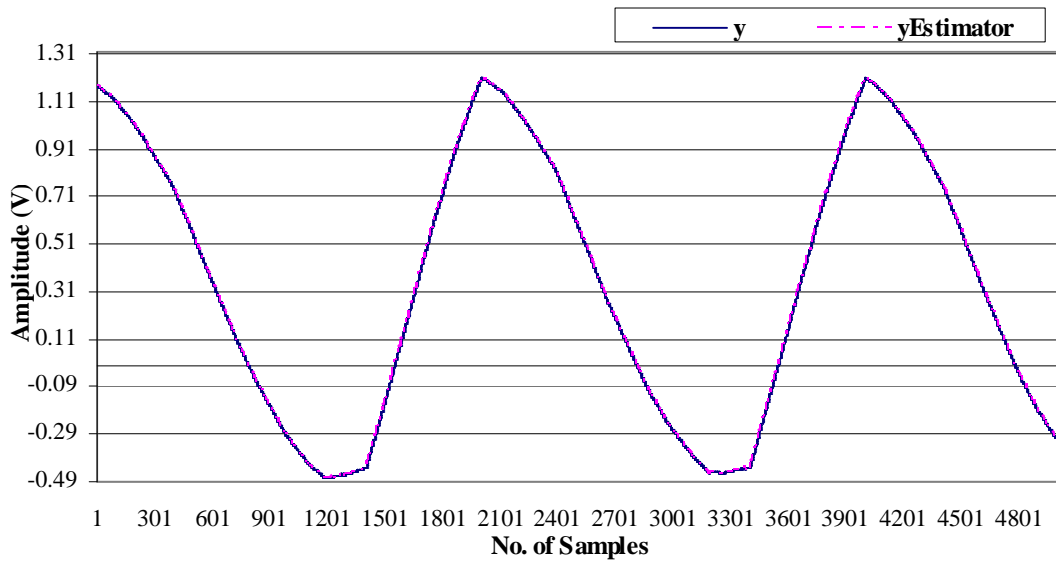


**Figure 8-4:** The predicted signal

The AME is employed to generate the model seen in Eq. 8-35 with  $T_s$  of 0.1ms. It is seen that two models can be matched referring to their coefficients.

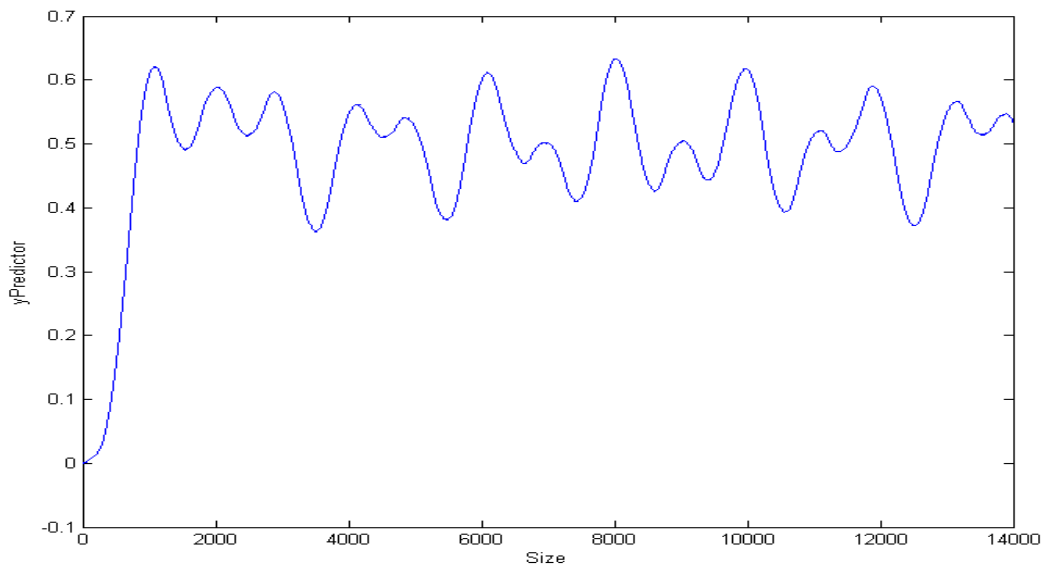
$$V_o = \frac{-(20s + 500)V_{in} + (10s + 250)V_{ip} + 250.02V_{offset}}{s^2 + 20s + 500} \quad \text{Eq. 8-35}$$

The output signal is depicted in Figure 8-5 (last 5,000 samples). It is seen that the original signal is closely be matched to the estimated signal. Using Eq. 5-2 the average difference between the original signal  $y$  and estimated signal  $y_{Estimator}$  is  $8.1567e-8\%$ .



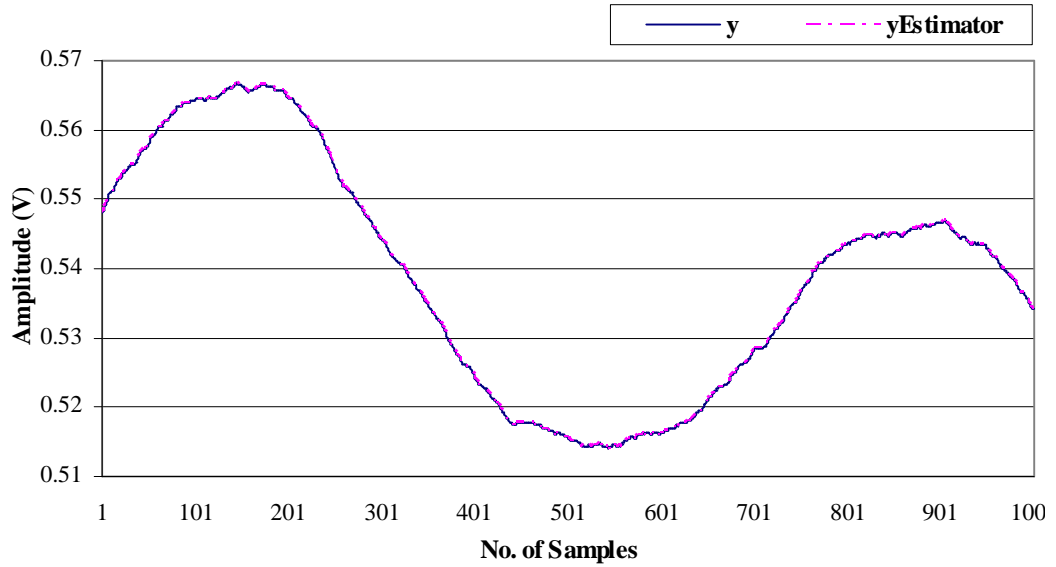
**Figure 8-5:** The estimated signal

The triangle PRBS is then used following the same process. The predicted signal is shown in Figure 8-6.



**Figure 8-6:** The predicted signal

After that the estimator is employed to generate the model. The same coefficients are achieved from the two models. The output signal from AME is shown in Figure 8-7 (last 1,000 samples). The average difference between the original signal  $y$  and estimated signal  $yEstimator$  is  $7.24e-12\%$ .



**Figure 8-7:** The estimated signal

The similar experiment as above is also implemented using the SystemVision simulator in order to prove the whole process works well in both MATLAB and VHDL-AMS. More details on the implementation can be found in Appendix J. It has proved that the MMGSD can produce a single model accurately.

### 8.3.2 Comparison of convergence speed between MMGS and MMGSD

In this section using the same procedure in section 8.3.1 we compare two systems (MMGS and MMGSD) based on the linear model seen in Eq. 8-34. The same triangle PRBS in Figure 8-3 is employed.

The AME system in the MMGSD is used to generate a model using the data from the AMP system. The model is shown in Eq. 8-36, only 250 samples were used.

$$V_o = \frac{-(20.01s + 499)V_{in} + (10s + 249.5)V_{ip} + 249.7V_{offset}}{s^2 + 20.01s + 499.5} \quad \text{Eq. 8-36}$$

It is seen that coefficients in both models are matched reasonably well, and the difference does not affect accuracy of the output signal.

We then investigate the MMGS. This continuous-time model is first converted into its corresponding discrete-time model using the system identification toolbox in MATLAB,

as seen in Figure 8-8, with a sampling rate of 0.1ms. In Figure 8-8 B1, B2 and B3 are coefficients for the first input, second input and offset, respectively; F1, F2 and F3 are the coefficients for the output:

$$\begin{aligned}
 &\text{Discrete-time IDPOLY model: } y(t) = [B(q)/F(q)]u(t) + e(t) \\
 &B1(q) = -0.002401 q^{-1} + 0.002394 q^{-2} \\
 &B2(q) = 0.0012 q^{-1} - 0.001197 q^{-2} \\
 &B3(q) = 1.799e-006 q^{-1} + 1.797e-006 q^{-2} \\
 &F1(q) = 1 - 1.998 q^{-1} + 0.9976 q^{-2} \\
 &F2(q) = 1 - 1.998 q^{-1} + 0.9976 q^{-2} \\
 &F3(q) = 1 - 1.998 q^{-1} + 0.9976 q^{-2}
 \end{aligned}$$

**Figure 8-8:** Coefficients under discrete-time from the AMP in the MMGS

The MMGS loads these data from a text file to generate the model show in Eq. 8-37, only 500 samples are required.

$$V_o = \frac{-(0.002401q^{-1} - 0.002393q^{-2})V_{in} + (0.0012q^{-1} - 0.001197q^{-2})V_{ip} + 0.000002q^{-2}V_{offset}}{1 - 1.9975q^{-1} + 0.99752q^{-2}}$$

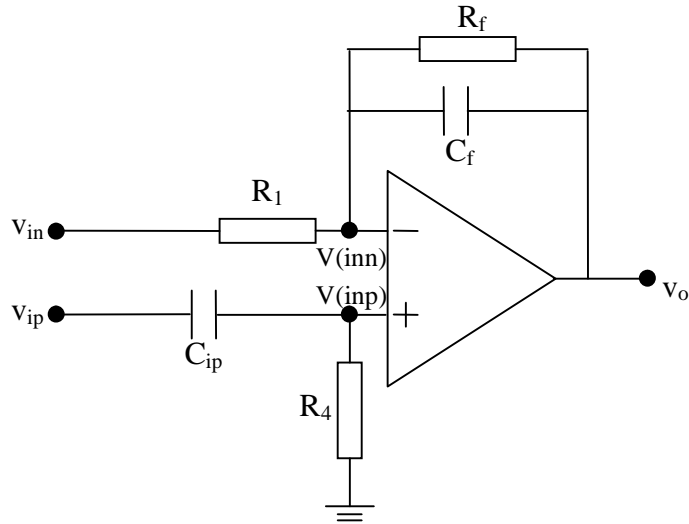
Eq. 8-37

Comparing the coefficients in Figure 8-8 with Eq. 8-37 we see that the two groups of discrete-time coefficients are reasonably close.

It is thus proved that the MMGSD is able to generate a model that has been set up. Moreover, it can converge about two times faster than the MMGS using a linear model.

### 8.3.3 System Test Using a Lead-lag Circuit

We demonstrate that the MMGSD is able to generate a single more complex linear model than section 8.3.1. A linear lead-lag circuit is employed using a high-pass filter and low-pass filter with frequencies of 1kHz and 10Hz, as shown in Figure 8-9, where  $R_l = 1k\Omega$ ,  $R_f = 10k\Omega$ ,  $C_f = 0.15915\mu F$ ,  $R_4 = 10k\Omega$  and  $C_{ip} = 15.915nF$ .



**Figure 8-9:** A linear system with a high pass and low pass filter

The transfer function is shown in Eq. 8-38:

$$v_o = -\frac{R_f}{R_1 \cdot R_f \cdot C_f \cdot s + R_1} \cdot v_{in} + \left( \frac{(R_1 \cdot R_f \cdot C_f \cdot s + R_1 + R_f) \cdot (C_{ip} \cdot R_4 \cdot s)}{(R_1 \cdot R_f \cdot C_f \cdot s + R_1) \cdot (C_{ip} \cdot R_4 \cdot s + 1)} \right) \cdot v_{ip}$$

Eq. 8-38

The system was analysed using the system identification toolbox in MATLAB to generate the polynomial based model seen in Figure 8-10 using the same PRBS training signals as above. The sampling interval is 0.1ms. B1 and B2 are coefficients for inputs, F1 and F2 represent the output.

```

Continuous-time IDPOLY model:
y(t) = [B(s)/F(s)]u(t) + e(t)

B1(s) = -62.83 s - 3.948e004
B2(s) = s^2 + 69.12 s
F1(s) = s^2 + 634.6 s + 3948
F2(s) = s^2 + 634.6 s + 3948

```

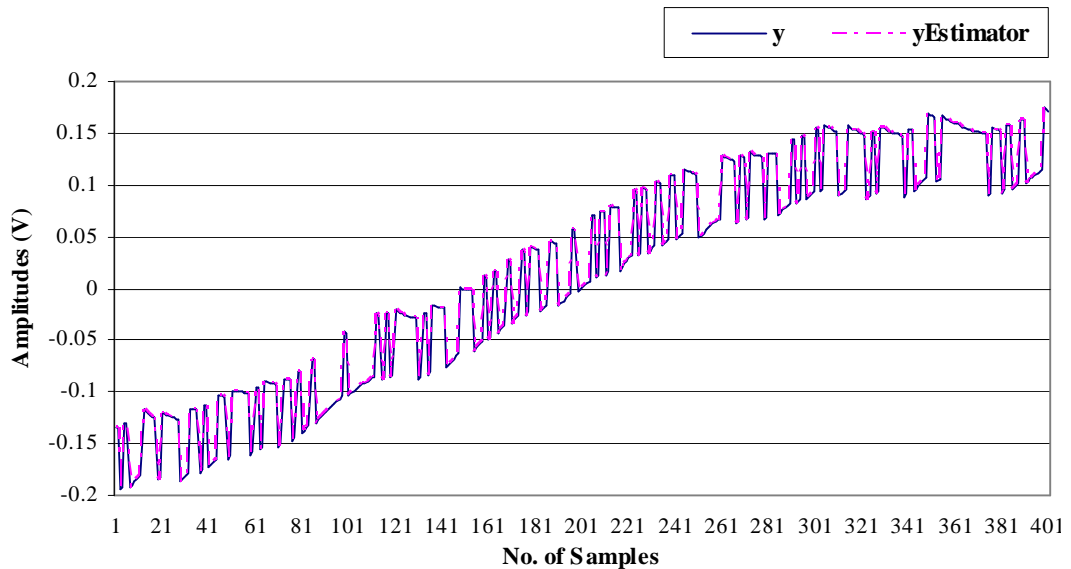
**Figure 8-10:** The coefficients from the high-pass filter

Both input and output data are then stored in a text file. The MMGSD was used to generate a model based on this data, as seen in Eq. 8-39. The output coefficients are



reasonably close to F1 and F2, and the two input coefficients are very close to B1 and B2 in Figure 8-10. The output signals are shown in Figure 8-11 with last 400 samples:

$$V_o = \frac{-(62.812s + 38252)V_{in} + (s^2 + 67.16s)V_{ip}}{s^2 + 615.3s + 3825.2} \quad \text{Eq. 8-39}$$



**Figure 8-11:** The estimated signal

It is seen that the MMGSD can generate the model that has been set. The average difference between two signals is 2.992e-13%.

Comparing with the MMGS, the MMGSD is able to handle both the low-pass and high-pass filters.

### 8.3.4 Verification on the Multiple Model Generation Approach

The aim of this section is to verify that the MMGSD is able to generate multiples models that have been set up. The same triangle PRBS stimulus in Figure 8-3 is employed. Three stable models used are shown in Eq. 8-40. Each of them has two input parameters and one offset parameter. The intervals used to divide the range of this stimulus for these models are: [-0.25V 0.0V 0.095V 0.25V], the sampling rate  $T_s$  is 0.1ms. The number of samples in each model ( $v_{o1}$ ,  $v_{o2}$  and  $v_{o3}$ ) is obtained: count = 6942,

3300, 3758, the first model contains more samples than others, which indicates that it may require more time to tune the system.

$$\begin{aligned}
 V_{o1} &= \frac{-(20s + 500)V_{in1} + (10s + 250)V_{ip1} + 250V_{offset1}}{s^2 + 20s + 500} \\
 V_{o2} &= \frac{-(20s + 500)V_{in2} + (10s + 250)V_{ip2} + 250V_{offset2}}{s^2 + 20s + 1000} \\
 V_{o3} &= \frac{-(20s + 500)V_{in3} + (10s + 250)V_{ip3} + 250V_{offset3}}{s^2 + 20s + 1500}
 \end{aligned}
 \tag{Eq. 8-40}$$

It is seen that the steady-states of the models are not identical at the transfer points, which may result in discontinuities. However, it is known that during estimation different models should have the same steady-state values in their interfaces because of the way the models have been trained. It has been mentioned with reference to Figure 7-5 that the way that we are dealing with models selection is not expected to achieve completely bumpless transfer but should suffer from minimum discontinuities, because we have all models are running in parallel and just switch their outputs at the right time, when two neighbouring models are not near the switching interface, the outputs are different. However, when they are approaching to the same switching interface, their outputs will line up, there should not be a discontinuity.

Initially the AMP is used to run the models with the triangle PRBS. After simulation both input and output data are stored in a text file. The AME then loads the data to produce the models shown in Eq. 8-41. These coefficients generated are reasonably close to those in Eq. 8-40, although the third model is not as accurate as the others because as the pole value is gets higher, instability is more likely. This can be improved by manually selecting a smaller value such as 1200 instead of 1500.

$$\begin{aligned}
 V_{o1} &= \frac{-(20s + 500)V_{in1} + (10s + 250)V_{ip1} + 249V_{offset1}}{s^2 + 20s + 499.5} \\
 V_{o2} &= \frac{-(20s + 500)V_{in2} + (10s + 250)V_{ip2} + 250.01V_{offset2}}{s^2 + 20s + 1000} \\
 V_{o3} &= \frac{-(20s + 483.5)V_{in3} + (10s + 239)V_{ip3} + 248.3V_{offset3}}{s^2 + 18.43s + 1390}
 \end{aligned}
 \tag{Eq. 8-41}$$

Furthermore, the same procedure was also implemented but using four models with different poles. The models are shown in Eq. 8-42. The intervals used to divide the

range of this stimulus for these models are: [-0.25 0.01 0.11 0.18 0.25], the sampling rate  $T_s$  is 0.1ms.

$$\begin{aligned}
 V_{o1} &= \frac{-(20s + 500)V_{in1} + (10s + 250)V_{ip1} + 250V_{offset1}}{s^2 + 20s + 500} \\
 V_{o2} &= \frac{-(20s + 500)V_{in2} + (10s + 250)V_{ip2} + 250V_{offset2}}{s^2 + 20s + 1000} \\
 V_{o3} &= \frac{-(20s + 500)V_{in3} + (10s + 250)V_{ip3} + 250V_{offset3}}{s^2 + 20s + 1500} \\
 V_{o4} &= \frac{-(20s + 500)V_{in4} + (10s + 250)V_{ip4} + 250V_{offset4}}{s^2 + 20s + 500}
 \end{aligned}
 \tag{Eq. 8-42}$$

The number of samples on each model ( $v_{o1}$ ,  $v_{o2}$ ,  $v_{o3}$  and  $v_{o4}$ ) is: count = 7255, 3515, 1941, 1289. This indicates that the first model uses more samples to tune the system.

After simulation both input and output data are stored in a text file. The AME then loads this data to produce the models shown in Eq. 8-43. It is seen that the coefficients generated are close to the original ones.

$$\begin{aligned}
 V_{o1} &= \frac{-(20s + 511.84)V_{in1} + (10s + 252)V_{ip1} + 249.5V_{offset1}}{s^2 + 20.14s + 500.73} \\
 V_{o2} &= \frac{-(20s + 442.8)V_{in2} + (10s + 230.5)V_{ip2} + 245V_{offset2}}{s^2 + 18.95s + 1086.1} \\
 V_{o3} &= \frac{-(20s + 471)V_{in3} + (10s + 245.4)V_{ip3} + 245.1V_{offset3}}{s^2 + 20.4s + 1498.57} \\
 V_{o4} &= \frac{-(20s + 497.4)V_{in4} + (10s + 249.1)V_{ip4} + 249.5V_{offset4}}{s^2 + 19.94s + 499.23}
 \end{aligned}
 \tag{Eq. 8-43}$$

This indicates that the MMGSD is able to generate various suitable models.

### 8.3.5 Nonlinearity Modelling

In this section the open-loop op amp SPICE netlist from Figure 4-1 is modelled using training data which creates strong nonlinearity (into saturation). A new stimulus is used: a 2.5V, 83.33Hz triangle waveform with a 0.5V, 100kHz PRBS superimposed on it. A similar signal but with same amplitude and lower frequency is applied to the non-inverting input. Five models are generated. The input voltage range is  $\pm 2.5V$ . The thresholds and the number of samples for each model are shown in Figure 8-12:

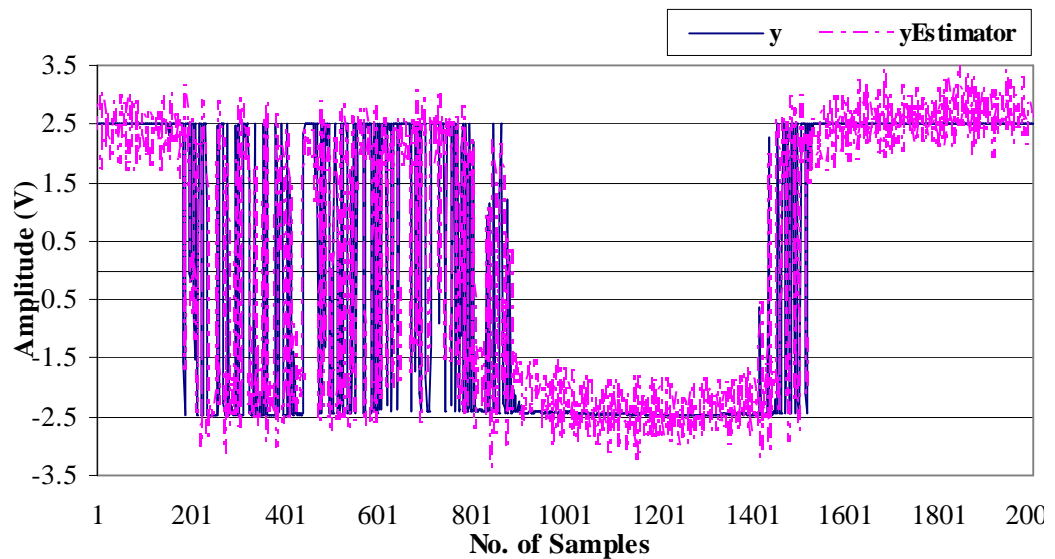
```

threshold = [-2.5 -1.5 -0.5 0.5 1.5 2.5]
count = 2263 2010 2267 2452 3048

```

**Figure 8-12:** Threshold and samples for each model

The estimated signal  $yEstimate$  is illustrated in Figure 8-13 with 2,000 samples:



**Figure 8-13:** The estimated signal with nonlinearity

It is seen that  $yEstimate$  is able to match the original output  $y$ , the difference between two signals is 9.5768% using an average difference measurement. Although there is some noise due to high sampling rate of 10kHz for the delta operator. Moreover, the estimator struggles to obtain enough information to make the best guess because there is no relationship between input and output in the saturation regions, that is, the output is not dependant on input. It has to generate excitation itself. This can be improved by using a saturation detector to delete the samples in the saturation region, so the estimator will focus on available information.

## 8.4 Conclusion

In this chapter the multiple model generation system using delta operator (MMGSD) is developed for either SISO or MISO models from transistor level SPICE simulations to perform high level fault modelling (HLFM). The MMGSD is able to converge twice as fast as discrete-time models using a linear model. We have shown that acceptably small

discontinuities can be achieved between models generated by using the algorithm developed. Moreover, it can handle both low-pass and high-pass filters accurately, and model nonlinear behaviours.

# ***Chapter 9: High Level Fault Modelling and Simulation based on Models from the MMGSD***

## ***9.1 Introduction***

The aim of the chapter is to evaluate that during high level fault modelling (HLFM) the models generated by the MMGSD can achieve better results in terms of simulation speed and accuracy than transistor level fault simulation (TLFS) and results from [Bartsch99]. The multiple model conversion system using delta transform (MMCSO) was developed to convert the models from the MMGSD into a suitable format, i.e., from MATLAB to VHDL-AMS. The model selection process is implemented by selecting the output of the model within its corresponding input range.

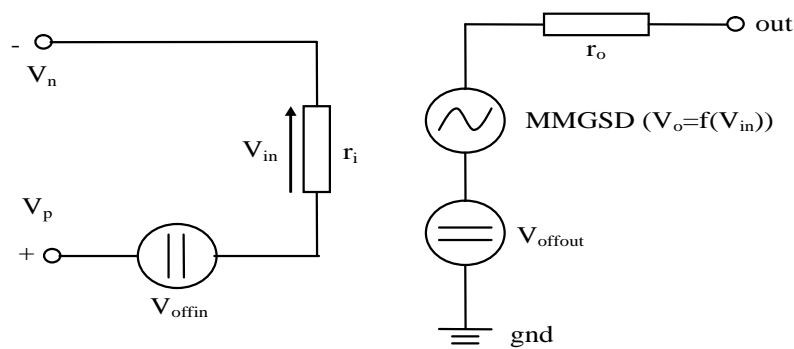
HLFM is run using the SystemVision simulator from Mentor Graphics [SystemVision], the faulty transistor level op amp in HSPICE will be replaced by the model from the MMCSO in VHDL-AMS and the rest of them remain at the transistor level. With this procedure we can observe if our model is able to model the fault and propagate it correctly. Only short faults are investigated, other faults will be covered in future work. The short faults are modelled at transistor level using a  $1\Omega$  resistor connected between the shorted nodes. This can be realised by using the fault injector ANAFINS, which is the part of the transistor level fault simulator ANTICS [Spinks98]. More details about how it works can be found in [Spinks04].

Simulation results in terms of accuracy and simulation speed are compared under the same conditions using the average confidence measure (ACM) [Spinks98]. The ACM basically measures the distance between two waveforms taking variability into account. To facilitate comparison the fault coverage has to be appropriate. On one hand, if the error band for the waveform is very large all faults may be undetectable; on the other hand, if the error band is narrow enough all faults can be detectable. We need to avoid the extreme situations above, so that we are able to investigate the quality of simulation for different models.

The following sections are outlined: section 9.2 introduces how to implement the multiple model conversion system using delta operator (MMCSO) based on a behavioural model; quality measurement methods based on mathematical equations are introduced in section 9.3; some experimental results from high level modelling (HLM) and high level fault modelling (HLFM) are given in section 9.4 using a biquadratic low-pass filter; section 9.5 supplies the conclusion.

## 9.2 The Approach for Multiple Model Conversion System using Delta Operator (MMCSO)

In this section the algorithm for automatically generating a VHDL-AMS model is introduced. This is based on the structure of a behavioural model shown in Figure 9-1, which is similar to Figure 7-1.



**Figure 9-1:** The structure of the behavioural op amp model

Multiple models from the MMGSD are included in the VCVS (implementing  $V_o = f(V_{in})$ ) to handle nonlinearity. In this case the same models from MMGSD are used in chapter 8, and the input voltage range is  $\pm 2.5$  V. Two linear resistors  $r_i$  and  $r_o$  represent the input impedance and output impedance, respectively,  $v_{offin}$  and  $v_{offout}$  are parameters for modelling the input offset and output offset, respectively.

The MMCSO converts models from the MMGSD into this behavioural model. Each of the models behaves as a continuous domain low-pass filter as shown in Eq. 9-1, where  $v_n$ ,  $v_p$  and  $v_{offset}$  represent the three inputs;  $v_{out}$  is the output;  $a_1 \dots a_{na}$  are coefficients of the output;  $b_0, b_1 \dots b_{nb}$ ,  $e_0, e_1 \dots e_{ne}$  are coefficients of the first and second inputs, respectively; and  $d$  represents coefficients for the offset.

$$v_{out} = \frac{b_0 \cdot s^{nb} + b_1 \cdot s^{(nb-1)} + \dots + b_{nb}}{s^{na} + a_1 \cdot s^{(na-1)} + \dots + a_{na}} v_n + \frac{e_0 \cdot s^{ne} + e_1 \cdot s^{(ne-1)} + \dots + e_{ne}}{s^{na} + a_1 \cdot s^{(na-1)} + \dots + a_{na}} v_p + \frac{d}{s^{na} + a_1 \cdot s^{(na-1)} + \dots + a_{na}} v_{offset}$$

Eq. 9-1

The equation above can be easily implemented in VHDL-AMS with the help of either ‘dot or ‘ltf attributes; both produce the same results. The latter is used in this case because it is far easier to derive and implement a higher order transfer function as a simple ratio of s-domain polynomials than it is to derive the equivalent differential equations for the functions [Ashenden03]. The model in VHDL-AMS is shown in Eq. 9-2.

$$v_{out} == v_n 'ltf(num\_1,den) + v_p 'ltf(num\_2,den) + v_{offset} 'ltf(num\_3,den) \quad \text{Eq. 9-2}$$

where ‘ltf is the attribute for forming the transfer function; num and den are the coefficients for the numerator and denominator, respectively; v<sub>n</sub>, v<sub>p</sub> and v<sub>offset</sub> represent the input voltages; v<sub>out</sub> is the output voltage.

The MMCSO extracts these coefficients one by one dynamically from each model library to form the equation in Eq. 9-1. The model selection algorithm has been discussed in Figure 7-3, now showing in Figure 9-2.

```

Display the transfer function for the first model
Display the transfer function for the second model
      .
      .
      .
Display the transfer function for the last model

If the input signal is within the range for the first model use
    The output voltage of the first model is selected
Else if the input signal is within the range for the second model use
    The output voltage of the second model is selected
      .
      .
      .
Else the input signal is not included in these ranges
    Either the output voltage of the first or the last model is selected

```

**Figure 9-2:** The algorithm for the model selection

It has been discussed that by doing this way we can achieve reasonable bumpless transfer. However, higher simulation speed may not be achieved. This can be improved by feeding only the neighbouring models instead of all models.



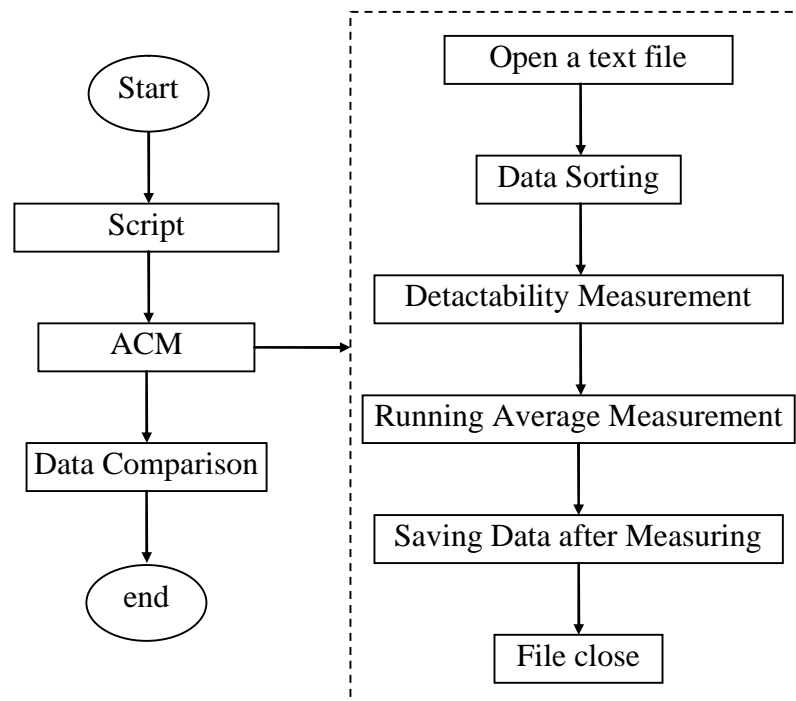
### 9.3 Introduction to Quality Measurement

In the section quality measurement methods for simulation accuracy and speed are introduced in subsection 9.3.1 and subsection 9.3.2, respectively.

#### 9.3.1 Average Confidence Measurement

The object of the section is to calculate the average of a number of values (output voltage in this case). The method used is the average confidence measure or ACM [Spinks98]. The aim of the section is to verify that our model performs better than other's macromodel and that our model can model faulty behaviour with good accuracy compared with TLFS.

Unlike previous work using ANACOV [Spinks98] this measurement was implemented in MATLAB to provide better integration with the other tools developed here. The whole process consists of multiple steps illustrated in Figure 9-3. A script is used to create a MATLAB editor file that accesses each of the files which require processing to calculate ACM for each file.



**Figure 9-3:** A flowchart for fault coverage measurement

The ACM skips header text and reads data from the analogue SystemVision simulation results file. For fault analysis and detectability measurement, a fixed envelope is applied not only around the fault-free response to define the region of acceptability, but also around each faulty circuit response. The points at which the two envelopes do not overlap are classed as detectable. This can be expressed mathematically as follows:

$$\begin{aligned}
 G_L[i] &= G[i] - \delta G_L \\
 G_H[i] &= G[i] + \delta G_H \\
 F_L[i] &= F[i] - \delta F_L \\
 F_H[i] &= F[i] + \delta F_H
 \end{aligned}
 \tag{Eq. 9-3}$$

where  $G[i]$ ,  $F[i]$  represent the fault-free (good) and faulty signals at  $i$ th point, respectively;  $G_L[i]$ ,  $G_H[i]$  indicate the low and high value of the envelope of the good signals at the  $i$ th point, respectively;  $F_L[i]$ ,  $F_H[i]$  represent the low and high value of the envelop of the faulty signal at the  $i$ th point, respectively;  $\delta G_L$ ,  $\delta G_H$  indicate the lower and upper bounds of the good signal, respectively;  $\delta F_L$ ,  $\delta F_H$  represent the lower and upper bounds, respectively, which are defined by the user.

Different types of circuit and test techniques will use various fault detection criteria which require different envelope regions. In all cases however a description for a region of acceptability ( $G_L[i] < y_g < G_U[i]$ ) and a faulty response range ( $F_L[i] < y_f < F_U[i]$ ) is required. Fault detectability is based on the separation of the two regions. A sample point is defined as detectable if the two regions are non-overlapping at that point. Moreover, for each sample point, a confidence measure  $x[i]$  can be defined based on the distance between the faulty and fault-free envelopes since a larger distance implies that at a given sample point the circuit under a particular fault condition is more easily detectable. This is described by Figure 9-4.

$$x[i] = \begin{cases} G_L[i] - F_U[i] & \text{for } G_L[i] > F_U[i] \\ F_L[i] - G_U[i] & \text{for } F_L[i] > G_U[i] \\ 0 & \text{otherwise} \end{cases}$$

**Figure 9-4:** The conditions for detecting the distance of two signals

The number of detectable points  $NP$  out of a total  $d$  can be defined as shown in Eq. 9-4, where  $U(x) = 1$  when  $x > 0$ , otherwise it is 0.

$$NP = \sum_{i=0}^{d-1} U(x[i]) \quad \text{Eq. 9-4}$$

A fault is classed as detectable if  $NP > c$ , where  $c$  is a user defined cutoff value, normally it is 1, but this can be increased if a higher confidence in the results is required. For each fault, the mean separation distance between the good and faulty thresholds for all detectable sample points can also be used as an additional confidence measurement. The average confidence measurement (ACM) based on these detectable data is given in Eq. 9-5.

$$ACM = \frac{\sum_{i=0}^{d-1} x[i]}{NP} \quad \text{Eq. 9-5}$$

The number of times that the ACM is employed is equal to the number of detectable faults.

### 9.3.2 Mathematical Equations for Measuring Simulation Speed

This section is to investigate the simulation speed using two mathematical equations: total average speed and simulation speed-up. Both of them will be used to evaluate our behavioural model during HLM and HLFM.

#### 9.3.2.1 Average Time

The total average time of simulation is calculated using Eq. 9-6. The total simulation time is divided by the number of faults to give the average speed  $Ave\_time$  for an individual fault simulation;  $NS$  is the number of simulations;  $CPU[i]$  is the simulation cpu time of the  $i$ th fault.

$$Ave\_time = \frac{\sum_{i=0}^{NS-1} CPU[i]}{NS} \quad \text{Eq. 9-6}$$

### 9.3.2.2 Simulation Speed-up

The speed-up is calculated using Eq. 9-7, where  $t_{TLFS}$  is transistor level simulation time,  $t_{HLFM}$  is high level modelling time,  $t_{op}$  is operating point analysis time at transistor level, it is 100ms in the case.

$$speed\_up = \frac{t_{TLFS}}{t_{HLFM} + t_{op}} \quad \text{Eq. 9-7}$$

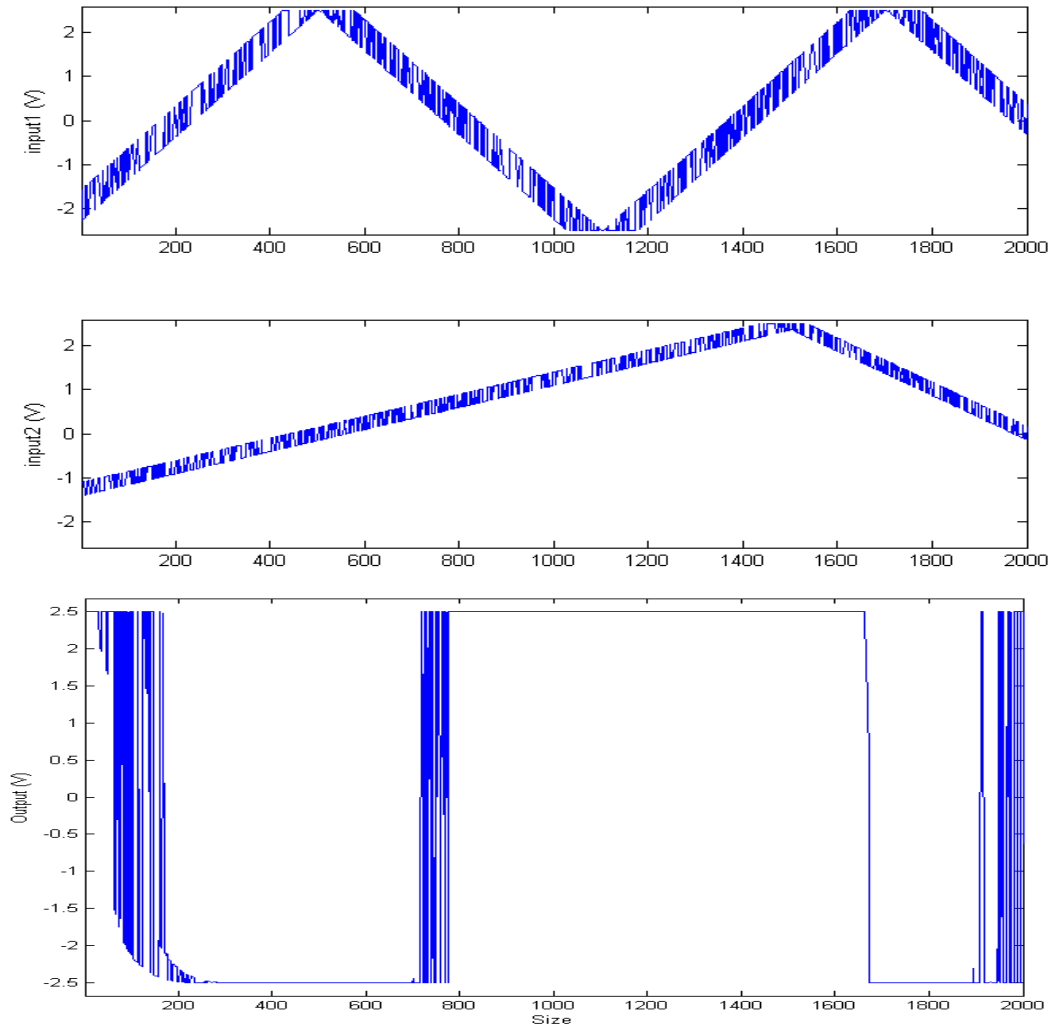
## 9.4 High Level Modelling and High Level Fault Modelling

In this section, both HLM and HLFM based on the models from the MMGSD are implemented in order to investigate if they are able to model faulty behaviour and handle nonlinearity in a system. The whole process requires the following steps:

1. The MMGSD generates models.
2. The MMCSO is used to convert these models into a VHDL-AMS behavioural model.
3. Transistor level fault-free and fault simulation are run based on short fault (one simulation run per fault).
4. The behavioural model from the MMCSO is used to replace the faulty transistor level op amp, the rest of circuits remaining the same.
5. The average coverage measurement (ACM) is used to measure quality of our model compared with TLFS. The average speed is also calculated and compared.
6. The same process for HLFM based on other published models is repeated from step 4.
7. Results from the two HLFM are compared.

The two-stage CMOS op amp in chapter 3 is employed, the training stimulus is a 2.5V, 83.33Hz triangle waveform with a 0.5V, 100kHz PRBS superimposed on it. A similar signal but with lower amplitude and frequency is applied to the non-inverting input, both inputs and output are shown in Figure 9-5. Only the last 2,000 samples are displayed. The  $x$  axis indicates the number of samples and the  $y$  axis show amplitudes of input voltage (V). The reason to use a higher amplitude of input signals is to force the generated models to cover saturation voltage ranges ( $\pm 2.5V$ ). By running the MMGSD

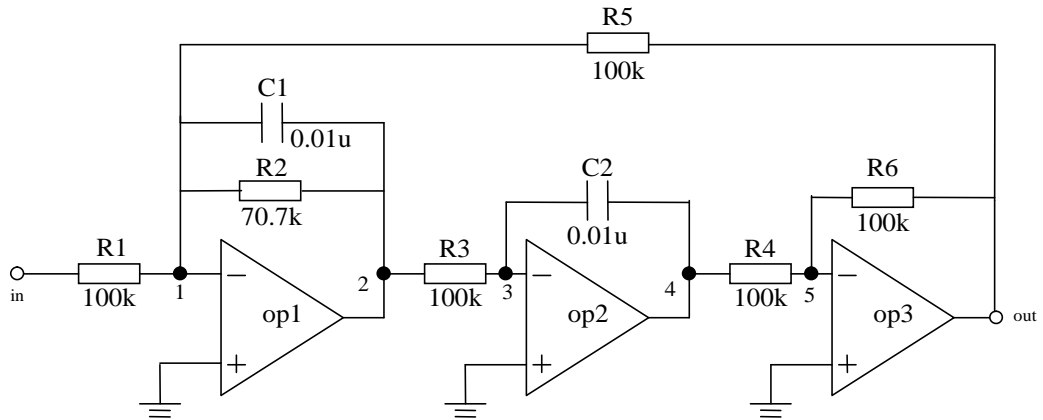
five models are generated. The MMCSO is then employed to convert these models into a VHDL-AMS behavioural model. Therefore, HLM and HLFM can be implemented under transient analysis in SystemVision [SystemVision].



**Figure 9-5:** The input signals with the saturation part

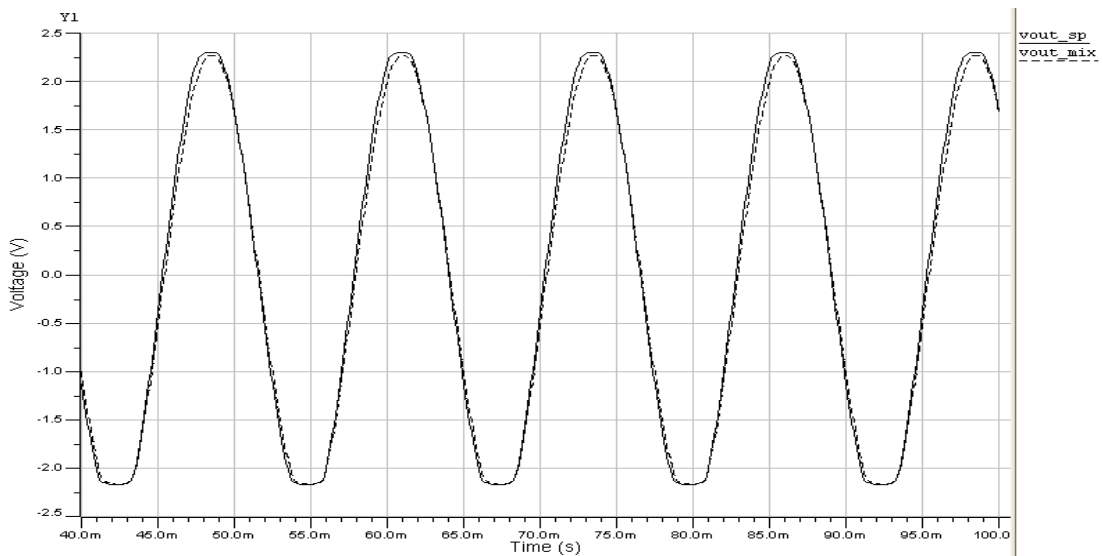
#### 9.4.1 High Level Modelling and Simulation

The same biquadratic low-pass filter used in chapter 8 was simulated using transient analysis seen in Figure 9-6. Two types of simulation are run: one is purely based on the transistor level circuit; another uses the MMGSD model to replace the first op amp *op1*. The input signal is a sine waveform with an amplitude of 2.5V at 80Hz. The simulation starts from 40ms to 100ms with step of 0.1ms.



**Figure 9-6:** The biquadratic low-pass filter

The output signals from TLS *vout\_sp* and HLS *vout\_mix* are plotted in Figure 9-7. It is seen that both signals can be matched with good accuracy. The total cpu time for HLM is 2.03s, and 1.062s for TLS.

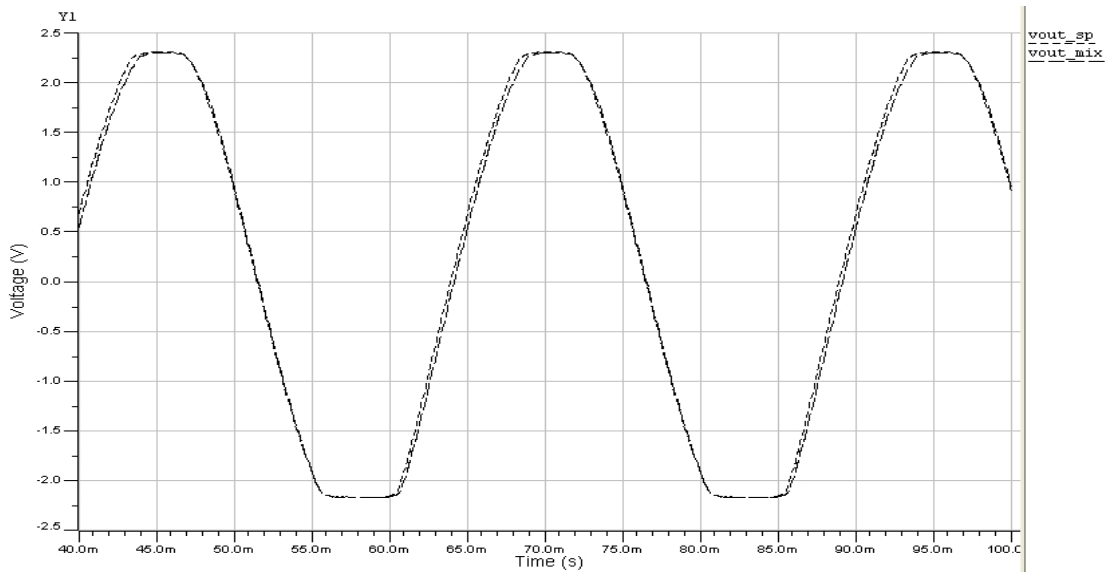


**Figure 9-7:** The output signals from the low-pass filter

It is seen that HLM is slower than TLS. This is because the SystemVision simulator has not been optimised to this kind of model structure, and so the computational overhead is high. Moreover, the model selection process requires time, this can be improved by feeding the neighbouring models instead of all models. Therefore, HLM using a single model instead of multiple models requires less time (1.07s) than 2.03s under the same

circumstance. However, accuracy becomes worse because the single model is not good enough to model nonlinearity.

Another transient analysis was conducted using this low-pass filter, the stimulus is a sine waveform with the amplitude of 2.5V at 40Hz. The simulation starts from 40ms to 100ms with step of 0.1ms. Output voltage signals are plotted in Figure 9-8.



**Figure 9-8:** The output signals from the low-pass filter

The results show the output signal at HLM *vout\_mix* can be matched reasonably well to the one at TLS *vout\_sp*, so the nonlinearity is modelled correctly.

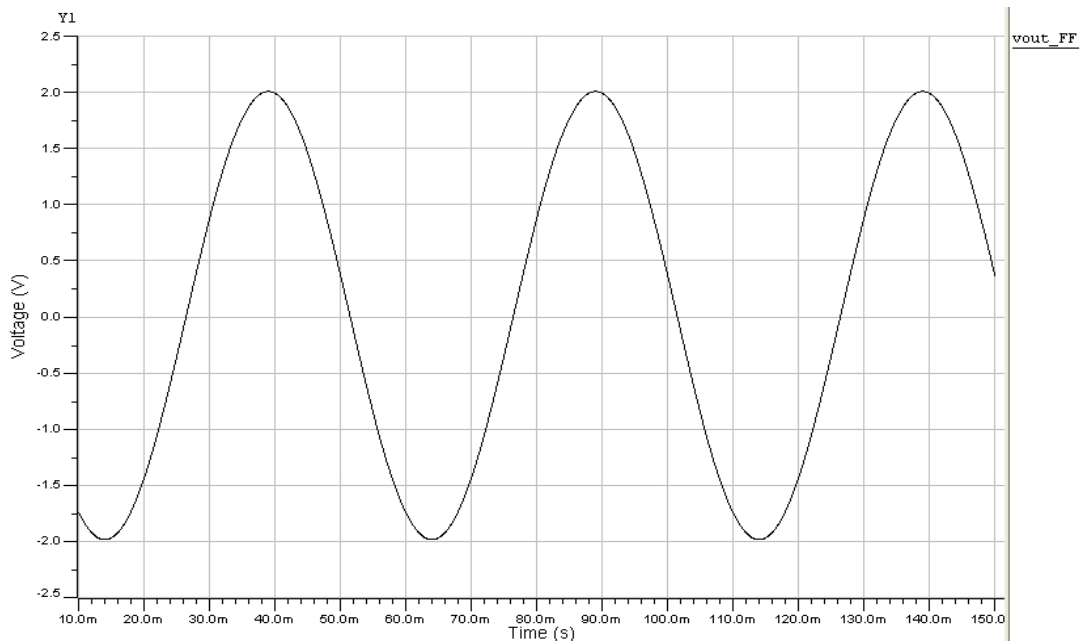
The total cpu time for HLM is 1.625s, it is 1.0s for TLS. The same process is then implemented but using one model instead of multiple models for HLM in order to investigate simulation time. It requires 0.953s of CPU time, which is faster than HLM using multiple models (1.625s), but accuracy is reduced significantly because the single model is not good enough to model nonlinearity.

Furthermore, accuracy is preserved when the behavioural model replaces other op amps in this filter. Unfortunately, significant speed-up using multiple models is not achieved compared with TLS due to significant computational overhead.

## 9.4.2 High Level Fault Modelling (HLFM) and High Level Fault Simulation (HLFS)

The aim of the section is to verify that our model performs better than other macromodels and that our model can model faulty behaviour with good accuracy compared with TLFS by using ACM seen in section 9.3.

By running fault modelling and simulation, three groups of results are obtained: two from HLFM, one from TLFS. Two models are employed from the same behavioural model from the MMGSD and the macromodel [Bartsch99] seen in Figure 3-2, respectively. All fault modelling and simulation are implemented under the same simulator setting and test circuit conditions. This circuit employed is the low-pass filter introduced in Figure 9-6. Only short faults are used. The stimulus is a sine waveform with the magnitude of 2.0V at 20Hz. Transient analysis is implemented from 10ms to 150ms with a step of 0.1ms. The signal from the transistor level fault-free simulation is plotted in Figure 9-9.



**Figure 9-9:** The output signal from the transistor level fault-free simulation

The same behavioural model from the MMGSD as above then replaces the faulty op amp for HLFM, and the rest of fault-free op amps remain unchanged, so that we can observe if it models faulty behaviour correctly. It is known that there are possible 99



short faults in this filter, so 99 simulations are performed for both TLFS and HLFM, with data for each saved in an individual file. The quality of output signals is determined using the ACM, in which the cutoff value  $c$  is set to 1, and the tolerance range for both good ( $\delta G$ ) and faulty signals ( $\delta F$ ) is set to 50uV after using different ranges in order to avoid extreme conditions mentioned above in this case.

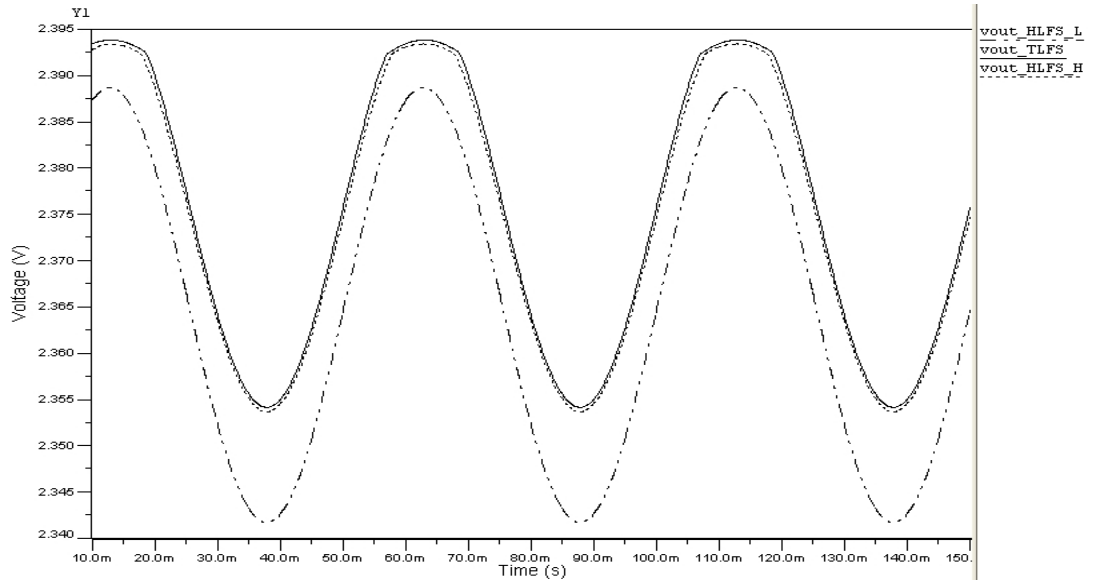
Testability measurement results using ACM from TLFS show that some faults (M8 and M13) are undetectable because they do not affect circuit behaviour. However, according to HLFM the same faults appear to be detectable because of difficulty in setting parameters such as offset voltage. These faults include  $m4\_gds\_1$ <sup>1</sup>,  $m4\_gds\_2$ ,  $m5\_gds\_1$ ,  $m5\_gds\_2$ ,  $m6\_gds\_1$ ,  $m6\_gds\_2$ ,  $m8\_dss\_2$ <sup>2</sup>,  $m8\_dss\_3$ ,  $m13\_gds\_1$  and  $m13\_gds\_2$ . Therefore, 89 out of 99 detectable faults are investigated under transient analysis.

The results demonstrate that HLFM based on linear macromodel can not model certain faults including  $M7\_gds\_2$ ,  $M10\_gds\_2$ ,  $M10\_gss\_3$  and  $M11\_dss\_3$  due to high nonlinearity. These can be modelled by the MMGSD model. Moreover, the MMGSD model can achieve more accurate fault simulation than the linear fault macromodel, for example,  $M11\_dss\_1$ , as illustrated in Figure 9-10, where  $vout\_TLFS$  is the output voltage signal from TLFS, and  $vout\_HLFS\_H$  and  $vout\_HLFS\_L$  represent the output voltage signals from the behavioural model by the MMGSD and the macromodel, respectively. It is seen that  $vout\_HLFS\_H$  is closer to  $vout\_TLFS$  than the macromodel  $vout\_HLFS\_L$ .

---

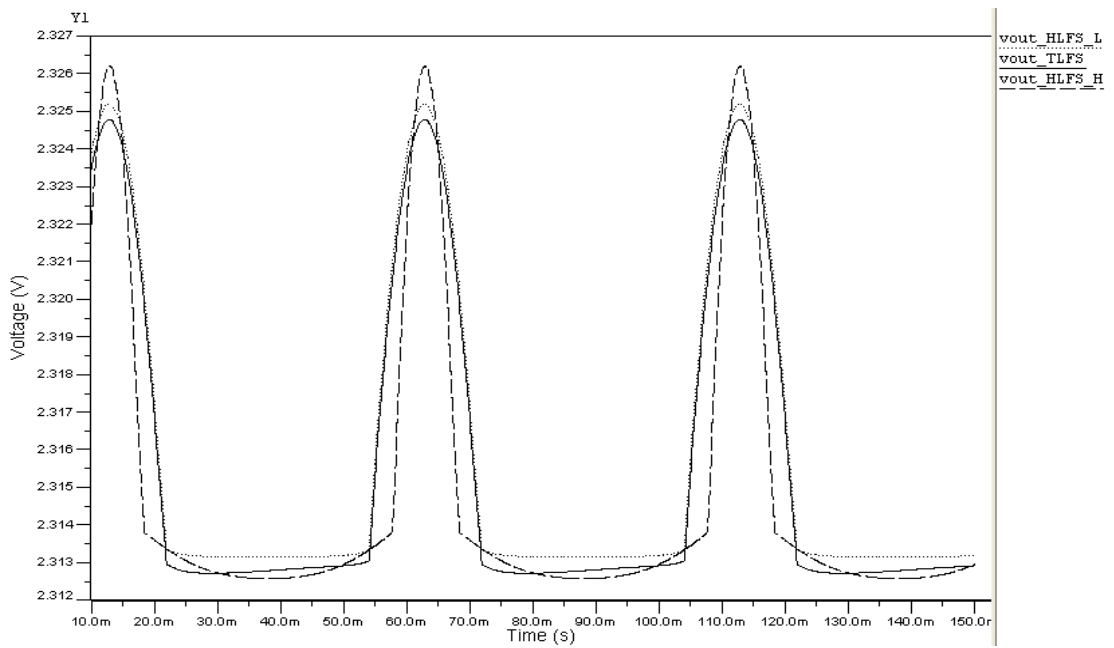
<sup>1</sup> short between gate and drain on transistor 4 at op1

<sup>2</sup> short between drain and source on transistor 8 at op2



**Figure 9-10:** HLFM for *M11\_dss\_1*

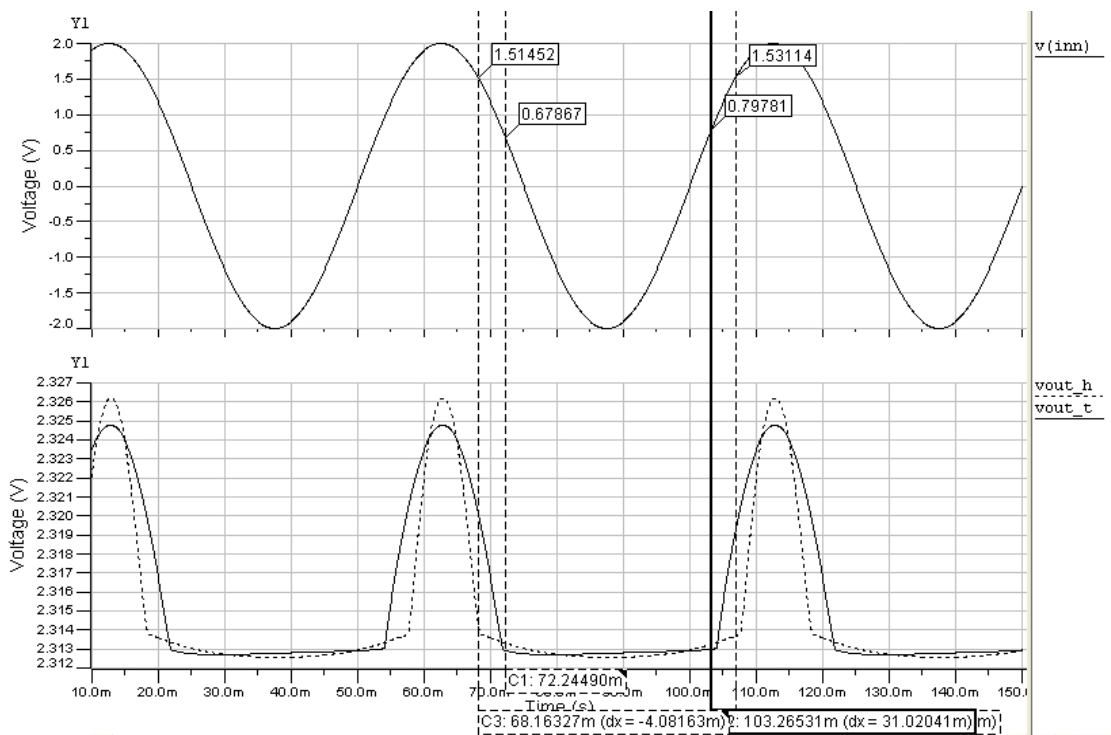
However, the linear macromodel may achieve better quality than our behavioural model when *M10\_gss\_2* is modelled, output signals are plotted in Figure 9-11. Where *vout\_HLFS\_L* represents the output signal from HLFS based on the model from [Bartsch99], *vout\_TLFS* is the output signal from TLFS and *vout\_HLFS\_H* is the output signal from HLFS using our model.



**Figure 9-11:** HLFM for *M10\_gss\_2*

It is seen that  $v_{out\_TLFS}$  contains extreme nonlinearity,  $v_{out\_HLFS\_L}$  gives the saturation part because  $op3$  is still the nonlinear transistor level model. The MMGSD model can not follow this nonlinear part accurately, especially the flat part.

To find out the reason we observe the input and the output signals of our model seen in Figure 9-12 because they are ones that are used to generate right models and thresholds.  $v(inn)$  is the input stimulus,  $v_{out\_h}$  and  $v_{out\_t}$  are output voltages from TLFS and HLFS, respectively. Several vertical lines are displayed to show where the thresholds and the extreme nonlinearities are and corresponding inputs.



**Figure 9-12:** Investigation which model is applied in relation to input and output

It is seen that the sharp corner (nonlinearity) at the output of TLFS  $v_{out\_t}$  corresponds to the input at 0.67867V and 0.79781V. Whereas the output from HLFS  $v_{out\_h}$  shows that the nonlinearity is at the input of 1.514V and 1.53114V.

If we check the thresholds for the models seen in Figure 9-13, we find that one of them is at 1.5V. It indicates that the model thresholds are in the wrong place.

```

threshold=[-2.5 -1.5 -0.5 0.5 1.5 2.5]
count = 2263 2010 2267 2452 3048

```

**Figure 9-13:** Threshold and samples for each model

To prove this we manually change the position of the thresholds to where nonlinearities are (0.67867V), and also the neighbouring ones in order to supply enough samples to these models. They are shown in Figure 9-14.

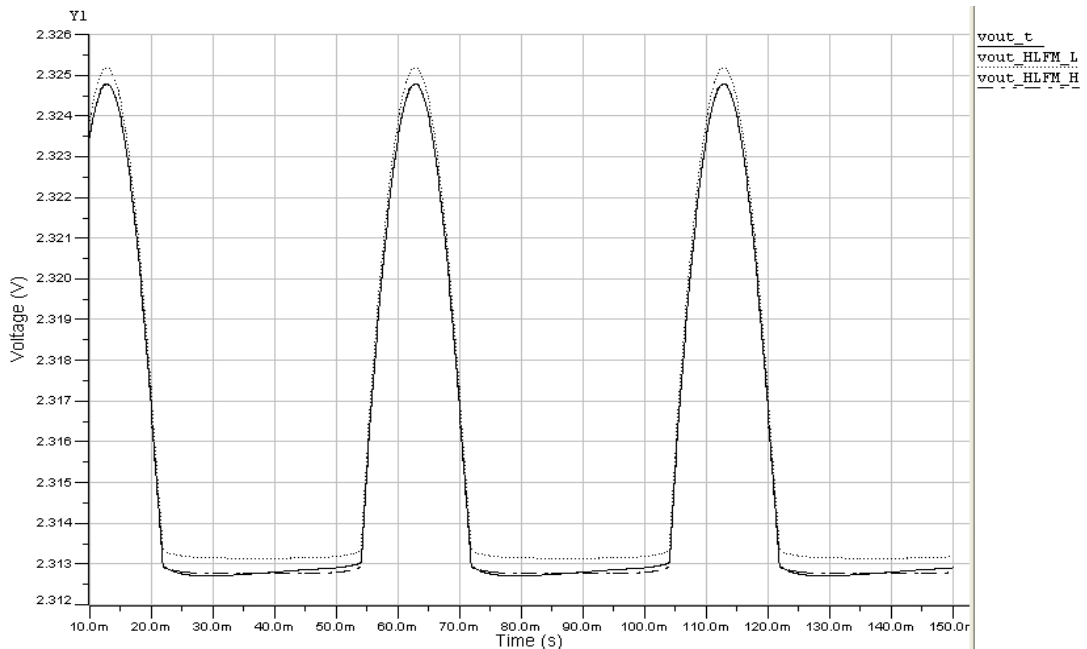
```

threshold=[-2.5 -1.5 -0.9 0.7 1.7 2.5]
count = 2263 1189 3607 2451 2530

```

**Figure 9-14:** New thresholds and samples for each model

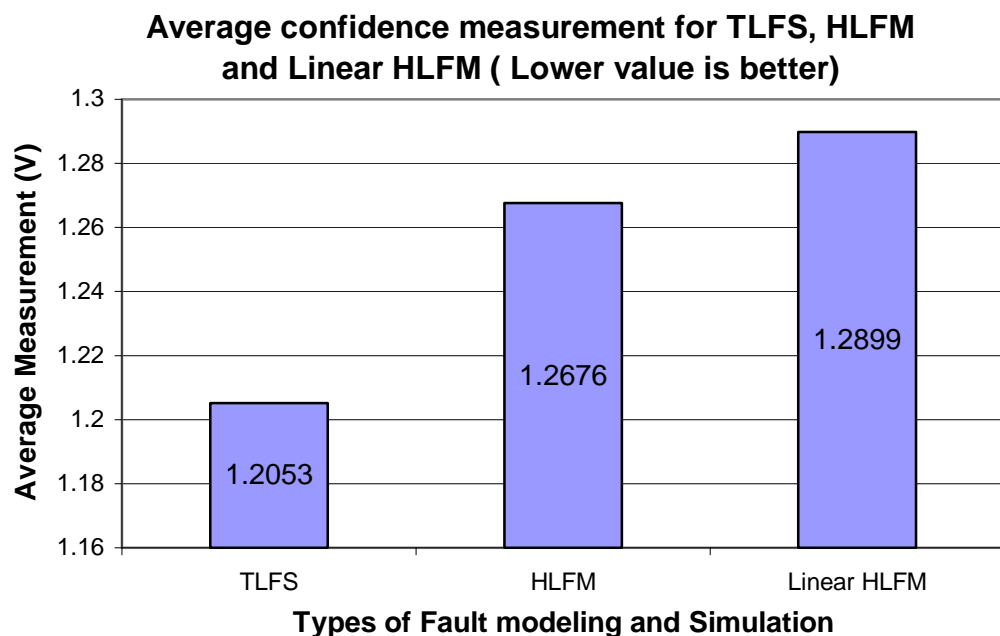
The output voltages from TLFS and HLFM are shown in Figure 9-15 based on the new thresholds. It is seen that the output signal from our model *vout\_HLFM\_L* is reasonably close to the TLFS *vout\_t*, and the quality of our model has been improved compared with Figure 9-11.



**Figure 9-15:** HLFM for *M10\_gss\_2* based on new threshold set

It indicates that the MMGSD is not intelligent enough to pick thresholds where the extreme nonlinearities are. In the future work a more intelligent threshold generator will be developed.

Using ACM in Eq. 9-5, average quality of HLFM and TLFS are measured in Figure 9-16. The aim is to verify that our model performs better than other macromodels and that our model can model faulty behaviour with good accuracy compared with TLFS. It is seen that TLFS achieves the best fault coverage (1.2053V), our models has the fault coverage of 1.2676V, which has shown better quality than the linear macromodel (1.2899V).

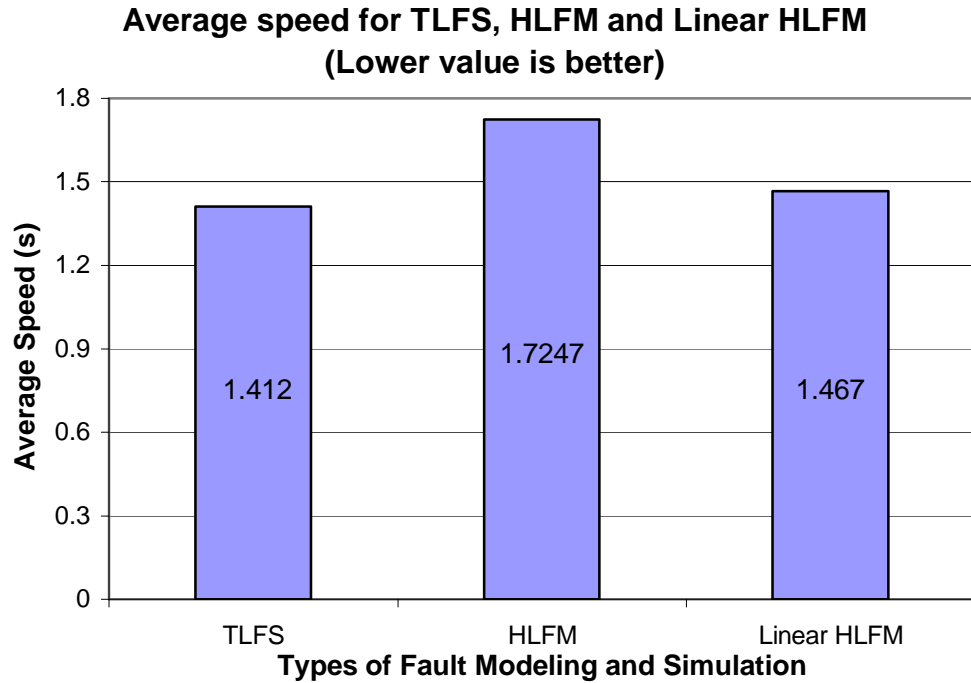


**Figure 9-16:** ACM for TLFS, HLFM and Linear HLFM

Furthermore, the simulation time for TLFS and HLFM is required. It is noticed that the highest speed-up is achieved using the linear fault model; for example for *M4\_gss\_1* HLFM needs 1.218s, whereas TLFS requires 1.341s.

The total average time for each simulation is shown in Figure 9-17. It is seen that TLFS has the highest average time of 1.412s, HLFM based on the MMGSD is about 0.3s

slower than TLFS, and HLFM based on the linear fault model has an average time of 1.467s.



**Figure 9-17:** Average speed measurement for TLFS, HLFM and Linear HLFM

In some cases our model needs less simulation time than HLFM based on the linear macromodel, for example, *M10\_dss\_3*, the models from MMGSD takes 1.68s to complete simulation, whereas the linear model requires 1.89s.

According to Eq. 9-7 simulation speed-up is obtained shown in Eq. 9-8:

$$speed\_up = \frac{1.412}{1.7247+0.1} \approx 0.774 \quad \text{Eq. 9-8}$$

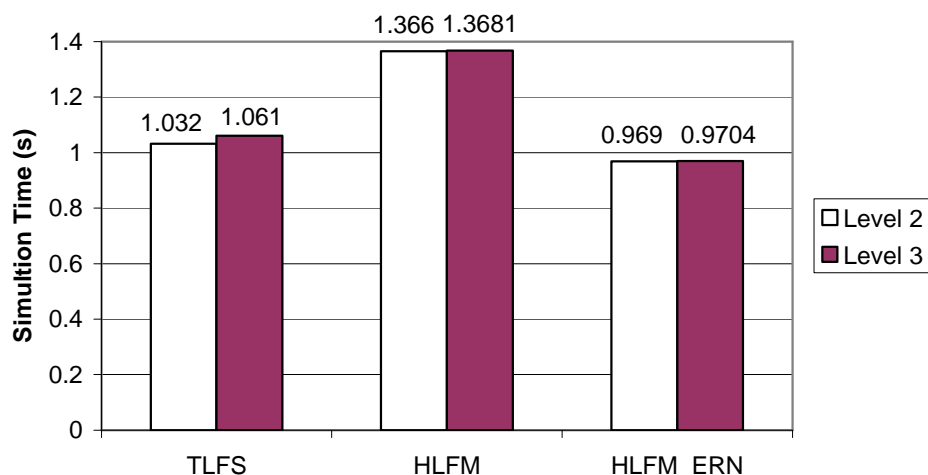
To investigate simulation speed, this time we focus on the SPICE level of the transistor model and prove that the level of transistor model can affect the simulation results. It is known that the transistor that has been used through the thesis is at level 2. In this experiment we use a new transistor, which is a 1.2 micron CMOS model (Level 3 instead of level 2) [Spiegel95]. The same op amp netlist in Figure 4-1 is still employed,

and the fault used is *m7\_dss\_1*, and the circuit used for simulation is the low-pass filter in Figure 9-6. The input stimulus is a 2V sine waveform at 20Hz.

Two types of comparisons are implemented: 1) simulation using the level 3 transistor is run, so we compare results from the TLFS, HLFM and HLFM based on the linear model in terms of simulation speed and accuracy. 2) we compare results based on the two levels of transistors.

Initially HLFM and TLFS based on the level 3 transistor are run. The simulation measurement is based on the section 9.3.2. Accuracy is preserved. Each type of simulation has been run 10 times, the average value is then chosen in order to reduce affects of interaction in computer shown in Figure 9-18. The fastest simulation is from HLFS using the linear model (0.9704s). Simulation speed from HLFS based on the MMGSD model (1.3681s) is slower than TLFS (1.061s). This is because the simulator SystemVision is not optimised to the structure of our models, but optimised to the structure of linear behavioural model and proves that the behavioural model is able to run faster than TLFS. It indicates that changing the level of transistors in the op amp does not improve speed of HLFM based on the MMGSD model.

We then compare simulation speed between the level 3 transistor and the level 2 transistor. Illustrative results are shown in Figure 9-18.



**Figure 9-18:** Simulation Speed Comparison between level 2 transistors and level 3 transistors

It is seen that during TLFS simulation speed based on the level 3 transistor (1.061s) is slower than the one using the level 2 transistor (1.032s). This is because the former is more complex than the latter.

Although the difference between TLFS time is not very significant, it has indicated that as the transistor is getting more complex (parameter level in the transistor is higher), TLFS becomes slower, whereas simulation time from HLFM almost does not change. Therefore, simulation speed-up during HLFM may be shown.

In the future work, more complex CMOS transistors (e.g., IBM 0.13 micron level 49 [Mosis]) will be employed to investigate the improvement of simulation speed.

## ***9.5 Conclusion***

In this chapter both VHDL-AMS HLM and HLFM are implemented using a behavioural model generated by the MMCSD. The ACM and average time measurement are developed to evaluate HLFM. We have demonstrated that our behavioural model is not only able to model linear but also nonlinear behaviour with good accuracy for HLM. Using the ACM we have proved that our model provides better quality than a fault macromodel [Bartsch99]. During simulation it is found that our system can not intelligently find the right thresholds to handle the extreme nonlinearity, which will be investigated in the future. Speed-up is not achieved compared with TLFS. This is because the simulator has not been optimised to deal with this kind of approach, and so the computational overhead is high. Moreover, speed-up can be improved by feeding the neighbouring models instead of all models during model selection process.

In the next chapter the process used in this chapter will be repeated but with in a more complex system in order to investigate simulation speed.



# ***Chapter 10: High Level Fault Simulation of a 3bit Flash Analogue to Digital Converter***

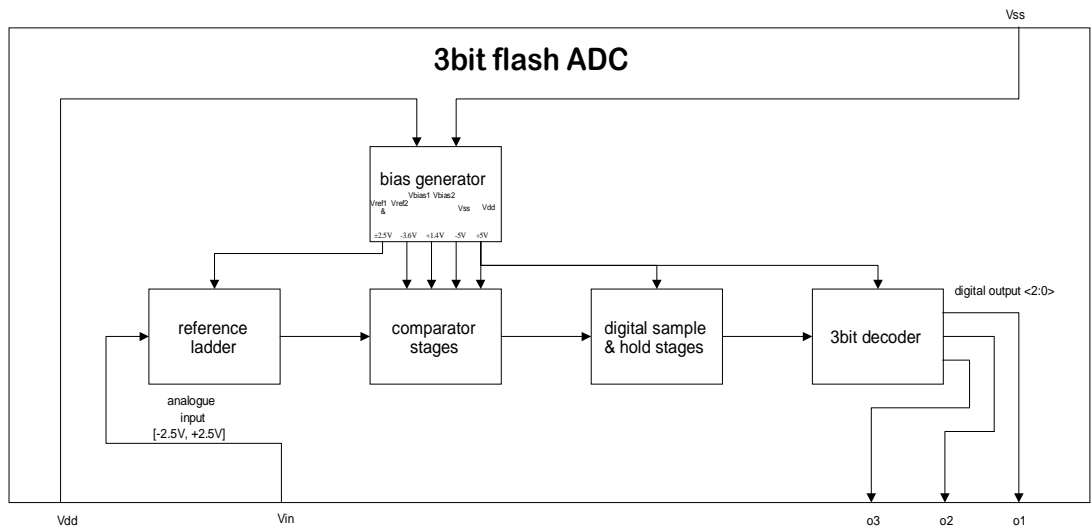
## ***10.1 Introduction***

In this chapter a more complex circuit is subjected to HLFM in order to investigate if our behavioural model based on the MMGSD is able to achieve better quality (accuracy and speed-up) compared with a TLFS. This is demonstrated by a 3bit flash analogue to digital converter (ADC) realised in CMOS technology under the simulator SystemVision [SystemVision]. Section 10.2 introduces the architecture and the design of the 3bit flash ADC. Models based on the MMGSD are generated in section 10.3. In section 10.4 the TLFS and HLFM are compared based on this ADC. The conclusion is drawn in section 10.5.

## ***10.2 Introduction to the 3bit Flash ADC***

The flash analogue to digital conversion concept is mainly used in telecommunication, high speed signal processing (e.g. video) and radar. A flash ADC converts the analogue input signal into digital code bits in one step. All other types of ADC such as successive approximation, semi-flash, sigma-delta need more than one step and therefore the main advantage of a flash ADC is its speed. There are  $2^n - 1$  reference voltages and comparator stages for a n-bit flash ADC. The reference voltages are usually generated with a voltage divider (reference ladder) and consequently  $2^n$  resistors are required. As a result flash ADCs with high resolution require a huge chip area. Therefore flash ADCs are generally used for analogue to digital conversion with low resolution (2-8 bits). The high power dissipation is also a drawback for flash ADCs. Both chip area and power dissipation increase linearly with the number of comparators and d-flip-flops (sample & hold stage) and thus exponentially with the number of bits. Similarly the input capacitance of the ADC increases exponentially with the number of bits.

The block diagram of the 3bit flash ADC is shown in Figure 10-1 [Bartsch99]. It converts an analogue input voltage from -2.5V to 2.5V to the corresponding binary code with a resolution of 3 bits. It consists of five functional blocks: the reference ladder, the bias generator, the comparator stages, the digital sample and hold stages and the 3bit decoder.



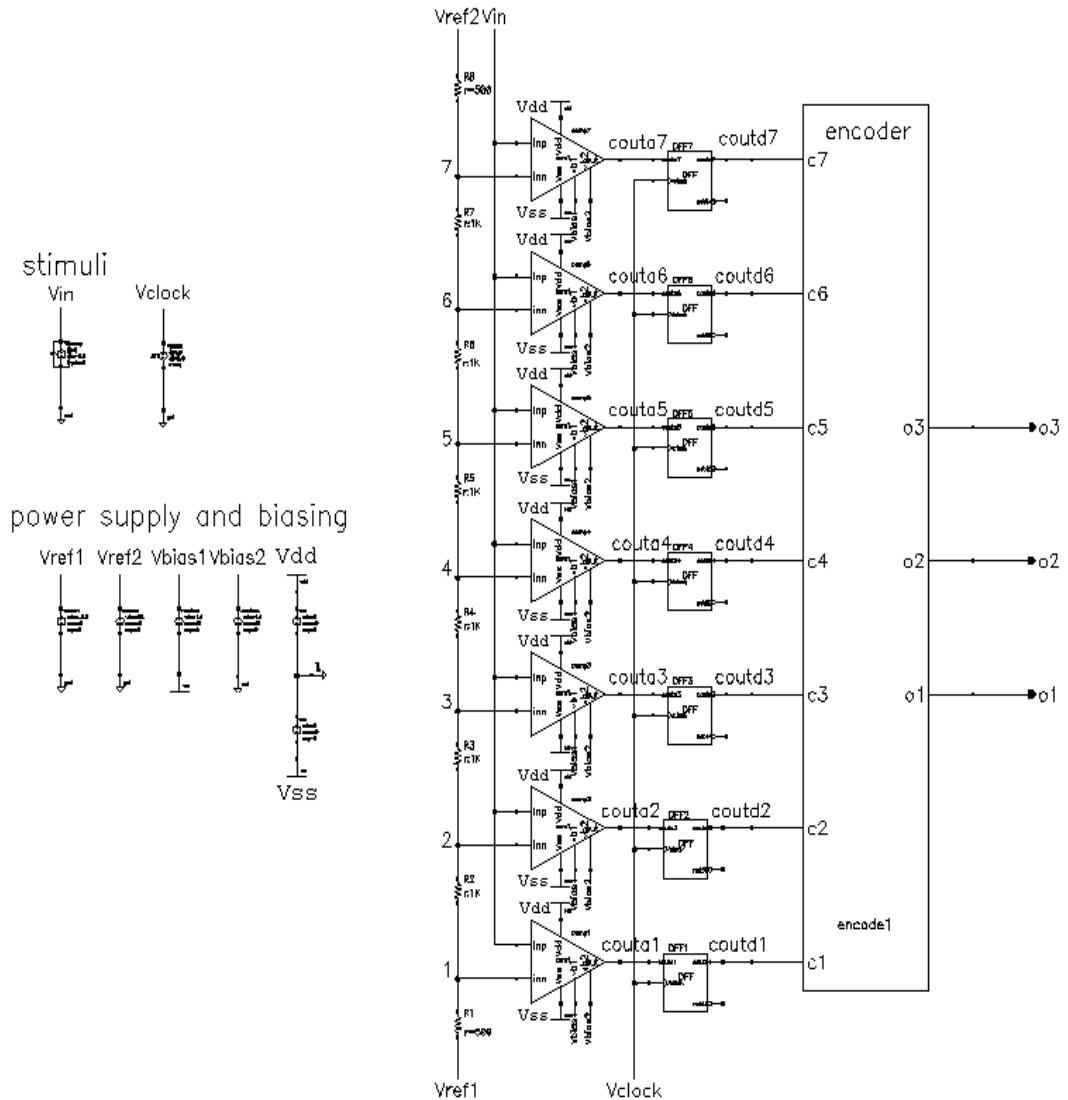
**Figure 10-1:** Block diagram of the 3bit flash ADC [Bartsch99]

The schematic of this 3bit flash ADC is shown in Figure 10-2. The reference ladder comprises 8 resistors and can generate 7 reference voltages. These resistors all have the same values except for the first and the last resistors, which are exactly half of others. In this case these two are set to 500Ω for the rest of them have the value of 1kΩ. This is a good compromise between chip area and power dissipation. The ADC is based on the one used in [Bartsch99].

## 3-bit flashADC

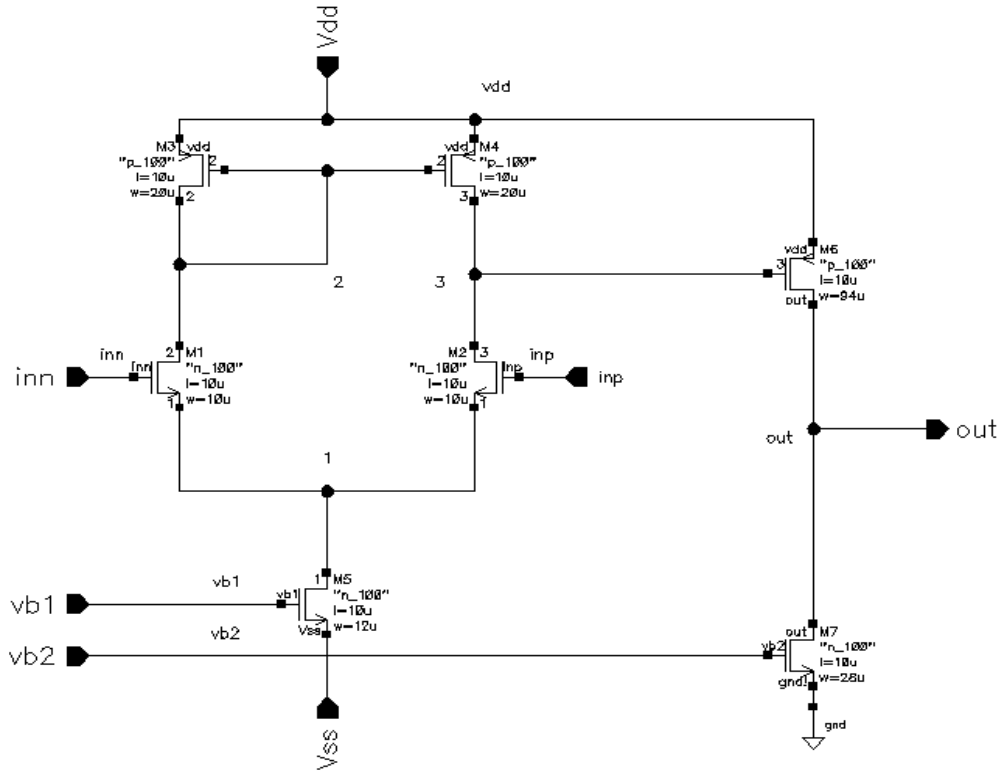
analogue part

digital part



**Figure 10-2:** Schematic of the 3bit flash ADC

The comparator is designed in CMOS technology, see in Figure 10-3. It has an input voltage range of  $\pm 2.5\text{V}$  and an output voltage swing of  $5\text{V}$  ( $0\text{V}$  to  $+5\text{V}$ ). Each comparator comprises an input stage and an output stage. The input stage is realised as a CMOS differential amplifier using n-channel MOSFETs. The differential amplifier is biased with a current mirror M3&M4. M5 is a current source. The output stage (M6 and M7) behaves as a voltage shifter to ensure the output voltage is always positive, M7 is also a current source.



**Figure 10-3:** The CMOS comparator

The comparator stages compare the analogue input voltage with the 7 reference voltages. The output voltages of the comparator stages form a so called thermometer code (Table 10-1).

comparator output voltages (thermometer code) <sup>5</sup>							digital output of the ADC (binary code)			decimal
c7	c6	c5	c4	c3	c2	c1	o3	o2	o1	d
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	1	1
0	0	0	0	0	1	1	0	1	0	2
0	0	0	0	1	1	1	0	1	1	3
0	0	0	1	1	1	1	1	0	0	4
0	0	1	1	1	1	1	1	0	1	5
0	1	1	1	1	1	1	1	1	0	6
1	1	1	1	1	1	1	1	1	1	7

**Table 10-1:** Truth table for the 3bit flash ADC

This thermometer code is then digitally sampled with 7 D-flip-flops. The digital sample and hold stage is necessary to avoid temporary wrong binary code words. Usually the result of the analogue to digital conversion is stored in a memory and so there is always a certain probability that a temporary wrong code is stored. The D-flip-flops can

guarantee that this situation can be avoided when the input signals of the decoder stays constant over the whole conversion period. Moreover, with digital sample and hold stage a very fast analogue to digital conversion can be realised, whereas only frequencies of a few megahertz can be reached with enormous effort with analogue sample and hold stages [Tietze93].

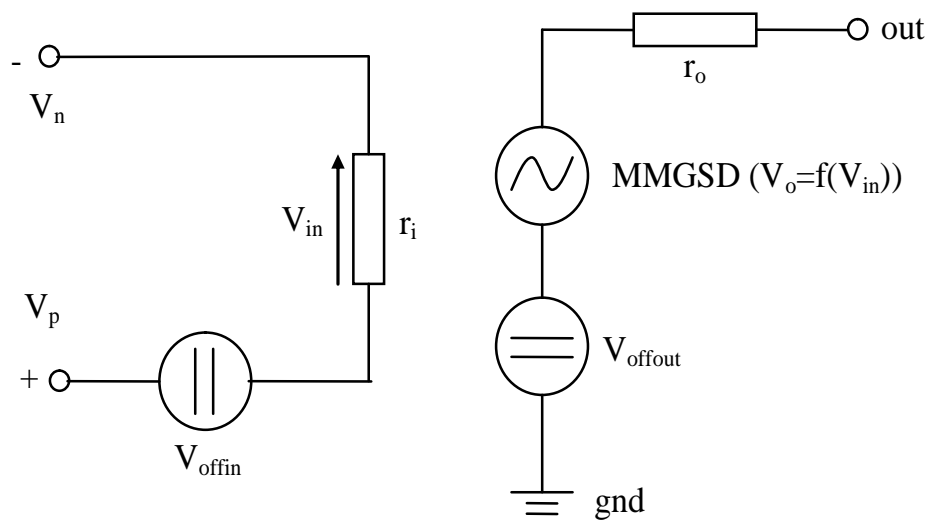
### 10.3 Multiple Model Generation by the Simulator and MMGSD

#### 10.3.1 Sample & Hold and Decoder Model

In SystemVision there is a built-in VHDL-AMS library, the digital part of this flash ADC (the 3bit decoder and D-flip-flop) is implemented and mixed with the analogue part on a schematic platform to perform simulation.

#### 10.3.2 Comparator Model

The behavioural model shown in Figure 9-1 (repeated in Figure 10-4) is used to model the comparator, in which the generated models from the MMGSD are used to handle nonlinearity. This behavioural model is created by the multiple model conversion system in delta transform (MMCSO) discussed in the previous chapter.



**Figure 10-4:** Architecture of the comparator model

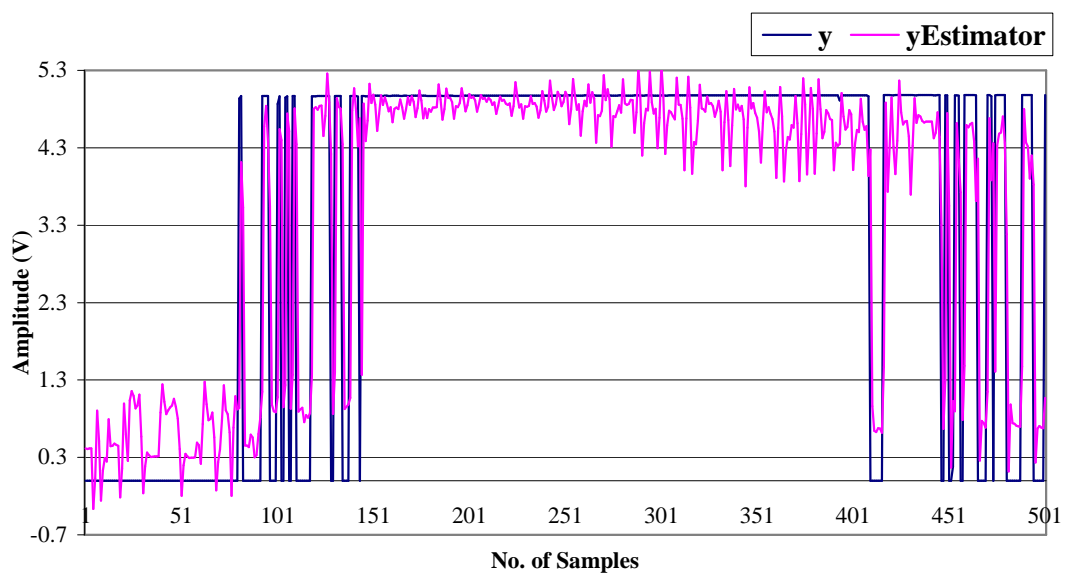
The CMOS comparator in open-loop is analysed by the MMGSD to generate multiple models using same PRBS training data in Figure 9-5. One such signal is applied at the inverting terminal and another is connected to the non-inverting terminal. Five models

are generated. Thresholds and the number of samples for each model are shown in Figure 10-5.

```
threshold=[-2.5 -1.5 -0.5 0.5 1.5 2.5]
count =      3412 2499 2583 2705 3801
```

**Figure 10-5:** Threshold and samples for each model

The estimated signal is seen in Figure 10-6, only the last 500 samples are displayed.

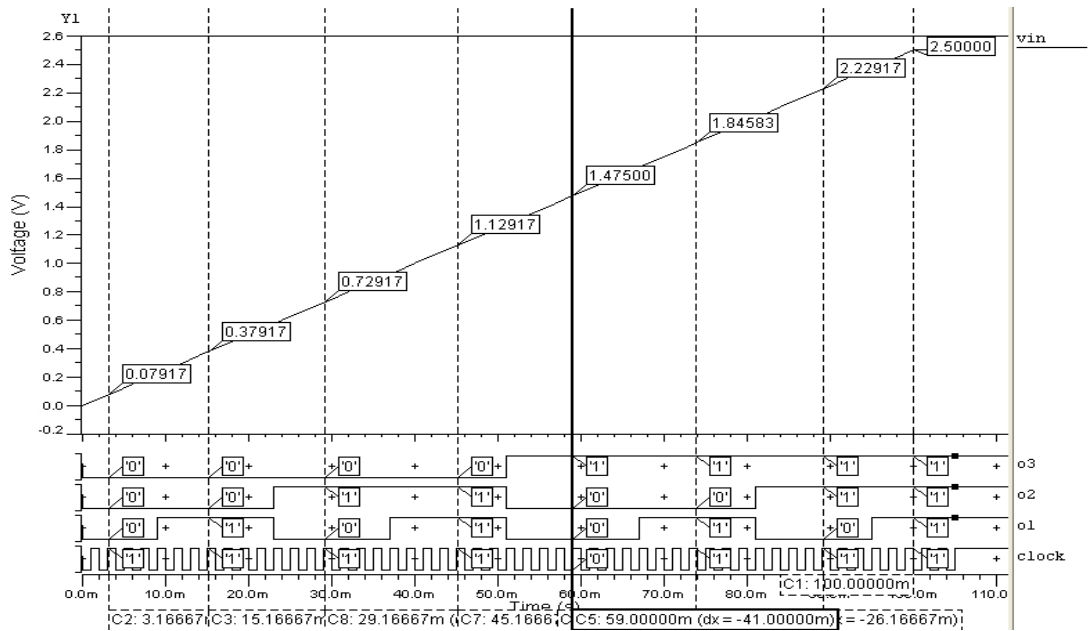


**Figure 10-6:** The estimated signal

Average difference between the estimated signal *yEstimator* from the MMGSD and original output signal *y* is 9.2672%, which is measured by the same equation used before and seen in Eq. 10-1. These models are then inserted in the behavioural model in Figure 10-4.

$$Average\_dif = \frac{\sum_{i=1}^N |y(i) - y_p(i)|}{y\_peak\_to\_peak} \times 100 \quad \text{Eq. 10-1}$$

The nominal operation of the 3bit flash ADC is plotted in Figure 10-7. A ramp input stimulus covering from 0V to 2.5V is used and the clock frequency is set to 500kHz.



**Figure 10-7:** Nominal operation of the 3bit flash ADC

#### ***10.4 High Level Fault Modelling of the 3bit Flash ADC***

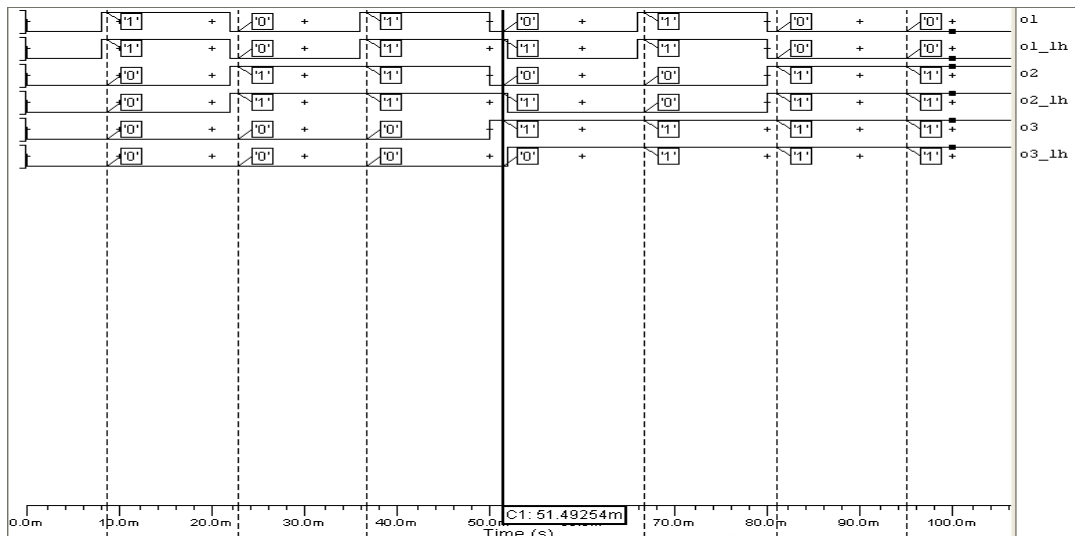
During HLFM and HLFS the comparison procedure using quality and speed as described in previous chapter is employed. Quality measurement focuses on accuracy of digital outputs. Only short faults are simulated at transistor level with a  $1\Omega$  resistor connected between the shorted nodes, which is injected by ANAFINS [Spinks98]. It is known that there are 7 transistors in the comparator, the number of short faults on one transistor is 3 and therefore the number of short faults in this comparator is 21. Since there are 7 comparators in this 3bit flash ADC, a total of 147 short faults are simulated for each model. This MMGSD model and the same fault macromodel in Figure 4-2 will replace every faulty comparator each time, separately, and the rest of system remains the same. The same ramping signal as above is supplied to the non-inverting terminal. A transient analysis is run from 0 to 0.1s with a step of 2ms.

The simulation results indicate that HLFM based on the MMGSD model is able to model all faults correctly compared with TLFS, but HLFM based on the fault macromodel can not accurately model some faults including  $M1\_dss\_1^1$ ,  $M2\_gss\_2$ ,

---

<sup>1</sup> Short between drain and source on transistor 1 at 1<sup>st</sup> comparator

$M5\_gss\_1$ ,  $M5\_gss\_7^2$ ,  $M6\_gss\_1$ ,  $M6\_gss\_2$ ,  $M7\_dss\_1$ ,  $M7\_gss\_1$  because this macromodel is unable to handle high nonlinearity. One example of modelling failure, for  $M6\_gss\_1$ , is plotted in Figure 10-8, where  $o1$ ,  $o2$  and  $o3$  are the output bits from TLFS, and  $o1\_lh$ ,  $o2\_lh$  and  $o3\_lh$  are for HLFM. It is seen that there is bit mismatched at between 50ms and 52ms.



**Figure 10-8:** Failure of modelling in  $M6\_gss\_1$

Simulation speed is also measured. In 146 out of 147 times TLFS takes less time than HLFM, but  $M4\_dss\_7$  our behavioural model has same simulation time as TLFS (0.5s). In 15 out of 147 times the linear macromodel can achieve fastest simulation time.

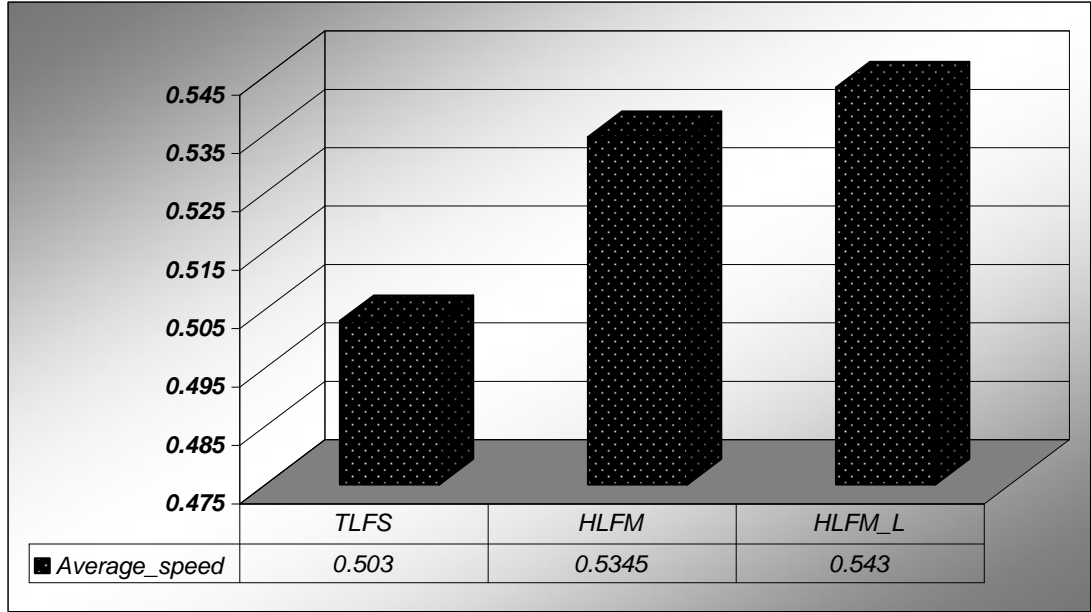
The average time for each simulation is calculated using Eq. 10-2, which has been previously defined in Eq. 9-9, where  $Ave\_time$  is the average time for individual fault simulation;  $NS$  indicates the number of simulation;  $CPU[i]$  represents the cpu time at the  $i^{th}$  fault during simulation.

$$Ave\_time = \frac{\sum_{i=0}^{NS-1} CPU[i]}{NS} \quad \text{Eq. 10-2}$$

<sup>2</sup> Short between gate and source on transistor 5 at 7<sup>th</sup> comparator



Speed for each simulation is obtained and depicted in Figure 10-9.



**Figure 10-9:** Average time for each simulation

It is seen that TLFS has the fastest average time of 0.503s, HLFM based on the MMGSD (0.5345s) is slower than TLFS due to the low complexity of this comparator, that is, the CMOS comparator consists of only 7 MOSFETs. HLFM based on the linear fault model *HLFM\_L* contains average time of 0.543s, which is slower than our HLFM.

The speed-up is then calculated according to Eq. 10-3, where  $t_{TLFS}$  is transistor level simulation time,  $t_{HLFM}$  is high level modelling time,  $t_{op}$  is operating point analysis time at transistor level (100ms). Substituting the data into this equation, the speed-up of simulation is obtained in Eq. 10-4.

$$speed\_up = \frac{t_{TLFS}}{t_{HLFM} + t_{op}} \quad \text{Eq. 10-3}$$

$$speed\_up = \frac{0.503}{0.5345 + 0.1} \approx 0.793 \quad \text{Eq. 10-4}$$

Comparing this with the result in Eq. 9-11 using a low-pass filter:  $0.793 > 0.774$ , i.e., speed has been decreased as the circuit used for HLFS and HLFM gets larger. It may indicate speed-up can be achieved when a more complex system is used.

Moreover, during simulation it is observed that some faults have given rise to the same digital outputs, e.g., *M1\_dss\_1* and *M1\_gds\_1*, *M5\_gss\_1* and *M5\_gss\_2*. These faults may be grouped and collapsed, for each fault group only one high-level simulation run of the whole ADC has to be performed. The more complex the design gets the larger the likelihood of fault groups and the number of faults within one group. So simulation time can be saved. However, for such a simple CMOS comparator not many fault groups can be found and relatively fewer faults can be grouped in one fault group.

### ***10.5 Conclusion***

In this chapter the comparison between TLFS and HLFM is continued using a more complex circuit (3bit flash ADC). The MMGSD generates multiple models for a CMOS comparator. These models are inserted in a behavioural model to handle nonlinearity. During HLFM this behavioural model and a fault macromodel replace the faulty comparator in the ADC, separately. Results have shown that the behavioural model can model all faults correctly according to the digital outputs, whereas the linear macromodel fails to model some faults.

A simulation time comparison of the whole ADC shows that no speed up can be gained from HLFM, although in some cases HLFM based on the linear macromodel can achieve fastest simulation time. This is because the event driven simulation in the digital domain is much faster than the Newton Raphson based iteration solving of the circuit equations in the analogue domain. The high level modelling just compensates this. However, according to speed-up equation HLFM of the flash ADC requires less simulation time than a low-pass filter. It indicates that speed-up may be achieved as the circuit used gets more complex.

# ***Chapter 11: Conclusions and Future Work***

## ***11.1 Introduction***

In recent years there has been a large increase in mixed-signal ICs with higher levels of integration. Although research in the digital test domain has provided well established fault models, DFT methodologies and test automation, the same is not true for the analogue test domain. Many of the problems in testing analogue and analogue portions of mixed-signal ICs are described in the literature review.

High level fault modelling (HLFM) has become one of the most important approaches for analogue test due to its high speed. The models can be created either manually or automatically. Automated model generation (AMG) approaches have showed their ability to handle soft or strong nonlinearity. Speed-up can be achieved using model order reduction (MOR) approaches. However, most published AMG approaches have been developed and evaluated in its context of high level fault-free modelling rather than HLFM. This thesis has investigated HLFM using the AMG approach.

## ***11.2 Automated Model Generation Approaches***

In this work two novel automated model generation (AMG) approaches based on the recursive maximum likelihood (RML) were developed for SISO and MISO systems: the multiple model generation system (MMGS) in  $z$  transform and multiple model generation system using delta operator (MMGSD). Both were evaluated using a two-stage CMOS open-loop operational amplifier (op amp), the input stimulus was a PRBS to achieve a wide spectrum. The MMGS can handle low-pass filters, and model nonlinear behaviour with good accuracy. The work has been published at DTIS in 2008 [Xia08a] and ISCAS in 2008 [Xia08b]. The MMGSD can converge twice as fast as the MMGS, handle both low-pass and high-pass filters, and model nonlinear behaviour correctly. This work has been published at WCE in 2008 [Xia08c].

Some key issues need to be considered: during estimation the estimator may not accurately estimate the offset coefficients because it is difficult for the estimator to

obtain enough information since it is a constant value. This has been improved by adding an offset parameter during HLM and HLFM. Furthermore, the MMGSD is not intelligent enough to always pick thresholds exactly where the extreme nonlinearities are because of fixed positions used for the thresholds, so it sometimes may not model nonlinearity accurately. This can be improved by manually adding a threshold on the nonlinear area.

### ***11.3 High Level Fault Modelling***

A VHDL-AMS behavioural model is developed to implement HLM and HLFM for transient analysis. The models from the MMGSD are used to form VCVSs in this model. Short faults were investigated, which are obtained from the fault injector ANAFINS [Spinks04]. The netlists used are a low-pass filter and a 3bit flash ADC. Results show that the model can handle both linear and nonlinear situations with good accuracy in the filter, and model digital outputs in the ADC correctly. Comparing with a published fault model [Bartsch99], better quality has been achieved in terms of output signals using fault coverage measurement [Spinks98].

Although speed-up is not achieved during simulation because the op amp only contains a small number of transistors (11 transistors), it has been proved that as the system is getting larger speed-up can be achieved more easily.

### ***11.4 Future Work***

Based on the findings and conclusions of this work, future work is justified in several areas.

1. A more complex system such as an IV amplifier will be employed for HLFM to investigate simulation speed. The same procedure will be used as the one for the low-pass filter.
2. A flexible threshold creation method for nonlinearity will be investigated.
3. The work here uses a model partitioning system based on a single input, further work will focus on model generation process by observing multiple inputs.

4. The simulator used (SystemVision) in the thesis is not optimised to the structure of the MMGSD model, which results in slow simulation. In the future work another simulator using different iteration method instead of the Newton-Raphson in SystemVision method will take place in order to deal with nonlinear part with less number of iterations.

5. The same technique of the transistor with different levels will be used for simulation speed investigation. Moreover, more update transistors will be used for simulation.

# ***Appendix A: Characterises of the Two-stage Op Amp in HSPICE***

The objective of this section is to provide the knowledge on how to simulate and test a CMOS operational amplifier (op amp). Thus, users who do not have the datasheet can still know about its range of performance when it is integrated in a system. It also gives users more confidence, even though the datasheet is supplied. The methods are suitable not only for SPICE, but also for other types of computer-simulation programs because the simulation and measurement of the CMOS op amp are almost identical and presented simultaneously. In this case the op amp used is the same one in Figure 4-1.

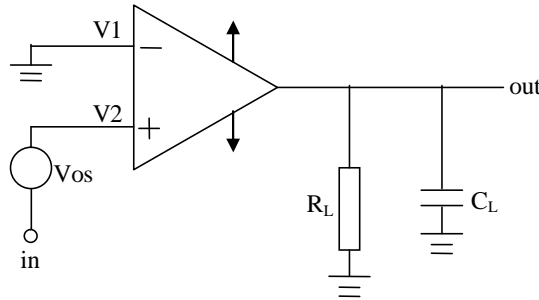
The characteristics of the op amp include: open-loop gain, open-loop frequency response (including the phase margin), input-offset voltage, common-mode gain (CMG), power-supply rejection ratio (PSRR), common-mode input- and output-voltage ranges, open-loop output resistance, and the transient response including slew rate (SR). Its design specification is shown in Table A-1.

Specification	Design
Open loop gain	>20000 = 86dB
GB (MHz)	>1
Input CMR (V)	±2
Slew Rate(V/us)	>2
$P_{diss}$ (uW)	<400
$V_{out}$ range (V)	>4
PSRR <sup>+</sup> (dB)	-
PSRR <sup>-</sup> (dB)	-
Settling Time (us)	-
Output Resistance (Kilohms)	524.2

**Table A-1:** Results between design and simulation

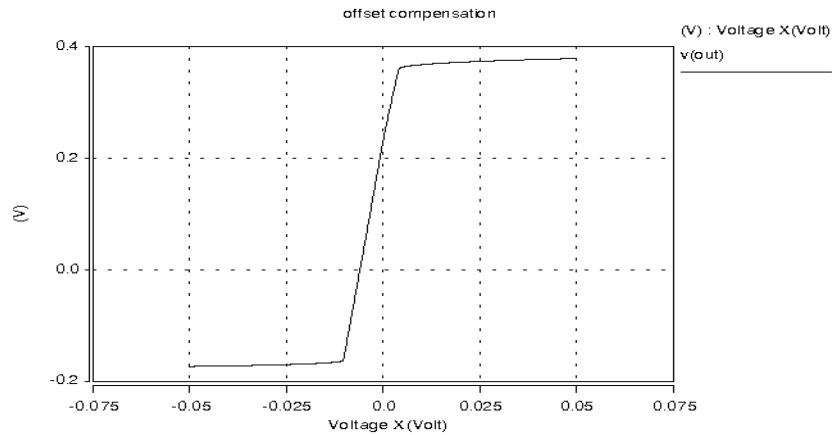
### A.1 Open-loop Mode with the Offset Compensation

Offset voltage is tested with the circuit shown in Figure A-1.  $R_L = 10\text{k}\Omega$ ,  $C_L = 10\text{pF}$ ,  $V_{dd}$  and  $V_{ss}$  are 2.5V and -2.5V, respectively. The input signal is swept between -0.05V and 0.05V with  $100\mu\text{V}$  steps.



**Figure A-1:** Open-loop circuit with the offset compensation

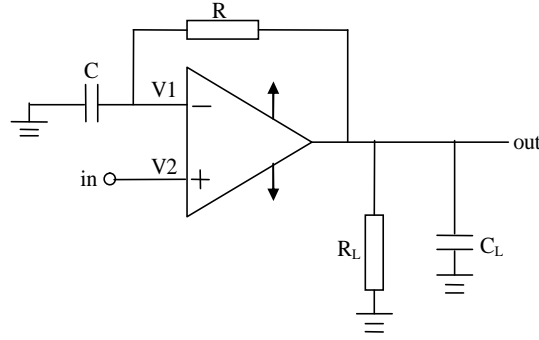
The signal is seen in Figure A-2. It is seen that this circuit has the reasonable offset voltage:  $V_{os} \approx 5.94\text{mV}$  because it should be in the range of millivolts.



**Figure A-2:** The signal for the offset voltage

### A.2 Open-loop Gain Measurement

The open-loop gain is the gain obtained when no feedback is used in the circuit. Ideally it is infinite, but normally it is around  $10^5$ . The method for measuring the open-loop gain is implemented shown in Figure A-3.



**Figure A-3:** A method of measuring open-loop characteristics with dc bias stability

In this circuit it is necessary to select the reciprocal  $RC$  time constant a factor of  $A_v(0)$  times less than the anticipated dominant pole of the op amp, so the op amp has total dc feedback which stabilizes the bias. The dc value of output will be exact the dc value of input. Moreover, the true open-loop frequency characteristics will not be observed until the frequency is approximately  $A_v(0)$  times  $1/RC$ . Above this frequency, the ratio of  $V_{out}$  to  $V_{in}$  is essentially the open-loop gain of the op amp. The anticipated loading at the output is required to obtain meaningful results [Allen87].

The dominant pole is calculated as follows. From the SPICE output file,  $\lambda_9 = 0.03$ ,  $\lambda_{12} = 0.024$ ,  $\lambda_{10} = 0.012$ ,  $\lambda_7 = 0.015$ ,  $k_p' = 17\mu$ . Also,  $I_d = 2.2\mu A$ ,  $I_{d1} = 73\mu A$ ,  $C_c = 1.6pF$ .

$$g_{m10} = \sqrt{2K' I_{d1} \left(\frac{W}{L}\right)} = 93.2\mu A/V \quad \text{Eq. A-1}$$

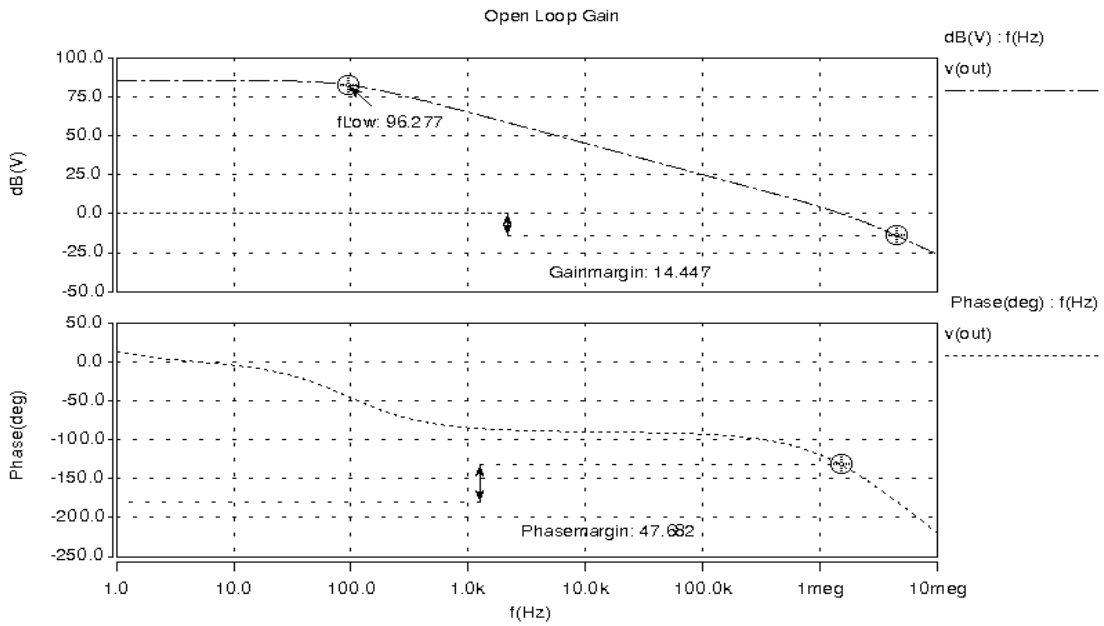
Referring to [Allen87], the first pole is found:

$$P1 = -\frac{(g_{ds9} + g_{ds12})(g_{ds10} + g_{ds7})}{g_{m10} \cdot C_c} = -\frac{(\lambda_9 \cdot I_d + \lambda_{12} \cdot I_d)(\lambda_{10} \cdot I_{d1} + \lambda_7 \cdot I_{d1})}{C_c \cdot g_{m10}} = -1.142kHz \quad \text{Eq. A-2}$$

$$|P1| = 1.142kHz$$

Assuming  $R$  is large enough, otherwise, it will take current from op amp,  $R = 300\text{Meg}\Omega$ ,  $C = 30\mu F$ ,  $R_L = 10\text{Meg}\Omega$ ,  $C_L = 10pF$ . The signal is shown in Figure A-4.



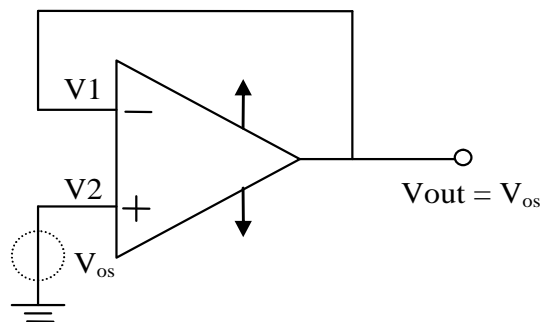


**Figure A-4:** The open-loop gain measurement

It is seen that the open-loop gain is about 85.539dB, i.e., 1.892Meg, and the critical frequency is about 96.277Hz. The gain bandwidth (GB) is the product of the open-loop gain and the critical frequency, which is 1.82MHz, and the phase margin is 47.682. Moreover, power dissipation is 383.1uW (found in the output file).

### A.3 Input Offset Voltage

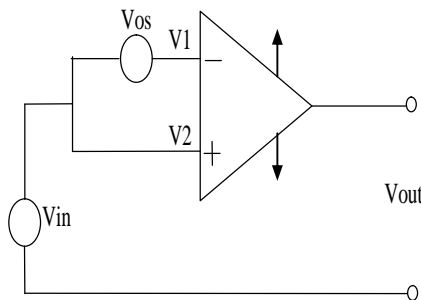
It is known that the input offset voltage is not only due to the bias mismatches, but also due to device and component mismatches. Moreover, it can be affected by time and temperature due to its small value. Therefore, it is difficult to simulate. In this case it is performed by using the circuit of Figure A-5.



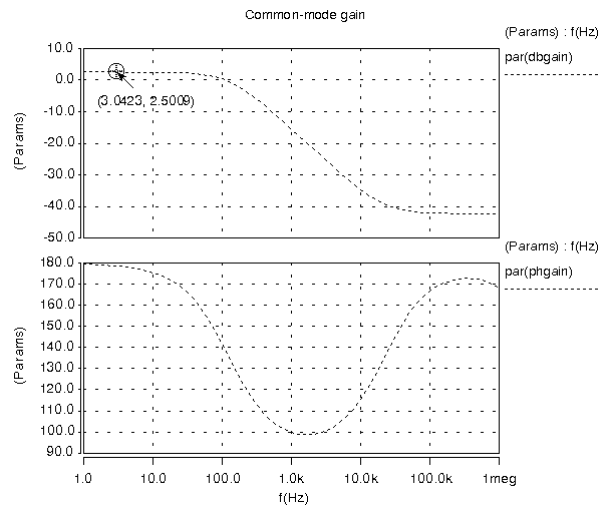
**Figure A-5:** Configuration for measuring the input offset voltage

## A.4 Common-mode gain

The common-mode gain is the ratio of the output voltage to the common-mode input signal. Ideally it is zero, but this is not the case for a real op amp due to the noise. The common-mode gain is most easily simulated or measured using the circuit in Figure A-6-a).



a)



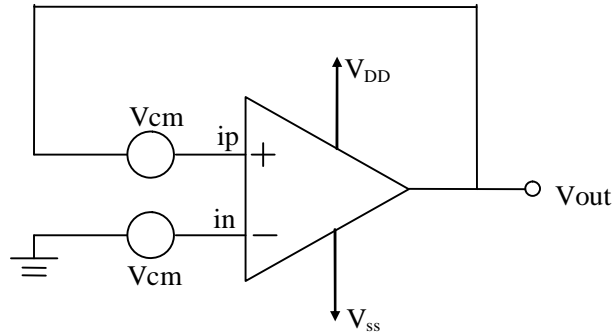
b)

**Figure A-6:** a) Configuration for simulation the common-mode gain, b) Signal for common-mode gain

The input voltage  $V_{in}$  is set to 0.05V, the offset compensation voltage  $V_{os}$  is -5.9365mV, from simulation the signal shown in Figure A-6-b) is obtained. The output voltage gain is about 2.5 dB, i.e.,  $\approx 1.334$ .

## A.5 Common-mode Reject Ratio (CMRR)

The common-mode gain is the ratio of the common-mode voltage at the input of a circuit, to the corresponding voltage at the output. The lower the CMRR, the larger the effect on the output signal [Allen87]. The circuit used for the measurement is shown in Figure A-7.

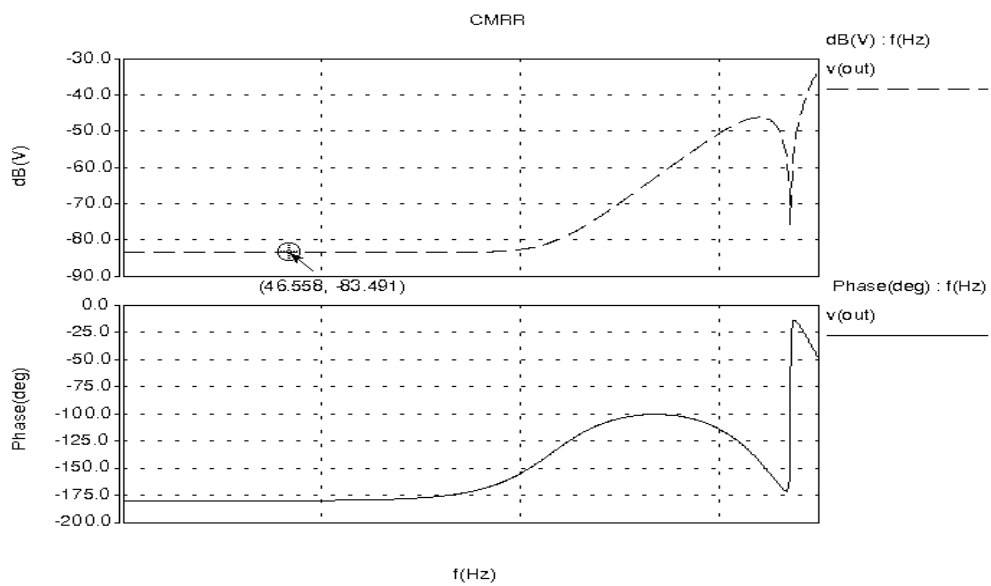


**Figure A-7:** Configuration for direct measurement of CMRR

Initially we measure the common mode gain and then use Eq. A-3 to obtain CMRR, where  $A_v$  is the open-loop gain.

$$\left( \frac{V_{out}}{V_{cm}} \right) = \frac{|A_c|}{A_v} = \frac{1}{CMRR} \quad \text{Eq. A-3}$$

From simulation the output response shown in Figure A-8 is obtained.



**Figure A-8:** CMRR frequency response of magnitude and phase

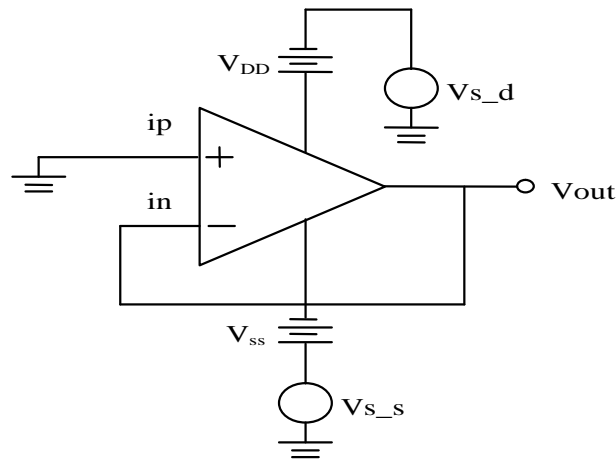
It is seen that the magnitude of common mode gain is -83.491dB, according to Eq. A-3, the  $CMRR = 2.048\text{dB}$ . The phase is shifted about 180 degree.

## A.6 Power Supply Reject Ratio (PSRR)

The power supply rejection ratio (PSRR) is a measure of how well the device rejects noise on the power supply line. Normally it is separated into rejection ratios for the positive power supply  $PSRR^-$  and negative power supply  $PSRR^+$ . Eq. A-4 shows the relationship between PSRR and supply voltage.

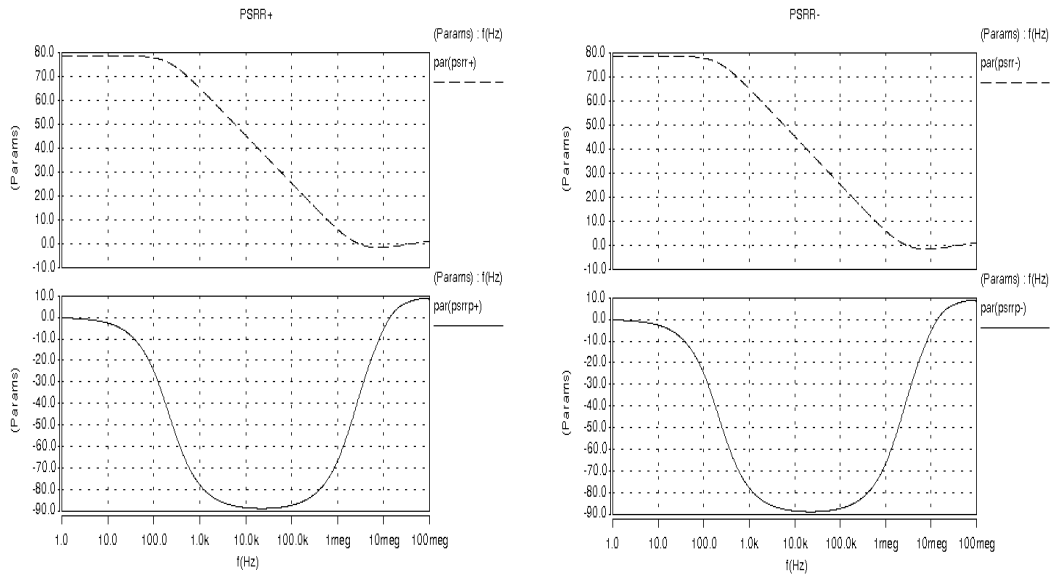
$$\left(\frac{V_{out}}{V_{dd}}\right) = \frac{1}{PSRR^+} \text{ or } \left(\frac{V_{out}}{V_{ss}}\right) = \frac{1}{PSRR^-} \quad \text{Eq. A-4}$$

A circuit from [Allen87] is used to measure PSRR shown in Figure A-9. A sinusoidal with amplitude 0.1mV and frequency 1kHz is used. It is inserted in series with  $V_{DD}$  and  $V_{SS}$  to measure  $PSRR^+$  and  $PSRR^-$ , respectively.



**Figure A-9:** Configuration for direct measurement of PSRR

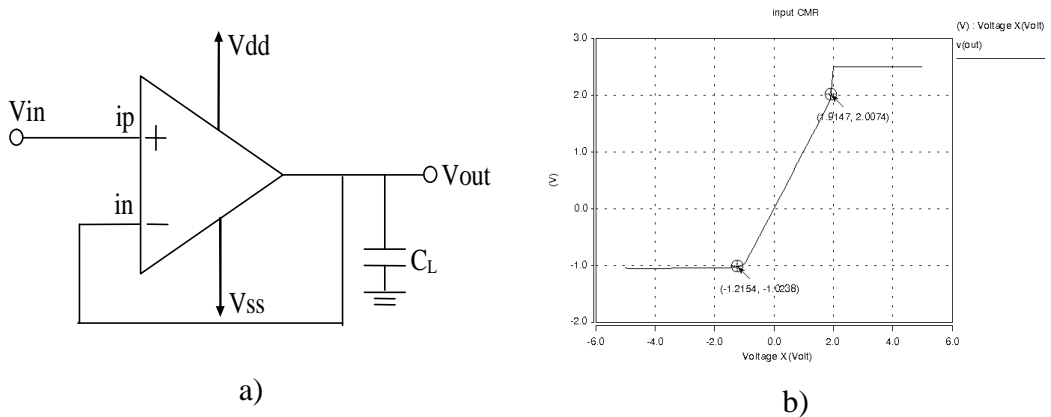
The signals for  $PSRR^+$  and  $PSRR^-$  from the simulation are shown in Figure A-10. It is seen that they are almost the same: about 80dB.



**Figure A-10:** Signals for  $PSRR^+$  and  $PSRR^-$

### A.7 Configuration of Unit Gain for Input and Output CMR

Typically, the output common-mode voltage range (CMR) is about half the power-supply range in open-loop [Allen87]. However, op amps are normally used in close-loop, so it makes more sense that both input and output CMR should be measured and simulated. Unit gain configuration shown in Figure A-11-a) is efficient for this measurement.

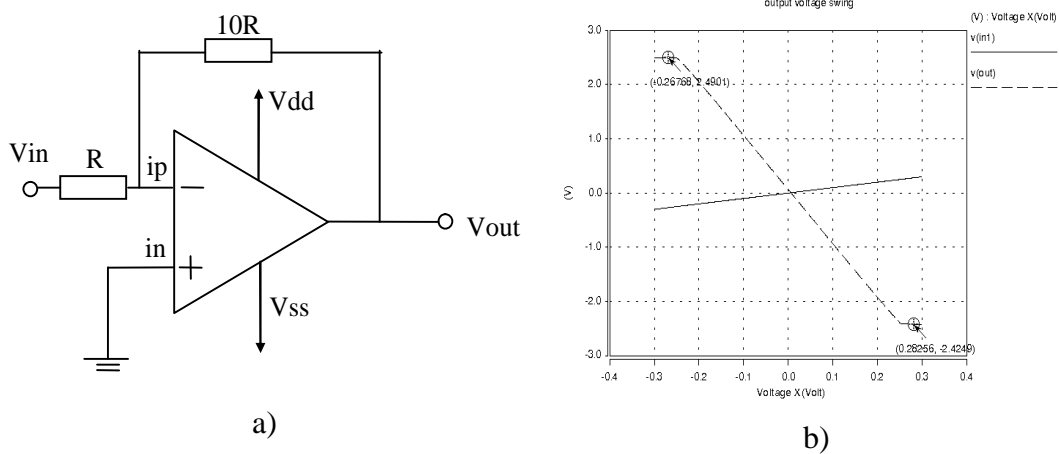


**Figure A-11:** a) Unit-gain for input CMR and b) the signal

DC analysis is performed from -5V to 5V with steps of 0.1V, the output signal is shown in Figure A-11-b). The linear part, where the slope is unity, corresponds to the input common-mode voltage range. It quickly increases at about 2V because the value of the compensation capacitor  $C_c$  is small, the voltage  $V_c$  essentially follows the voltage at  $in$ . During the rising edge of the signal, the charging current to  $C_c$  can be much bigger than

the current around output stage, which results in a bigger charge current to  $C_c$ . Therefore a rapid rise of the output voltage is caused. It is seen that the range of input CMR is  $-1.2\text{V}$  to  $+2\text{V}$  with dc supply voltages of  $\pm 2.5\text{ V}$ .

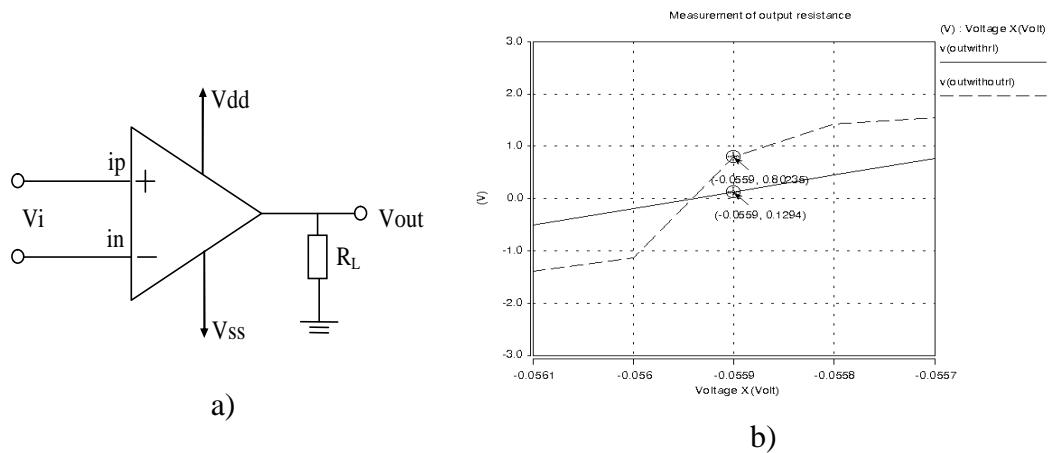
The range of the output voltage can be measured with an inverting amplifier seen in Figure A-12-a), the feedback resistance is ten times bigger than the input resistance, i.e.,  $K_v = -10$ . The signal is shown in Figure A-12-b). The output voltage range is between  $-2.429\text{V}$  and  $2.49\text{V}$ . Moreover, the linear part of transfer curve corresponds to the output-voltage swing.



**Figure A-12:** Signal for the input CMR a) and the signal b)

### A.8 The Output Resistance

In this section, the output resistance is calculated with the circuitry shown in Figure A-13-a). The dc voltage sources are  $\pm 0.05\text{ V}$ ,  $R_L = 100\text{k}\Omega$ .



**Figure A-13:** a) Measurement of the output resistance, b) output signal

The output voltage drop caused by load resistance  $R_L$  is obtained from Figure A-13-b. The relationship between the input voltages and output resistance is given in Eq. A-5, where  $V_{withoutRL}$  is the voltage without measuring the load resistor  $R_L$ , and  $V_{withRL}$  is the voltage when  $R_L$  is included.

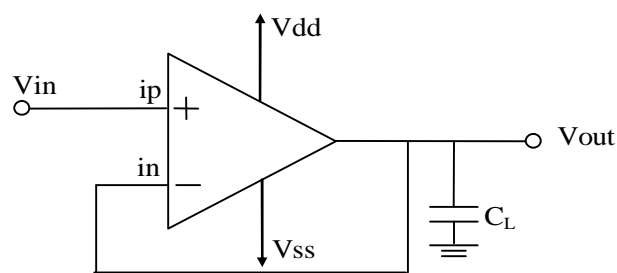
$$R_{out} = R_L \left( \frac{V_{withoutRL}}{V_{withRL}} - 1 \right) \quad \text{Eq. A-5}$$

Therefore, the output resistance is  $R_{out} \approx 520.054\text{k}\Omega$ .

### A.9 The Slew Rate and Settling Time

The slew rate (SR) is the maximum rate of change of signal at the amplifier output. In general, the settling time is defined as the time that it takes the system to settle within a certain value (tolerance say 1%) when it is stimulated. In the amplifier it includes a very brief propagation delay, plus the time required for the output to slew to the vicinity of the final value, recover from the overload condition associated with slewing, and finally settle to within the specified error.

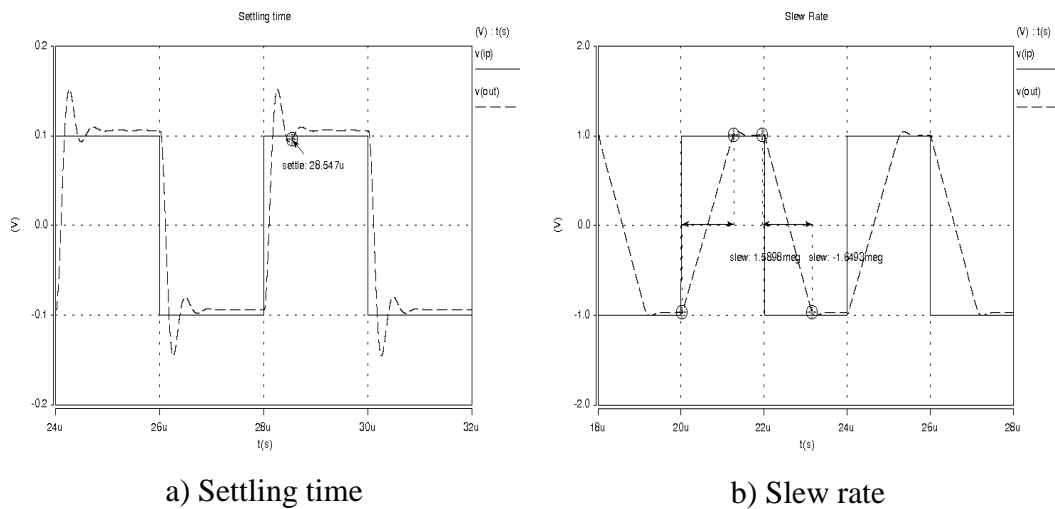
In this case, the same circuitry as Figure A-11-a) is used, as shown in Figure A-14 to measure the slew rate and settling time. The load capacitance  $C_L$  is set to 10pF.



**Figure A-14:** Measurement of the slew rate and settling time

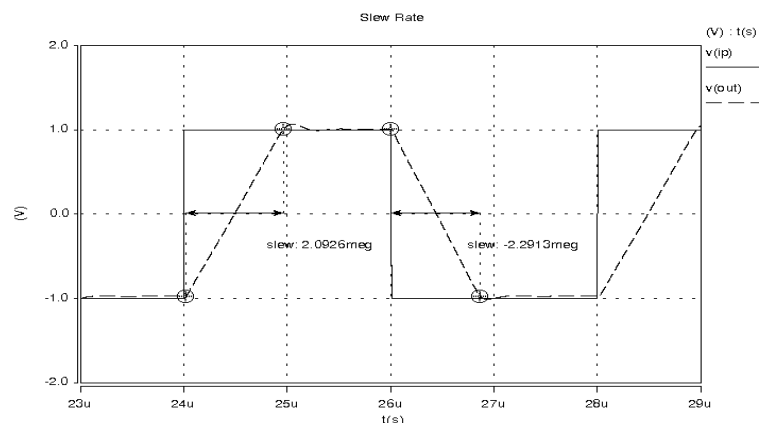
The results of settling time measurement is shown in Figure A-15-a), which is obtained by using the small signal transient response with a 0.2V pulse to the circuit above. The relatively large value of compensation capacitor  $C_c$  prevents the 10pF load from causing

significant ringing in the transient response. It is seen that the settling time to within  $\pm 5\%$  is about  $28.547 - 28 = 0.547\mu\text{s}$ .



**Figure A-15:** Signals from a) settling time and b) slew rate

The slew rate is then calculated. A pulse input signal with the amplitude of 2V is utilized, the input step has to be sufficiently large, so the op amp is able to slew by virtue of not having enough current to charge or discharge the compensating and/or load capacitances. The simulated output signal is shown in Figure A-15-b). The slew rate is determined by the slope of the output waveform during the rise or fall of the output. Both positive and negative slew rates are approximate 1.65V/us, which are less than the specification. One way to improve it is to reduce the compensating capacitance  $c_c$  to 1.8pF. The signal is shown in Figure A-16. It is seen that the slew rate is increased:  $SR+ = 2.2\text{V/us}$ ,  $SR- = -2.43\text{V/us}$ .



**Figure A-16:** Improved slew rate by reducing the compensating capacitance



### ***A.10 Comparison of the Simulation with Specification***

In this section designed values are compared with these results from the simulation in HSPICE illustrated in Table A-2:

Specification	Design	Simulation in SPICE
Open loop gain	>20000 = 86dB	85.539dB
GB (MHz)	>1	1.82
Input CMR (V)	$\pm 2$	+2, -1.2
Slew Rate(V/us)	>2	+2.2, -2.43
$P_{diss}$ (uW)	<400	383.1
$V_{out}$ range (V)	>4	+2.49, -2.43
PSRR <sup>+</sup> (dB)	-	80
PSRR <sup>-</sup> (dB)	-	80
Settling Time (us)	-	0.547
Output Resistance (Kilohms)	524.2	520.054

**Table A-2:** Results between design and simulation

It is seen that characters of the op amp between the design and simulation have been well matched, although the open loop gain and the input CMR from the simulation are a little less than expected. Therefore, this op amp is designed successfully.

# ***Appendix B: User Guide for MAST Language and Cosmos Simulator in Saber***

## ***B.1 Introduction to Saber Simulator***

In 1987 the *Saber* simulator was developed, which is able to support both a hardware description language (HDL) and single kernel mixed-signal simulation solution [DataSheet93]. It has various optional model libraries that contain thousands of models for designing integrated circuits (ICs) to complete high-power embedded control systems. *Saber* controls simulation through an intuitive, graphical user interface, for example, the interface between MATLAB and *Saber*, and supports all the standard analog simulation analysis – DC operating point such as transfer analyses, transient, AC, noise, distortion and Fourier spectral analysis. Furthermore, the *Saber* and inSpecs family of design analysis products provide the ability to perform Monte Carlo, stress, sensitivity, and parametric analysis.

The *Cosmos* simulator in *Saber* is used to specially support the hardware description language (HDL) – MAST. Unfortunately full information on how to operate *Cosmos* is not provided in *Saberbook* [Saber03], which is one of few documents to introduce this language. In addition although each function in MAST is explained with an example, they are not based on a complete model. Thus it is difficult and tedious for users especially for beginners to understand and learn quickly about the HDL. This appendix introduces the user guide on MAST language and *Cosmos* to bridge the gap between fundamental knowledge and more complex work.

This user guide is divided into four sections: section B.2 introduces structures of the MAST language with a complete example. Section B.3 demonstrates how to manipulate the *Cosmos* simulator to display results. The conclusion is supplied in section B.4.

## ***B.2 Introduction to the MAST***

### **B.2.1 Construction of the MAST Language**

The general form of the MAST language is shown in Figure B-1:

```
Unit definitions
Connection point definitions
Template header
Header declarations
{
    Local declarations
    Parameters section
    Netlist section
    When statements
    Values section
Control section
    Equations section
}
```

**Figure B-1:** The structure of the MAST language

Some of these sections may be optional depending on the requirement of the circuits and users. The MAST language is recognized by the simulator with *.sin* extension. More details can be found in *Saberbook*.

### **B.2.2 The complete program in MAST language**

In this section a full program in MAST for an op amp is supplied as shown in Figure B-2 [Wilson01]. Each line is numbered, so readers are able to follow explanations easily.

- 1) #... A closed loop op amp
- 2) **template** test vout vin1 vin2 = a,m,k
- 3) **electrical** vout, vin1, vin2
- 4)
- 5) #...Operational Amplifier Parameters
- 6) **number** a=1

```

7) #...Fault offset Voltage Parameters
8) number m=0
9) number k=0
10) {
11)     #... local Declarations
12)     var i ic
13)     val v vou,vi,fo,voutcalc
14)     #...Procedural Expressions
15)     values{
16)         #...Terminal Voltages
17)         vou=v(vout)-v(vin2)
18)         vi=v(vin1)-v(vin2)
19)         #...Fault offset voltage
20)         fo=m*vi+k
21)         voutcalc=a*(vi+fo)
22)         #...Supply Voltage Limit
23)         if (voutcalc> 2.5) voutcalc=2.5
24)         if (voutcalc< -2.5) voutcalc=-2.5
25)     }
26)     equations{
27)         #...Fundamental Equations
28)         i(vout->vin2) +=ic
29)         vou=voutcalc
30)     }
31) }

```

**Figure B-2:** The program in MAST language

### **B.2.3 Explanation**

These explanations are separated into following sections: B.2.3.1 Comment section; B.2.3.2 Template section; B.2.3.3 Declaration section; B.2.3.4 Values section and Equations section in B.2.3.5.

#### ***B.2.3.1 Comment Section***

```

1) #...A closed loop op amp
5) #...Operational Amplifier Parameters

```

```
7) #...Fault offset Voltage Parameters
11) #...local Declarations
...
```

Comment line is recognized by a hash sign (#) as running to the end of the line. This is same as (//) in the C program, or (--) in VHDL. It starts anywhere within a line and is useful to temporarily remove a line or part of the program during debugging. In this program line 1), 5), 7), 11), 14), 16), 19), 22) and 27) are comments.

### ***B.2.3.2 Template Section***

*template* is a required keyword that identifies the line as a template header, see line 2).

```
2) template test vout vin1 vin2 = a,m,k
```

It is seen that the name of the template is `test`, then connections: `vout vin1 vin2`, and arguments `a,m,k`. Note: it is important to insert the comma between each of the arguments. It is necessary to include the (=) sign to isolate connections from arguments.

There are two types of templates: the standard `template`, which does not have a specified type; the `element template` that uses the keyword `element` for the type. Templates can be in a program and have relationship such as hierarchy, so if one template (template A) contains a reference to another (template B), this indicates, in the model, the system represented by template B is a subsystem of that represented by template A. Designers can create a template that defines a subsystem, and then refer to it in the system template wherever the subsystem is used. It is similar to the function `class` in C++, whose public member of the base class can be inherited by its subclass.

### ***B.2.3.3 Declaration Section***

#### **B.2.3.3.1 Header Declaration**

```
3) electrical vout, vin1, vin2
4)
5) #...Operational Amplifier Parameters
6) number a=1
7) #...Fault offset Voltage Parameters
8) number m=0
```

9) **number** k=0

There are two types of documents: header declaration and local declaration. Similar to application languages such as C, C++, all names (identifiers) must be defined before they are used. The aim of header declarations is to define the system names used in the header. It specifies connection points and argument names, as seen from line 3) to line 9). Connection nodes and argument names are defined after keywords `electrical` and `number`, respectively. `electrical` is used in the analogue part; keywords such as `state logic_4` are employed in the digital part. Connection names can not have a number as the first character unless all remaining characters are numbers, non-alphanumeric characters (such as + or -) are not allowed. Furthermore, between names of arguments, commas are necessary. For example:

Correct: `V10, 2, Vcc94b, 124`

Incorrect: `10V, +15V, 15V1 18megV`

`number` is a keyword for parameters. It indicates both integers and floats. It needs to follow the rule: simple, composite, or arrays of simple/composite. More details can be found in [Saber03].

#### **B.2.3.3.2 Local declaration**

```
11)      #... local Declarations
12)      var i ic
13)      val v vou,vi,fo,voutcalc
14)      #...Procedural Expressions
```

Lines 11) to 14) are the local declaration, which have to be inside `template`. It comprises declarations for all identifiers used inside the template such as `vars`, `ref`, `vals`, `states`. The general syntax is: `keyword unit nameOfParameter`. `keyword` (names are a required part of MAST statements) is predefined by MAST and requires no further declaration. In this case keywords `var` and `val` are used for variables and values, respectively. `i` is used to declare a variable of type of current. The second `ic` is a variable. If many parameters need to be defined such as line 13),

commas are used to separate each of the parameters, alternatively they can be defined one by one. For example,

```
val v vou
val v vi
val v fo
val v voutcalc
```

#### ***B.2.3.4 Values Section***

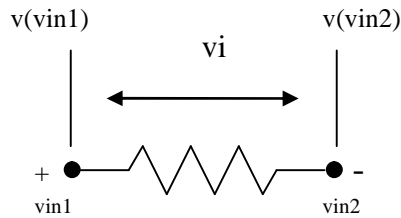
```
15)      values{
16)          #...Terminal Voltages
17)          vou = v(vout)-v(vin2)
18)          vi = v(vin1)-v(vin2)
19)          #...Fault offset voltage
20)          fo = m*vi+k
21)          voutcalc = -a*(vi+fo)
22)          #...Supply Voltage Limit
23)          if (voutcalc > 2.5) voutcalc = 2.5
24)          if (voutcalc < -2.5) voutcalc = -2.5
25)      }
```

In MAST the *values* section is either operational or declarative. It is set up for handling *foreign* functions that are required in the *equations* section, and promotes clarity in the equations. The *foreign* section is not discussed in this user guide, and the *equations* section will be introduced later. The *values* section starts with the keyword `values` followed by a left-hand brace (`{`), and the right-hand brace (`}`) must be used at the end when all statements are complete as seen in Figure B-3. Note: the left-hand brace should not start from a new line, which is regarded as an error during simulation. This is the same as other statements such as `equations` or `if`.

```
values {
    statements
}
```

**Figure B-3:** The syntax for the *values* statement

On line 18)  $v_i$  is defined in the local declaration section,  $vin1$  and  $vin2$  are the connection nodes defined in the header declaration section. This equation shows that the voltage between two nodes ( $v(vin1)$ ,  $v(vin2)$ ) is  $v_i$ . These variables are described using Figure B-4.



**Figure B-4:** The resistor

Lines 20) and 21) display the relationship between the input and output voltage. Parameters ( $a$ ,  $m$ ,  $k$ ) and values ( $v_o$ ,  $v_i$ ,  $f_o$ ,  $v_{outcalc}$ ) have been defined in the header declaration and local declaration, respectively. The `if` statement on lines 23) and 24) are used to execute two expressions. It can be used in `parameters`, `values`, `control`, and `equations` sections of the template, and also in `when` statements. If there is more than one expression, `else if` and `else` statements are required. Note: `else if` statement may appear more than once. The syntax is shown in Figure B-5:

```

if(expression){
    statements
}
else if(expression){
    statements
}
...
else {
    statements
}

```

**Figure B-5:** The syntax for `if` statement



### B.2.3.5 The Equations Section

The *equations* section describes the analogue characteristics at the terminals of the element being modeled.

```
26) equations{
27)     #...Fundamental Equations
28)     i(vout->vin2) += i
29)     vou = voutcalc
30) }
```

The syntax of the *equations* section is shown in Figure B-6.

```
equations{
    statements
}
```

**Figure B-6:** The syntax for *equations* section

Statements in the *equations* section either define the dependent through *vars* or *refs* in the system, they are expressed as the across variables or the equations necessary for each *var* declared in the template. It is important to know that a compound statement in a template can not have an empty body, e.g.,

```
equations{
    # statement -- that is illegal
}
```

On line 28) where all values have been defined in the local declaration, the symbol *->* indicates a flow of the through variable from the first node (*vout*) to the second (*vin2*). Operators (*+=*, *-=*) indicate whether to add to or subtract from the node. Line 29) shows that two specified voltages are equal. More details such as other types of statement which can be used in this section can be found in [Saber03]. Line 30) shows the end of the *equations* section. Lines 10) and 31) inform the *Saber* simulator when the *template* starts and finishes, respectively.

### B.2.3.6 Netlist Section

In order to check if the op amp behaves as expected a netlist is required. The general syntax of a netlist is shown in Figure B-7:

```
templatename.refdes connection_pt_list
    [=argument_assignments]
```

**Figure B-7:** The syntax for the netlist

where `templatename` indicates the name of template in the model. `refdes` is the reference designator, `connection_pt_list` shows the connections for the template, and `argument_assignments` supplies values of parameters for the model.

In this case an inverting amplifier is configured as shown in Figure B-8.

```
1) #... Top level of design
2) v.vinp y 0 = tran=(sin=(vo=0.1, va=0.5, f=100))
3) #...v.vinp y 0 = dc = 0.5
4) r.Ri y vin = 10k # define input resistance
5) #...define the op amp's nodes
6) test.inv vo vin 0 = 2, 0, 0
```

**Figure B-8:** The netlist for the operational amplifier

Line 1), line 3) and line 5) are comments. Line 2) declares a sinusoid voltage source, `v.vinp`, `v` is predefined as voltage (`i` will be used if a current source is required), `vinp` is the name of the sinusoid voltage source. `y` and `0` are two connections, `0` indicates ground. `tran` expresses *transient analysis*, `sin` stands for the sinusoid signal, the parameter `vo` is the offset voltage, `va`, `f` are the amplitude and the frequency, respectively. These parameters have to be defined in coupled square brackets. Line 3) defines a dc voltage source with a value of 0.5V. Line 4) declares the input resistor. `r` is predefined for resistors `Ri` and `Rl`. Line 6) instantiates the op amp defined above. `test` is the name of template for the op amp. `vo vin 0` are the connections of the

op amp, 2, 0, 0 are values of parameters. Note: these connections and values must match the orders and numbers of the op amp model.

### ***B.3 Implementation in the Cosmos Simulator***

In this section, different types of analysis are described in section B.3.1 and B.3.2, respectively.

#### **B.3.1 Simulation Run**

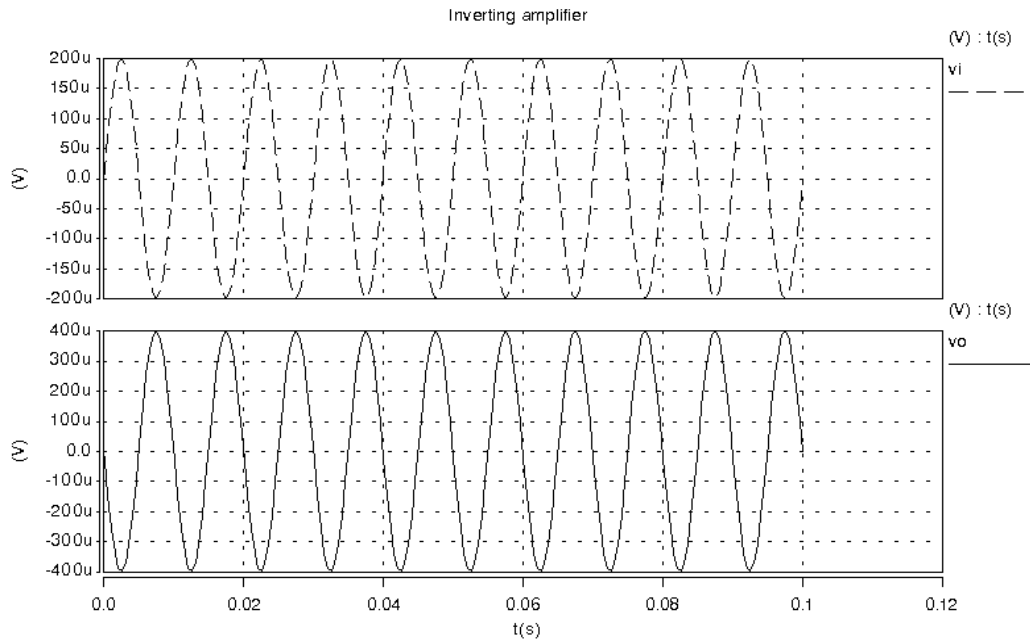
The aim of the section is to demonstrate how to use the *Cosmos* simulator. Follows steps: **file** -> **open** -> **design**, select a file with a *.sin* extension, the *Cosmos* transcript starts to compile, the **report** dialog appears with information such as copyright, license, and date. If there are errors, double click highlighted error parts in the dialogue, it will take the user to where the error is in the program. Note: when the program is changed, it has to be saved again, or the simulator will not recognize the update. After that the user may go back to the **file**, and then choose **reload design** to check if there are other errors. The analysis is implanted if there are not errors any more.

#### **B.3.2 Analysis**

The aim of the section is to observe if the model behaves as expected. Three types of analyses are introduced: the transient, dc and ac analysis.

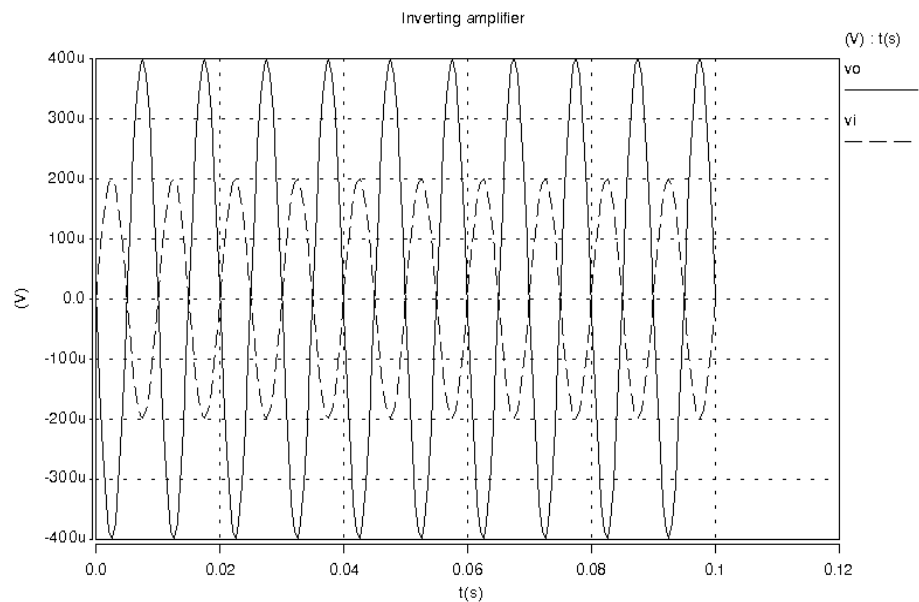
##### ***B.3.2.1 The Transient Analysis***

Transient analysis is typically used to investigate the response of the system to a time-dependent excitation. Uses: **analyses** -> **time** -> **domain** -> **transient**, after that a dialog appears to prompt users to input both **end time** and **time step**. Note: users must not input a unit for these values, otherwise an error is displayed. Then choose **yes** to **Run DC Analysis First**, after that press **ok**. Finally the signal is plotted from **plotfiles**: **file** -> **open** -> **plotfiles**, then find the directory where the file is saved, and choose the name with *.tr.ai\_pl* extension, two dialog boxes appear: **signal management** and *filename.tr.ai\_pl* extension. Therefore, users can display any signals. In this case signals *vi* and *vo* are selected shown in Figure B-9.



**Figure B-9:** Signals from the transient analyses

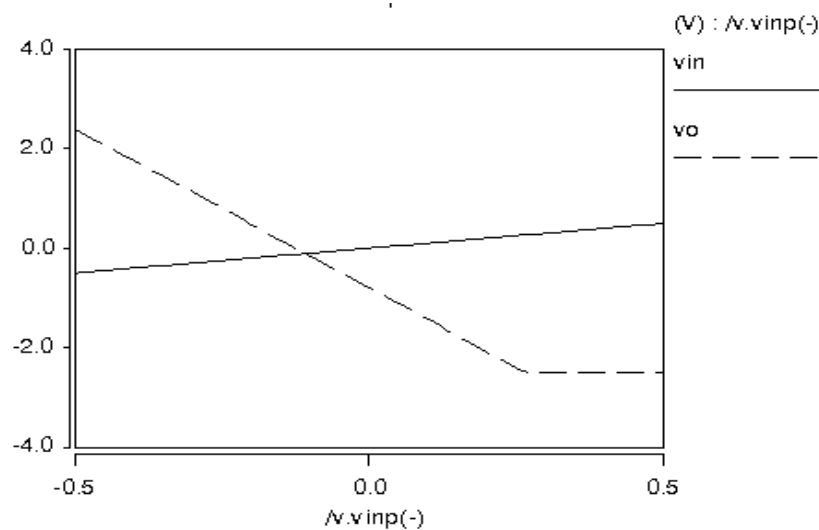
If the user prefers to compare these signals in the same graph, they initially have to select both signals, and then right click the mouse to choose **select signals -> stack region -> analogue 1**. Thus two signals are combined into a same box shown in Figure B-10.



**Figure B-10:** Combination between input and output signals

### B.3.2.2 DC Analysis

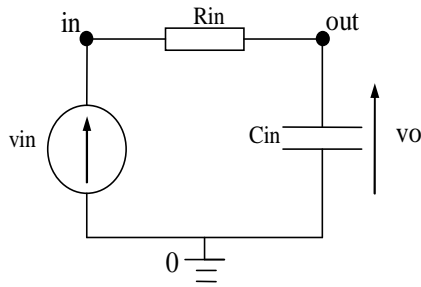
The dc analysis follows a similar procedure to transient analysis: **analyses** -> **operating point** -> **dc**, a dialog box appears to prompt the user to choose **independent source**, then **sweep type**, note: there are many types, in this case, **step by** is chosen. The range is defined by the user, for example, **from -0.5 to 0.5 by 100**, which means the voltage ranges between -0.5V to 0.5V with 100 steps. Values in **Sample Point Density** and **Monitor Progress** are set to their default values of 1 and 0 respectively, or defined by the designer. It is noted the former needs to be large, or an error appears. In this case it is set to 200. After that, **DC Analysis Run First** is selected. Finally, dc analysis is invoked. **Plotfile** is required to display signals: **file** -> **open** -> **plotfiles**, then a dialog box appears so that the file with *.dt.ai\_pl* extension is selected. Unfortunately, this extension does not appear automatically in the Plotfiles dialog box, thus the user needs to choose **All** in the **files of type**, then selects *.dt.ai\_pl* file. After that choose **open**, two dialog boxes appear: **signal management** and filename.*dt.ai\_pl*. The result is shown in Figure B-11.



**Figure B-11:** Results from the DC transfer analysis

### B.3.2.3 AC analysis

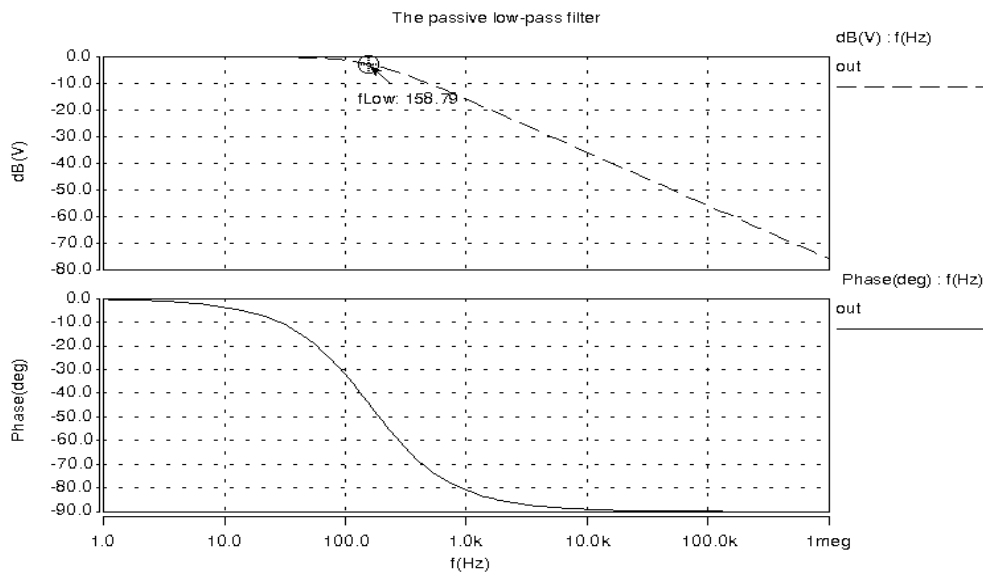
A simple RC circuit is used to demonstrate ac analysis as shown in Figure B-12:



- 1) #...AC voltage source
- 2) v.vin in 0 = ac = (1,0)
- 3) #...define value of R1 and C1
- 4) r.Rin in out = 1k
- 5) c.Cin out 0 = 1u

**Figure B-12:** The RC filter for the ac analysis

Most of lines have been explained except for line 2) that presents the ac source. The values in parentheses (1,0) stand for the magnitude and phase, respectively. The same procedure as for transient analysis is used except that **Analyses -> Frequency -> Small signal** is selected to display signals. They are shown in Figure B-13.



**Figure B-13:** Signals from AC analysis

Above all three frequently used analyses are introduced, others such as the Monte Carlo (MC) simulation can be found in [Saber03].

### ***B.4 Conclusion***

A user guide based on [Saber03] has been developed. It consists of two parts: the first one introduces the structure of the MAST language using an operational amplifier (op amp) model that includes commonly used statement and syntax with detailed

explanations for each line; the second part demonstrates how to subject the model to frequently used analyses. For each analysis, explanations are given. Comparing with *Saberbook* this user guide is structured more simply, especially for beginners to understand languages and the process of manipulating the simulator.

# ***Appendix C: Behavioral Models Written in MAST***

## ***C.1MAST Code of Linear HLFMs***

### **C.1.1 MAST Code of opdc**

```
#... behavioural model of an op amp for general modeling

template Opamp out inn inp vdd vss gnd = gain, r, c, ro, voffin,
voffout, ibn, ibp, vb1, vb2

electrical out, inn, inp, vdd, vss, gnd

#... parameters values
number gain = -1
number r = 1k
number c = 1u
number ro = 1k
number voffin = -0.5u
number voffout = 0.1u
number ibn = 0.1n
number ibp = 0.1n
number vb1 = 0.7
number vb2 = 0.7
{
    electrical off, n1, n2
    val v vin, vip, vi, voff, vout, vd, vs
    val i iR1, iRo
    var i iC1, ioff, i1, i2

    d.diode1 out n1
    d.diode2 n2 out
    #... thermal noise from the input resistor
    noiseR.nr inn off= r
    noiseR.nro out gnd= ro

    values{
        #... define all connections
```



```

vin = v(inn)-v(gnd)
vip = v(inp)-v(gnd)
vout = v(out)-v(gnd)
vd = v(vdd)-v(gnd)
vs = v(vss)-v(gnd)

#... equation between input and output
#... define the input offset voltage
voff = vip + voffin
vi = vin - voff

#... define the current for the resistor
iR1 = vi/r

#... output current
iRo = (vout + voffout - gain*vi)/ro
}

equations{
#... define current around the capacitor
iC1:iC1 = d_by_dt(vi*c)

#... current in input stage
i(inn->off)+= iR1+iC1

#... bias current
i(off->gnd)+=ibp
i(inn->gnd)+=ibn

#... current of the input impedance
i(inp->off)+= ibp-(iR1+iC1)

#... define current and voltage in the point
i(inp->off)+= ioff
ioff :v(inp)-v(off) = voff

#... current in output stage
i(out->gnd)+= iRo

#... define the bias voltage
i(vdd->n1) += il

```

```

        i1: v(vdd)-v(n1) = vb1

        i(n2->vss) += i2
        i2: v(n2)-v(vss) = vb2
    }
}

```

### C.1.2 MAST Code of opac

#... behavioural model of an op amp ac with extra pole and zero.

```

template Opamp out inn inp vdd vss gnd = r, c, rol, voffin, ibn, ibp,
vb1, vb2, ro2, cc, ra, rstuck, vstuck, ioffset, Gb, cpL, LzL, Ga

```

```

electrical out, inn, inp, vdd, vss, gnd

```

#... parameters values

```

#number gain = -1

```

```

number r = 1k

```

```

number c = 1u

```

```

number rol = 1k

```

```

number voffin = -0.5u

```

```

#number voffout = 0.1u

```

```

number ibn = 0.1n

```

```

number ibp = 0.1n

```

```

number vb1 = 0.7

```

```

number vb2 = 0.7

```

```

#number rdp = 1k

```

```

number ro2 = 1k

```

```

number cc = 10p

```

```

number ra = 1k

```

```

number rstuck = 1meg

```

```

number vstuck = 0

```

```

number ioffset = 0

```

```

number Gb = 2.9e-3

```

```

number cpL = 10p

```

```

number LzL = 1m

```

```

number Ga= 10e-6

```

```

{

```

```

    electrical off, n1, n2, no, nol, pL, zL

```

```

val v vin, vip, vi, voff, vout, vd, vs
val i iR1
var i iC1, ioff, i1, i2, iC2, izL

d.diode1 out n1
d.diode2 n2 out

values{
    #... define all connections
    vin = v(inn)-v(gnd)
    vip = v(inp)-v(gnd)
    vout = v(out)-v(gnd)
    vd = v(vdd)-v(gnd)
    vs = v(vss)-v(gnd)

    #... equation between input and output
    #... define the input offset voltage
    voff = vip + voffin
    vi = voff-vin

    #... define the current for the resistor
    iR1 = vi/r
}

equations{
    #... define current around the capacitor cin
    iC1:iC1 = d_by_dt(vi*c)

    #... current in input stage
    i(inn->off)+= iR1+iC1

    #... bias current
    i(off->gnd)+=ibp
    i(inn->gnd)+=ibn

    #... current of the input impedance
    i(inp->off)+= ibp+(iR1+iC1)

    #... define current and voltage in the point
    i(inp->off)+= ioff
    ioff :v(inp)-v(off) = voff
}

```

```

#... define the bias voltage
i(vdd->n1) += i1
i1: v(vdd)-v(n1) = vb1

i(n2->vss) += i2
i2: v(n2)-v(vss) = vb2

#... define the current between supply voltage
#i(vdd->vss) += idp

#... current in output stage between ro1
i(nol->out)+= (v(nol)-v(out))/ro1

#... define current around the capacitor cc
i(no->nol)+= iC2
iC2:iC2 = d_by_dt((v(no)-v(nol))*cc)

#... define current around Ra
i(no->gnd)+= (v(no)-v(gnd))/ra

#... define stuck current
#i(out->gnd)+=istuck

#... define the voltage for vnol
i(nol->gnd)+=(v(nol)-v(gnd))/ro2

#... additional poles and zeros
i(pL->gnd)+= vi*1e-3
i(pL->gnd)+=(v(pL)-v(gnd))*1e-3+ d_by_dt(((v(pL)-
v(gnd))*CpL)

i(zL->gnd)+=izL
izL:v(zL)-v(gnd)=(v(pL)-v(gnd))+ d_by_dt(((v(pL)-
v(gnd))*1e-3)*LzL)

#... define the voltage around no
i(no->gnd)+=Ga*(v(zL)-v(gnd))

#... define offset current for output stage
i(nol->gnd)+=ioffset

```

```

        #... define current between Gb
        i(no1->gnd)+= v(no)*Gb
    }
}

```

## ***C.2 MAST Code of Nonlinear HLFMs***

```

# this op amp is developed for surface response model.
element template wholedata inp inn out gnd = model,c,voffin,r
electrical inp, inn, out, gnd
number c=10p
number voffin=0.5m
number r=1k

struc {
    string
        file="E:\LiKun\Interpolation1\HSPICE4data\FinalData\FinalData500
m.txt"
    number      interp=1,
                extrap[4]=[1,1,1,1],
                fill[2]=[2,0],
                density[2]=[200,200]
    } model=()
{
foreign tlu

number      dim=2,datap[*],sp1[*],sp2[*]
#number rdat[*]
number vx1,vx2,vy

electrical off
val v v1,v2,vout
val i io, ir
var i iC,ioffin

parameters {
    datap=tlu(0,2,model->file,datap,1,[1,1,1,1],[2,0])
    #message("can the file % be loaded?,the length is %",model-
>file,datap)

```

```

    sp1=tlu(1,addr(datap),1,1)
    #rdat=tlu(7,addr(datap),0,0)
    #message("the rdat is %",rdat)
    sp2=tlu(1,addr(datap),2,1)
    #message("sp2 is %",sp2)
    vx1=0
    vx2=0
    vy = tlu(2,addr(datap),vx1,vx2)
    message("value at (%,% ) is %",vx1,vx2,vy)

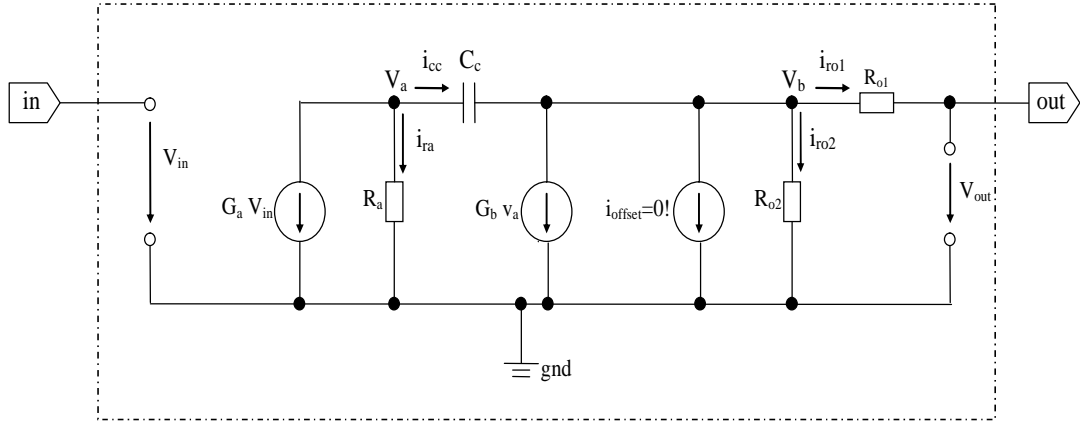
}
control_section {
    #...declare dependant source iout depends on two independant
sources v1 and v2
    pl_set(io,(v1,v2))
    #...two arrays sp1 and sp2 holds sample points for two variables
v1 and v2
    sample_points(v1,sp1)
    #message("the value is %",v1)
    sample_points(v2,sp2)
}
values {
    v1=v(inn)-v(gnd)
    v2=v(inp)-v(gnd)
    vout=v(out)-v(gnd)
    ir=v(inn,off)/r
    io=tlu(2,addr(datap),v1,v2)
    #iout=tlu(2,addr(datap),io,(v1-v2),vout)
}

equations {
    i(inp->off)+=ioffin
    ioffin:v(inp)-v(off)=voffin
    iC:iC = d_by_dt(v(off,inn)*c)
    i(inn->off)+=-iC
    i(inn->off)+=-ir
    i(inn->inp)+=-iC+ir)
    i(out->gnd)+=io
}
}

```

# **Appendix D: Analysis of Boyle's Output Stage in the Complex Frequency Domain**

The notation shown in Figure D-1 is used in the whole of Appendix D.



**Figure D-1:** Boyle's output stage

## ***D.1 Input Output Transfer Function***

According to the current law, the following system of equations is obtained:

$$\text{Node } V_a: i_{cc} + i_{ra} + G_a \cdot V_{in} = 0 \quad \text{Eq. D-1}$$

It is known:  $i_{ra} = \frac{v_a}{R_a}$ , and  $i_{cc} = (v_a - v_b) \cdot s \cdot C_c$ , substitute it into Eq. D-1:

$$(v_a - v_b) \cdot s \cdot C_c + \frac{v_a}{R_a} + G_a \cdot V_{in} = 0 \quad \text{Eq. D-2}$$

$$\text{Node } V_b: i_{cc} = G_b \cdot v_a + i_{ro2} + i_{ro1} \quad \text{Eq. D-3}$$

The above equation is then transformed into the following system of equations ( $i_{ro1} = 0$ , no load):

$$i_{cc} = G_b \cdot v_a + \frac{v_b}{R_{o2}} \quad \text{Eq. D-4}$$

With Eq. D-4 and  $i_{cc} = (v_a - v_b) \cdot s \cdot C_c$  one can derive:

$$\begin{aligned}
(v_a - v_b) \cdot s \cdot C_c &= G_b \cdot v_a + \frac{v_b}{R_{o2}} \\
\Leftrightarrow v_a \cdot s \cdot C_c - v_b \cdot s \cdot C_c &= G_b \cdot v_a + v_b \cdot \frac{1}{R_{o2}} \\
\Leftrightarrow v_a (s \cdot C_c - G_b) &= v_b \left( \frac{1}{R_{o2}} + s \cdot C_c \right) \\
\Leftrightarrow \frac{v_b}{v_a} &= \frac{s \cdot C_c - G_b}{\frac{1}{R_{o2}} + s \cdot C_c}
\end{aligned} \tag{Eq. D-5}$$

With Eq. D-1 and Eq. D-2 Eq. D-6 can be derived:

$$G_b \cdot v_a + v_b \cdot \frac{1}{R_{o2}} + v_a \cdot \frac{1}{R_a} + G_a \cdot V_{in} = 0 \tag{Eq. D-6}$$

$$v_a \cdot \left( G_b + \frac{1}{R_a} \right) = - \left( G_a \cdot V_{in} + v_b \cdot \frac{1}{R_{o2}} \right) \tag{Eq. D-7}$$

By combining Eq. D-7 with Eq. D-4 the final equation for the input output transfer function can be derived:

$$\begin{aligned}
\Leftrightarrow -v_b \cdot \frac{G_b + \frac{1}{R_a}}{G_a \cdot V_{in} + v_b \cdot \frac{1}{R_{o2}}} &= \frac{s \cdot C_c - G_b}{s \cdot C_c + \frac{1}{R_{o2}}} \\
\Leftrightarrow -v_b \cdot \left[ \left( G_b + \frac{1}{R_a} \right) \cdot \left( s \cdot C_c + \frac{1}{R_{o2}} \right) + \frac{1}{R_{o2}} \cdot (s \cdot C_c - G_b) \right] &= G_a \cdot (s \cdot C_c - G_b) \cdot V_{in} \\
\Leftrightarrow v_b \cdot \left[ G_b \cdot s \cdot C_c + \frac{G_b}{R_{o2}} + \frac{s \cdot C_c}{R_a} + \frac{1}{R_{o2} \cdot R_a} + \frac{(s \cdot C_c - G_b)}{R_{o2}} \right] &= -G_a \cdot (s \cdot C_c - G_b) \cdot V_{in} \\
\Leftrightarrow \frac{v_b \cdot \left[ G_b \cdot (1 + s \cdot C_c \cdot R_{o2}) + s \cdot C_c - G_b + \frac{1}{R_a} \cdot (1 + s \cdot C_c \cdot R_{o2}) \right]}{R_{o2} \cdot (s \cdot C_c - G_b)} &= -G_a \cdot V_{in}
\end{aligned}$$

Since  $i_{rol} = 0$  (no load!)  $v_b = V_{out}$ . Therefore,  $v_b$  is replaced by  $v_{out}$ :

$$\Leftrightarrow \frac{V_{out}}{V_{in}} = -G_a \cdot \frac{R_{o2} \cdot (s \cdot C_c - G_b)}{G_b \cdot (1 + s \cdot C_c \cdot R_{o2}) + s \cdot C_c - G_b + \frac{1}{R_a} \cdot (1 + s \cdot C_c \cdot R_{o2})}$$



$$\Leftrightarrow \frac{V_{out}}{V_{in}} = -G_a \cdot R_a \cdot \frac{s \cdot C_c \cdot R_{o2} - R_{o2} \cdot G_b}{1 + s \cdot C_c \cdot R_a \cdot \left(1 + \frac{R_{o2}}{R_a} + R_{o2} \cdot G_b\right)} \quad \text{Eq. D-8}$$

## D.2 Output Impedance

The current on the output node can be expressed as:

$$i_{out} = \frac{V_{out} - v_b}{R_{o1}} \quad \text{Eq. D-9}$$

$$\text{Therefore, } v_b = V_{out} - i_{out} \cdot R_{o1} \quad \text{Eq. D-10}$$

$$\text{Furthermore, } -i_{out} = \frac{v_b}{R_{o2}} + G_b \cdot v_a + v_b \cdot \frac{1}{R_a + \frac{1}{s \cdot C_c}} \quad \text{Eq. D-11}$$

$$v_b \cdot \frac{R_a}{R_a + \frac{1}{s \cdot C_c}} = v_a \quad \text{Eq. D-12}$$

With the help of Eq. D-5, Eq. D-11 and Eq. D-12 the following relation are obtained for the output impedance:

$$\begin{aligned} & -i_{out} + \frac{V_{out} - i_{out} \cdot R_{o1}}{R_{o2}} + G_b \cdot (V_{out} - i_{out} \cdot R_{o1}) \cdot \frac{s \cdot R_a \cdot C_c}{1 + s \cdot R_a \cdot C_c} + (V_{out} - i_{out} \cdot R_{o1}) \cdot \frac{s \cdot C_c}{1 + s \cdot R_a \cdot C_c} = 0 \\ \Leftrightarrow & -i_{out} + V_{out} \cdot \frac{1}{R_{o2}} - i_{out} \cdot \frac{R_{o1}}{R_{o2}} + V_{out} \cdot G_b \cdot \frac{s \cdot R_a \cdot C_c}{1 + s \cdot R_a \cdot C_c} - i_{out} \cdot R_{o1} \cdot G_b \cdot \frac{s \cdot R_a \cdot C_c}{1 + s \cdot R_a \cdot C_c} + V_{out} \cdot \frac{s \cdot C_c}{1 + s \cdot R_a \cdot C_c} \\ & - i_{out} \cdot R_{o1} \cdot \frac{s \cdot C_c}{1 + s \cdot R_a \cdot C_c} = 0 \\ \Leftrightarrow & V_{out} \cdot \frac{1}{R_{o2}} + V_{out} \cdot G_b \cdot \frac{s \cdot R_a \cdot C_c}{1 + s \cdot R_a \cdot C_c} + V_{out} \cdot \frac{s \cdot C_c}{1 + s \cdot R_a \cdot C_c} = \\ & i_{out} + i_{out} \cdot \frac{R_{o1}}{R_{o2}} + i_{out} \cdot R_{o1} \cdot G_b \cdot \frac{s \cdot R_a \cdot C_c}{1 + s \cdot R_a \cdot C_c} + i_{out} \cdot R_{o1} \cdot \frac{s \cdot C_c}{1 + s \cdot R_a \cdot C_c} \\ \Leftrightarrow & V_{out} \cdot \left( \frac{1}{R_{o2}} + G_b \cdot \frac{s \cdot R_a \cdot C_c}{1 + s \cdot R_a \cdot C_c} + \frac{s \cdot C_c}{1 + s \cdot R_a \cdot C_c} \right) = i_{out} \cdot \left( 1 + \frac{R_{o1}}{R_{o2}} + R_{o1} \cdot G_b \cdot \frac{s \cdot R_a \cdot C_c}{1 + s \cdot R_a \cdot C_c} + R_{o1} \cdot \frac{s \cdot C_c}{1 + s \cdot R_a \cdot C_c} \right) \\ \Leftrightarrow & V_{out} \cdot \left( \frac{1 + s \cdot R_a \cdot C_c}{R_{o2} \cdot (1 + s \cdot R_a \cdot C_c)} + \frac{s \cdot R_a \cdot C_c \cdot G_b \cdot R_{o2}}{R_{o2} \cdot (1 + s \cdot R_a \cdot C_c)} + \frac{s \cdot C_c \cdot R_{o2}}{R_{o2} \cdot (1 + s \cdot R_a \cdot C_c)} \right) = \\ & i_{out} \cdot \left( \frac{R_{o2} \cdot (1 + s \cdot R_a \cdot C_c)}{R_{o2} \cdot (1 + s \cdot R_a \cdot C_c)} + \frac{R_{o1} \cdot (1 + s \cdot R_a \cdot C_c)}{R_{o2} \cdot (1 + s \cdot R_a \cdot C_c)} + \frac{s \cdot R_{o1} \cdot G_b \cdot R_{o2} \cdot R_a \cdot C_c}{R_{o2} \cdot (1 + s \cdot R_a \cdot C_c)} + \frac{s \cdot R_{o1} \cdot R_{o2} \cdot C_c}{R_{o2} \cdot (1 + s \cdot R_a \cdot C_c)} \right) \end{aligned}$$

$$\begin{aligned}
&\Leftrightarrow V_{out} \cdot \left( \frac{1 + s \cdot R_a \cdot C_c + s \cdot R_{o2} \cdot R_a \cdot G_b \cdot C_c + s \cdot R_{o2} \cdot C_c}{R_{o2} \cdot (1 + s \cdot R_a \cdot C_c)} \right) = \\
&\quad i_{out} \cdot \left( \frac{R_{o2} + s \cdot R_{o2} \cdot R_a \cdot C_c + R_{o1} + s \cdot R_{o1} \cdot R_a \cdot C_c + s \cdot R_{o1} \cdot R_{o2} \cdot R_a \cdot G_b \cdot C_c + s \cdot R_{o1} \cdot R_{o2} \cdot C_c}{R_{o2} \cdot (1 + s \cdot R_a \cdot C_c)} \right) \\
&\Leftrightarrow \frac{V_{out}}{i_{out}} = \frac{R_{o1} + R_{o2} + s \cdot C_c \cdot (R_{o2} \cdot R_a + R_{o1} \cdot R_a + R_{o1} \cdot R_{o2} \cdot R_a \cdot G_b + R_{o1} \cdot R_{o2})}{1 + s \cdot C_c \cdot (R_a + R_a \cdot G_b \cdot R_{o2} + R_{o2})} \\
&\Leftrightarrow \frac{V_{out}}{i_{out}} = \frac{R_{o1} + R_{o2} + s \cdot C_c \cdot [R_a \cdot (R_{o1} + R_{o2} + R_{o1} \cdot R_{o2} \cdot G_b) + R_{o1} \cdot R_{o2}]}{1 + s \cdot C_c \cdot [R_a \cdot (1 + G_b \cdot R_{o2}) + R_{o2}]} \quad \text{Eq. D-13}
\end{aligned}$$

Thus, the output impedance is obtained.

# ***Appendix E: Manual Implementation for the MMGS***

## ***E.1 Process With the Offset Parameter***

### **E.1.1 The Estimator**

```
function [thm,yhat,p,phi,psi] = Estimator(z,nn,adm,adg,th0,p0,phi,psi)
% RARMAX Computes estimates recursively for an ARMAX model.

disp('im in the estimator');

% default values for the output signals
thm=[];           % initialise the estimates
yhat=[];         % initialize the estimated result
p=[];
phi=[];
psi=[];

if nargin < 4
    disp('Usage: MODEL_PARS = RARMAX(DATA,ORDERS,ADM,ADG)')
    disp('      [MODEL_PARS,YHAT,COV,PHI,PSI] =
RARMAX(DATA,ORDERS,ADM,ADG,TH0,COV0,PHI,PSI)')
    disp('      ADM is one of 'ff', 'kf', 'ng', 'ug'.')
    return
end
adm=lower(adm(1:2));
if ~(adm=='ff'|adm=='kf'|adm=='ng'|adm=='ug')
    error('The argument ADM should be one of 'ff', 'kf', 'ng', or
'ug'.')
end

[nz,ns]=size(z);[ordnr,ordnc]=size(nn);
if ns>2,error('This routine is for single input only. Use RPEM
instead!'),end
if ns==1,
    if ordnc~=2;error('For a time series nn should be [na nc]!'),end
else
    if ordnc~=4, error('the argument nn should be [na nb nc nk]!'),end,
end
if ns==1,
    na=nn(1);nb=0;nc=nn(2);nk=1;
else
    na=nn(1);nb=nn(2);nc=nn(3);nk=nn(4);nu=1;
end
if nk<1,error('Sorry, this routine requires nk>0; Shift input sequence
if necessary!'),end
d=na+nb+nc+1;           % define a parameter for the offset
if ns>2,error('Sorry, this routine is for single input only!'),end
if ns==1,nb=0;end
if nb==0,nk=1;end
nam=max([na,nc]);nbm=max([nb+nk-1,nc]);
tic=na+nb+1:na+nb+nc;
ia=1:na;iac=1:nc;
ib=nam+nk:nam+nb+nk-1;ibc=nam+1:nam+nc;
ic=nam+nbm+1:nam+nbm+nc;
```

```

id=nam+nbm+nc+1;

iia=1:nam-1;iib=nam+1:nam+nbm-1;iic=nam+nbm+1:nam+nbm+nc-1;
iid=nam+nbm+nc-1+1;
dm=nam+nbm+nc+1;
if nb==0,iib=[];end
ii=[iia iib iic iid];i=[ia ib ic id];

if nargin<8, psi=zeros(dm,1);end
if nargin<7, phi=zeros(dm,1);end
if nargin<6, p0=10000*eye(d);end
if nargin<5, th0=eps*ones(d,1);end
if isempty(psi),psi=zeros(dm,1);end
if isempty(phi),phi=zeros(dm,1);end
if isempty(p0),p0=10000*eye(d);end
if isempty(th0),th0=eps*ones(d,1);end
if length(th0)~=d, error('The length of th0 must equal the number of
estimated parameters!'),end
[th0nr,th0nc]=size(th0);if th0nr<th0nc, th0=th0';end
p=p0;th=th0; % initialise the p matrix
p1=p0;th1=th0;
p2=p0;th2=th0;
f=1

if adm(1)=='f', R1=zeros(d,d);lam=adg;end
if adm(1)=='k', [sR1,SR1]=size(adg);
    if sR1~=d | SR1~=d,
        error('The R1 matrix should be a square matrix with dimension
equal to number of parameters!'),
    end
    R1=adg;lam=1;
end
if adm(2)=='g',
    grad=1;
else
    grad=0;
end

thm0(nz,:)=th'; % separate different estimated parameters
thm1=thm0;
thm2=thm0;

aa=max(z(:,2)); % find out the maximum value of input
bb=min(z(:,2)); % find out the minimum value of input

yhat=[]; % set up the default condition for the output
for kcou=1:nz % start the loop
    if((bb<=z(kcou,2))&(z(kcou,2)<0.01))
        phi(id)=1;psi(id)=1;
        yh=phi(i)'*th;
        epsi=z(kcou,1)-yh;
        if ~grad,
            K=p*psi(i)/(lam + psi(i)'*p*psi(i));
            p=(p-K*psi(i)'*p)/lam+R1;
        else
            K=adg*psi(i);
        end
        if adm(1)=='n', K=K/(eps+psi(i)'*psi(i));end
        th=th+K*epsi;
        if nc>0,
            c=fstab([1;th(tic)]);

```

```

else
    c=1;
end
th(tic)=c(2:nc+1);
epsilon=z(kcou,1)-phi(i)'*th;
if nb>0,
    zb=[z(kcou,2),-psi(IBC)'];
else
    zb=[];
end
ztil=[[z(kcou,1),psi(iac)'];zb;[epsilon,-psi(ic)']]*c;

phi(ii+1)=phi(ii);psi(ii+1)=psi(ii);phi(iid)=1;psi(iid)=1;
if na>0,phi(1)=-z(kcou,1);psi(1)=-ztil(1);end
if nb>0,
    phi(nam+1)=z(kcou,2);psi(nam+1)=ztil(2);
end
if nb==0,
    zc=ztil(2);
else
    zc=ztil(3);
end
if nc>0,phi(nam+nbm+1)=epsilon;psi(nam+nbm+1)=zc;end

thm0(kcou,:)=th';yhat(kcou)=yh;
end

if((0.01<=z(kcou,2))&(z(kcou,2)<0.1))
    phi(id)=1;psi(id)=1;
    yh=phi(i)'*th1;
    epsi=z(kcou,1)-yh;
    if ~grad,
        K=p1*psi(i)/(lam + psi(i)'*p1*psi(i));
        p1=(p1-K*psi(i)'*p1)/lam+R1;
    else
        K=adg*psi(i);
    end
    if adm(1)=='n', K=K/(eps+psi(i)'*psi(i));end
    th1=th1+K*epsi;
    if nc>0,
        c=fstab([1;th1(tic)'])';
    else
        c=1;
    end
    th1(tic)=c(2:nc+1);
    epsilon=z(kcou,1)-phi(i)'*th1;
    if nb>0,
        zb=[z(kcou,2),-psi(IBC)'];
    else
        zb=[];
    end
    ztil=[[z(kcou,1),psi(iac)'];zb;[epsilon,-psi(ic)']]*c;

    phi(ii+1)=phi(ii);psi(ii+1)=psi(ii);
    phi(iid)=1;
    psi(iid)=1;
    if na>0,phi(1)=-z(kcou,1);psi(1)=-ztil(1);end
    if nb>0,
        phi(nam+1)=z(kcou,2);psi(nam+1)=ztil(2);
    end
    if nb==0,

```

```

        zc=ztil(2);
    else
        zc=ztil(3);
    end
    if nc>0,phi(nam+nbm+1)=epsilon;psi(nam+nbm+1)=zc;end

    thm1(kcou,:)=th1';yhat(kcou)=yh;
end

if((0.1<=z(kcou,2))&(z(kcou,2)<=aa))
phi(id)=1;psi(id)=1;
yh=phi(i)'*th2;
epsi=z(kcou,1)-yh;
if ~grad,
    K=p2*psi(i)/(lam + psi(i)'*p2*psi(i));
    p2=(p2-K*psi(i)'*p2)/lam+R1;
else
    K=adg*psi(i);
end
if adm(1)=='n', K=K/(eps+psi(i)'*psi(i));end
th2=th2+K*epsi;
if nc>0,
    c=fstab([1;th2(tic)]);
else
    c=1;
end
th2(tic)=c(2:nc+1);
epsilon=z(kcou,1)-phi(i)'*th2;
if nb>0,
    zb=[z(kcou,2),-psi(IBC)'];
else
    zb=[];
end
ztil=[[z(kcou,1),psi(iac)'];zb;[epsilon,-psi(ic)']]*c;

phi(ii+1)=phi(ii);psi(ii+1)=psi(ii);phi(iid)=1;psi(iid)=1;
if na>0,phi(1)=-z(kcou,1);psi(1)=-ztil(1);end
if nb>0,
    phi(nam+1)=z(kcou,2);psi(nam+1)=ztil(2);
end
if nb==0,
    zc=ztil(2);
else
    zc=ztil(3);
end
if nc>0,phi(nam+nbm+1)=epsilon;psi(nam+nbm+1)=zc;end

    thm2(kcou,:)=th2';yhat(kcou)=yh;
end
end
yhat = yhat';

% combine these three groups of parameters
thm=[thm0,thm1,thm2];

```

### E.1.2 The Predictor

```

function yhat= Predictor(u,nn,thm)

% define the size of input u, and the size of matrix nn

```

```

[nz,ns]=size(u)
[ordnr,ordnc]=size(nn);
na=nn(1);nb=nn(2);nc=nn(3);nk=nn(4);nu=1;

if nk<1,error('Sorry, this routine requires nk>0; Shift input sequence
if necessary!'),end
d=na+nb+nc+1; % define number of parameters from nn

%if ns==1,nb=0;end
if nb==0,nk=1;end

nam=max([na,nc]);nbm=max([nb+nk-1,nc]);
tic=na+nb+1:na+nb+nc;
ia=1:na;iac=1:nc;
ib=nam+nk:nam+nb+nk-1;
ibc=nam+1:nam+nc;
ic=nam+nbm+1:nam+nbm+nc;
id=nam+nbm+nc+1; % increase the dimation of th

iia=1:nam-1;iib=nam+1:nam+nbm-1;iic=nam+nbm+1:nam+nbm+nc-
1;iid=nam+nbm+nc-1+1;
dm=nam+nbm+nc+1;

if nb==0,iib=[];end
ii=[iia iib iic iid];i=[ia ib ic id];

psi=zeros(dm,1);
phi=zeros(dm,1);

aaa=phi(i)
th0=eps*ones(d,1);

[th0nr,th0nc]=size(th0);if th0nr<th0nc, th0=th0';end

[tm,tn]=size(thm); % define the size of resulting estimating

p=round(tn/d); % find out how many groups, and the closest integer

% find the the indices of thm that point to nonzero elements.
% If none is found, find returns an empty matrix.
ind0=find(abs(thm(:,1))>eps);
%ind0
g0=size(ind0)

ind1=find(abs(thm(:,1+d))>eps);
%ind1
g1=size(ind1)

ind2=find(abs(thm(:,1+2*d))>eps);
%ind2
g2=size(ind2)

% find the best model from each group by searching the three maximum
index (from 1 to d, from d+1 to 2d, from 2d+1 to 3d)
th=thm(max(ind0),1:d)'
th1=thm(max(ind1),(1+d):2*d)'
th2=thm(max(ind2),(1+2*d):3*d)'

aa=max(u); % find out the maximum value of input
bb=min(u); % find out the minimum value of input

```

```

yhat=[]; % set up the default condition for the output
for kcou=1:nz % start the loop
    if((bb<=u(kcou))&(u(kcou)<0.01))
        phi(id)=1;psi(id)=1;
        %d=1
        yh=phi(i)'*th;
        %aaa=phi(i)
        if nb>0
            zb=[u(kcou),-psi(IBC)'];
        else
            zb=[];
        end

        phi(ii+1)=phi(ii);psi(ii+1)=psi(ii);phi(iid)=1;psi(iid)=1;
        if na>0,phi(1)=-yh;end
        if nb>0,
            phi(nam+1)=u(kcou);
        end

        if nc>0,phi(nam+nbm+1)=0;psi(nam+nbm+1)=0;end

        yhat(kcou)=yh;
        %yh
    end

    if((0.01<=u(kcou))&(u(kcou)<0.1))
        phi(id)=1;psi(id)=1;
        %d=2
        yh1=phi(i)'*th1;
        if nb>0,
            zb=[u(kcou),-psi(IBC)'];
        else
            zb=[];
        end

        phi(ii+1)=phi(ii);psi(ii+1)=psi(ii);phi(iid)=1;psi(iid)=1;

        if na>0
            phi(1)=-yh1;
        end

        if nb>0
            phi(nam+1)=u(kcou);
        end

        if nc>0,phi(nam+nbm+1)=0;psi(nam+nbm+1)=0;end

        yhat(kcou)=yh1;
        %yh1
    end

    if((0.1<=u(kcou))&(u(kcou)<=aa))
        phi(id)=1;psi(id)=1;
        %d=3
        yh2=phi(i)'*th2;
        %yh2
        if nb>0,
            zb=[u(kcou),-psi(IBC)'];
        else
            zb=[];
        end
    end
end

```



```

        %phi(1)
        phi(ii+1)=phi(ii);psi(ii+1)=psi(ii);phi(iid)=1;psi(iid)=1;
        if na>0,
            phi(1)=-yh2;
        end
        if nb>0
            phi(nam+1)=u(kcou);
        end

        if nc>0,phi(nam+nbm+1)=0;psi(nam+nbm+1)=0;end

        yhat(kcou)=yh2;
    end
end
yhat = yhat';

```

## ***E.2 Process without the offset parameter***

### **E.2.1 The Estimator**

```

function [thm,yhat,p,phi,psi] =
AME_nooffset(z,nn,adm,adg,th0,p0,phi,psi)
% RARMAX Computes estimates recursively for an ARMAX model.

% default values for the output signals
thm=[]; % initialise the estimates
yhat=[]; % initialize the predicted result
p=[];
phi=[];
psi=[];

if nargin < 4
    disp('Usage: MODEL_PARS = RARMAX(DATA,ORDERS,ADM,ADG)')
    disp(' [MODEL_PARS,YHAT,COV,PHI,PSI] =
RARMAX(DATA,ORDERS,ADM,ADG,TH0,COV0,PHI,PSI)')
    disp(' ADM is one of ''ff'', ''kf'', ''ng'', ''ug''.')
    return
end
adm=lower(adm(1:2));
if ~(adm=='ff'|adm=='kf'|adm=='ng'|adm=='ug')
    error('The argument ADM should be one of ''ff'', ''kf'', ''ng'', or
''ug''.')
end

[nz,ns]=size(z);[ordnr,ordnc]=size(nn);
if ns>2,error('This routine is for single input only. Use RPEM
instead!'),end
if ns==1,
    if ordnc~=2;error('For a time series nn should be [na nc]!'),end
else
    if ordnc~=4, error('the argument nn should be [na nb nc nk]!'),end,
end
if ns==1,
    na=nn(1);nb=0;nc=nn(2);nk=1;
else
    na=nn(1);nb=nn(2);nc=nn(3);nk=nn(4);nu=1;
end
if nk<1,error('Sorry, this routine requires nk>0; Shift input sequence
if necessary!'),end
d=na+nb+nc;

```

```

if ns>2,error('Sorry, this routine is for single input only!'),end
if ns==1,nb=0;end
if nb==0,nk=1;end
nam=max([na,nc]);nbm=max([nb+nk-1,nc]);
tic=na+nb+1:na+nb+nc;
ia=1:na;iac=1:nc;
ib=nam+nk:nam+nb+nk-1;ibc=nam+1:nam+nc;
ic=nam+nbm+1:nam+nbm+nc;

iia=1:nam-1;iib=nam+1:nam+nbm-1;iic=nam+nbm+1:nam+nbm+nc-1;
dm=nam+nbm+nc;
if nb==0,iib=[];end
ii=[iia iib iic];i=[ia ib ic];

if nargin<8, psi=zeros(dm,1);end
if nargin<7, phi=zeros(dm,1);end
if nargin<6, p0=10000*eye(d);end
if nargin<5, th0=eps*ones(d,1);end
if isempty(psi),psi=zeros(dm,1);end
if isempty(phi),phi=zeros(dm,1);end
if isempty(p0),p0=10000*eye(d);end
if isempty(th0),th0=eps*ones(d,1);end
if length(th0)~=d, error('The length of th0 must equal the number of
estimated parameters!'),end
[th0nr,th0nc]=size(th0);if th0nr<th0nc, th0=th0';end
p=p0;th=th0; % initialise the p matrix
p1=p0;th1=th0;
p2=p0;th2=th0;
f=1
thm0(nz,:)=th'; % seperate different estimated parameters
thm1=thm0;
thm2=thm0;

if adm(1)=='f', R1=zeros(d,d);lam=adg;end
if adm(1)=='k', [sR1,SR1]=size(adg);
if sR1~=d | SR1~=d,
error('The R1 matrix should be a square matrix with dimension
equal to number of parameters!'),
end
R1=adg;lam=1;
end
if adm(2)=='g',
grad=1;
else
grad=0;
end

aa=max(z(:,2)) % find out the maximum value of input
bb=min(z(:,2)) % find out the minimum value of input

yhat=[]; % set up the default condition for the output
for kcou=1:nz % start the loop
if ((bb<=z(kcou,2))&(z(kcou,2)<0.01))
%d=1
yh=phi(i) '*th;
epsi=z(kcou,1)-yh;
if ~grad,
K=p*psi(i)/(lam + psi(i) '*p*psi(i));
p=(p-K*psi(i) '*p)/lam+R1;
else
K=adg*psi(i);

```

```

end
if adm(1)=='n', K=K/(eps+psi(i)*psi(i));end
th=th+K*epsi;
if nc>0,
    c=fstab([1;th(tic)]);
else
    c=1;
end
th(tic)=c(2:nc+1);
epsilon=z(kcou,1)-phi(i)*th;
if nb>0,
    zb=[z(kcou,2),-psi(IBC)'];
else
    zb=[];
end
ztil=[[z(kcou,1),psi(iac)'];zb;[epsilon,-psi(ic)']]*c;

phi(ii+1)=phi(ii);psi(ii+1)=psi(ii);
if na>0,phi(1)=-z(kcou,1);psi(1)=-ztil(1);end
if nb>0,
    phi(nam+1)=z(kcou,2);psi(nam+1)=ztil(2);
end
if nb==0,
    zc=ztil(2);
else
    zc=ztil(3);
end
if nc>0,phi(nam+nbm+1)=epsilon;psi(nam+nbm+1)=zc;end

thm0(kcou,:)=th';yhat(kcou)=yh;
end

if ((0.01<=z(kcou,2))&(z(kcou,2)<0.1))
yh=phi(i)*th1;
epsi=z(kcou,1)-yh;
if ~grad,
    K=p1*psi(i)/(lam + psi(i)*p1*psi(i));
    p1=(p1-K*psi(i)*p1)/lam+R1;
else
    K=adg*psi(i);
end
if adm(1)=='n', K=K/(eps+psi(i)*psi(i));end
th1=th1+K*epsi;
if nc>0,
    c=fstab([1;th1(tic)]);
else
    c=1;
end
th1(tic)=c(2:nc+1);
epsilon=z(kcou,1)-phi(i)*th1;
if nb>0,
    zb=[z(kcou,2),-psi(IBC)'];
else
    zb=[];
end
ztil=[[z(kcou,1),psi(iac)'];zb;[epsilon,-psi(ic)']]*c;

phi(ii+1)=phi(ii);psi(ii+1)=psi(ii);
if na>0,phi(1)=-z(kcou,1);psi(1)=-ztil(1);end
if nb>0,
    phi(nam+1)=z(kcou,2);psi(nam+1)=ztil(2);

```

```

end
if nb==0,
    zc=ztil(2);
else
    zc=ztil(3);
end
if nc>0,phi(nam+nbm+1)=epsilon;psi(nam+nbm+1)=zc;end

thm1(kcou,:)=th1';yhat(kcou)=yh;
end

if((0.1<=z(kcou,2))&(z(kcou,2)<=aa))
    %h=3
    yh=phi(i) '*th2;
    epsi=z(kcou,1)-yh;
    if ~grad,
        K=p2*psi(i)/(lam + psi(i) '*p2*psi(i));
        p2=(p2-K*psi(i) '*p2)/lam+R1;
    else
        K=adg*psi(i);
    end
    if adm(1)=='n', K=K/(eps+psi(i) '*psi(i));end
    th2=th2+K*epsi;
    if nc>0,
        c=fstab([1;th2(tic)]);
    else
        c=1;
    end
    th2(tic)=c(2:nc+1);
    epsilon=z(kcou,1)-phi(i) '*th2;
    if nb>0,
        zb=[z(kcou,2),-psi(IBC)'];
    else
        zb=[];
    end
    ztil=[[z(kcou,1),psi(iac)'];zb;[epsilon,-psi(ic)']]*c;

    phi(ii+1)=phi(ii);psi(ii+1)=psi(ii);
    if na>0,phi(1)=-z(kcou,1);psi(1)=-ztil(1);end
    if nb>0,
        phi(nam+1)=z(kcou,2);psi(nam+1)=ztil(2);
    end
    if nb==0,
        zc=ztil(2);
    else
        zc=ztil(3);
    end
    if nc>0,phi(nam+nbm+1)=epsilon;psi(nam+nbm+1)=zc;end

    thm2(kcou,:)=th2';yhat(kcou)=yh;
end
end
yhat = yhat';

% combine these three groups of parameters
thm=[thm0,thm1,thm2];

```

### E.2.2 The Predictor

```

function yhat= AMP_nooffset(u,nn,thm)
% mrramax2 generates the output model

```

```

disp('Im in the predictor');
% define the size of input u, and the size of matrix nn
[nz,ns]=size(u);[ordnr,ordnc]=size(nn);
na=nn(1);nb=nn(2);nc=nn(3);nk=nn(4);nu=1;

if nk<1,error('Sorry, this routine requires nk>0; Shift input sequence
if necessary!'),end

% define number of parameters from nn
d=na+nb+nc;

if nb==0,nk=1;end

nam=max([na,nc]);nbm=max([nb+nk-1,nc]);
tic=na+nb+1:na+nb+nc;
ia=1:na;iac=1:nc;
ib=nam+nk:nam+nb+nk-1;
ibc=nam+1:nam+nc;
ic=nam+nbm+1:nam+nbm+nc;

iia=1:nam-1;iib=nam+1:nam+nbm-1;iic=nam+nbm+1:nam+nbm+nc-1;
dm=nam+nbm+nc;

if nb==0,iib=[];end
ii=[iia iib iic];i=[ia ib ic];

psi=zeros(dm,1);
phi=zeros(dm,1);

aaa=phi(i)
th0=eps*ones(d,1);

[th0nr,th0nc]=size(th0);if th0nr<th0nc, th0=th0';end

% define the size of resulting estimating
[tm,tn]=size(thm);

% find out how many groups, and the closest integer
p=round(tn/d);

% find the the indices of thm that point to nonzero elements.
% If none is found, find returns an empty matrix.
ind0=find(abs(thm(:,1))>eps);

g0=size(ind0)

ind1=find(abs(thm(:,1+d))>eps);
ind2=find(abs(thm(:,1+2*d))>eps);

% find the best model from each group by searching the three maximum
index (from 1
% to d, from d+1 to 2d, from 2d+1 to 3d)
th=thm(max(ind0),1:d)'
th1=thm(max(ind1),(1+d):2*d)'
th2=thm(max(ind2),(1+2*d):3*d)'

aa=max(u) % find out the maximum value of input
bb=min(u) % find out the minimum value of input

yhat=[]; % set up the default condition for the output

```

```

for kcou=1:nz                                % start the loop

    if((bb<=u(kcou))&(u(kcou)<0.01))
        yh=phi(i)'*th;
        if nb>0
            zb=[u(kcou),-psi(IBC)'];
        else
            zb=[];
        end

        phi(ii+1)=phi(ii);psi(ii+1)=psi(ii);
        if na>0,phi(1)=-yh;end
        if nb>0,
            phi(nam+1)=u(kcou);
        end

        if nc>0,phi(nam+nbm+1)=0;psi(nam+nbm+1)=0;end

        yhat(kcou)=yh;
    end

    if ((0.01<=u(kcou))&(u(kcou)<0.1))
        %d=2
        yh1=phi(i)'*th1;
        if nb>0,
            zb=[u(kcou),-psi(IBC)'];
        else
            zb=[];
        end

        phi(ii+1)=phi(ii);psi(ii+1)=psi(ii);

        if na>0
            phi(1)=-yh1;
        end

        if nb>0
            phi(nam+1)=u(kcou);
        end

        if nc>0,phi(nam+nbm+1)=0;psi(nam+nbm+1)=0;end

        yhat(kcou)=yh1;
        %yh1
    end

    if((0.1<=u(kcou))&(u(kcou)<=aa))
        yh2=phi(i)'*th2;
        if nb>0,
            zb=[u(kcou),-psi(IBC)'];
        else
            zb=[];
        end

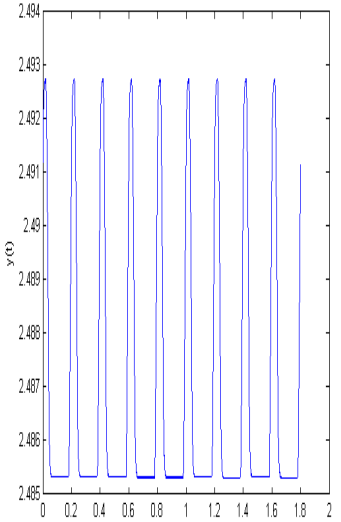
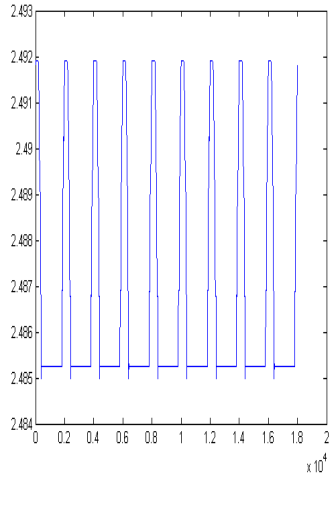
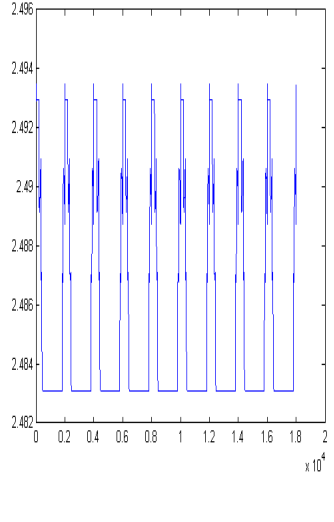
        phi(ii+1)=phi(ii);psi(ii+1)=psi(ii);
        if na>0,
            phi(1)=-yh2;
        end
        if nb>0
            phi(nam+1)=u(kcou);
        end
    end
end

```

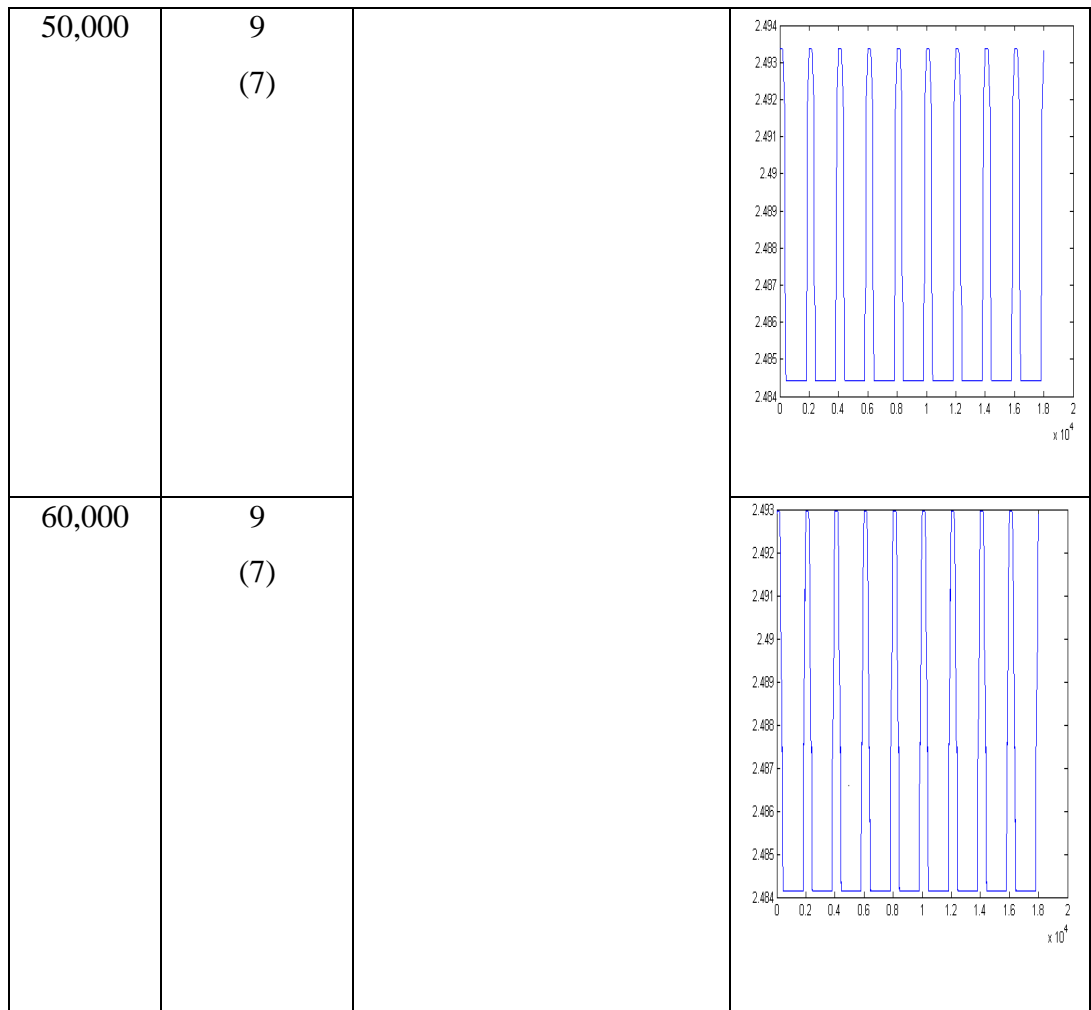
```
        if nc>0,phi(nam+nbm+1)=0;psi(nam+nbm+1)=0;end
        yhat(kcou)=yh2;
    end
end
yhat = yhat';
```

# ***Appendix F: Quality Measurement based on Number of Samples***

It is known that generally more samples produce better quality results during modeling and simulation. However, this is not always true. An example is shown, in which many simulations are run using various numbers of samples. The same pulse waveform is used each time as input to an open-loop amplifier. 9 models are generated by the MMGS for prediction. Predicted signals are plotted in Figure F-1.

<i>No. of samples</i>	<i>No. of models (stable models )</i>	<i>Original Signal</i>	<i>Predicted Signal</i>
30,000	9 (6)	 <p>The plot shows a square wave pulse train. The y-axis is labeled 'y(t)' and ranges from 2.485 to 2.494. The x-axis is labeled 'Size' and ranges from 0 to 2 x 10<sup>4</sup>. The signal alternates between approximately 2.485 and 2.493.</p>	 <p>The plot shows a square wave pulse train. The y-axis ranges from 2.484 to 2.493. The x-axis is labeled 'x 10<sup>4</sup>' and ranges from 0 to 2. The signal alternates between approximately 2.485 and 2.492.</p>
40,000	9 (8)		 <p>The plot shows a square wave pulse train. The y-axis ranges from 2.482 to 2.496. The x-axis is labeled 'x 10<sup>4</sup>' and ranges from 0 to 2. The signal alternates between approximately 2.485 and 2.493.</p>





**Figure F-1:** Predicted signals based on different samples

It is seen that these predicted signals can match the original signal with good accuracy. However, by comparing the signal in 30,000 with the one in 40,000 samples, it is seen that the former is more accurate. It indicates that more samples do not have to give rise to better results.

As it is seen the output signal using 40,000 samples shows that it is struggling on saturation regions, it indicates that samples in some of models may not be informative. Although it has more stable models (8) than others, some of them may not be very accurate because information such as the number of samples has to be balanced across all models used.

# ***Appendix G: Methodologies for Quality Improvement of the MMGS in MATLAB***

In this section various methods are used in either the AME or AMP to improve their quality and diagnose problems in this system. This work focuses mainly on the AME because models are generated here. In addition, it has been discovered that the AME has a higher tolerance of handling nonideal or wrong parameters than the predictor.

## ***G.1 Suitable Values Check***

In the AME a command *isnan* is used within an *if* statement in MATLAB to detect if the estimator fails to converge shown in Eq. G-1. When not a number (NaN) is detected the program is stopped. The special status is indicated by a K appearing that indicates the keyboard takes control. Where *yh* is the estimated output signal, *keyboard* indicates the keyboard takes control.

```
if isnan(yh)  
    keyboard  
end
```

Eq. G-1

## ***G.2 Sample Detection***

A method for counting the number of samples in each model was developed in both the AME and AMP, so that the author is able to observe if each individual model has obtained sufficient information to behave correctly. A variable *count* is defined and initialized in Eq. G-2, where *modt* indicates the total number of models.

```
count=zeros(1,modt)
```

Eq. G-2

A *for* loop is used to count how many samples present in each model shown in Eq. G-3, where *modn* stands for the number of models, *+1* indicates one more sample is checked.

```

for modn = 1 : modt
    count(modn) = count(modn) + 1
end

```

Eq. G-3

For instance, assume there is a five-model system with 20,000 samples, and that during the simulation the number of samples in each model is shown in Figure G-1:

*count* = 2508      2756      2694      9522      2520

**Figure G-1:** Number of samples in each model

It is seen that all models have similar amount of samples except for the fourth one, which indicates that it may exhibit high nonlinearity and require more samples to converge properly.

### ***G.3 Observation of Covariance $p$***

In the estimator the diagonal element of covariance  $p$  is observed. It contains a large diagonal value (initialized from 10,000), so that the model may find it hard to gain enough information. However,  $p$  is a  $d \times d$  matrix, where  $d$  is the size of variables. In order to achieve diagonal elements it has to be reshaped. A command *reshape* is used to shape the matrix into a  $1 : d \times d$  row and save it in a temporary file (*ptmp*), as shown in Eq. G-4).

$$ptmp = reshape(p(20000,1 : d * d), d, d) \tag{Eq. G-4}$$

Eq. G-5 is designed to pick up only diagonals (*pdiag*), where *eye(d)* gives the  $d$ -by- $d$  identity matrix with 1's on the diagonal and 0's elsewhere; *.\** indicates array multiplication, which only produces the element-by-element product of the arrays; *ones(d,1)* shows values of 1 in the  $d$ -by-1 matrix.

$$pdia = eye(d). *ptmp * ones(d,1) \tag{Eq. G-5}$$

After that a test based on these diagonal elements is implemented. If they are very large the model may be disregarded.

#### ***G.4 Stability Detector***

It is known that an unstable model may cause inaccuracies, so it is necessary to detect such models and replace them with stable ones. The unstable model is caused because there are not enough samples for the RLS estimator to tune. If this model is used it may cause the predictor to numerically explore very quickly. Therefore, it is necessary to ignore this model and use its neighbouring one. In  $z$  transform the way to know if a model is stable is to check the roots of its polynomial (within a unit circle), in the delta transform or Laplace transform the roots of its polynomial (left half plane) is observed.

In the MMGS we developed a stability detector, which consists of two parts: the first part detects these unstable models iteratively; the second one replaces the unstable one with the nearest one. It is noticed during detection if the first model is unstable, the last model from the first scan is used to replace it, but it is not the neighbour of the first model. Thus, during model detection the detector starts from detecting the last model to the first one. The second iteration remains increasing order. With this method a stable model will be available to replace the first unstable model.

During first iteration the roots of the model are found using the MATLAB *roots* command. If a model is unstable, it will be stored in a temporary file *temp*, otherwise the stable model is stored. This is implemented in MATLAB with an *if-else* statement as shown in Figure G-2, where *h* is an array to handle variables for outputs; *na* is the number of the output variables; *abs* is used to find absolute values for all the roots.

```
h=[1 thm(j,1:na)];  
Noofroot=roots(h);  
abvalue=abs(Noofroot);  
  
if any(abvalue>1)  
    temp=[temp j];  
else  
    thst=thm(j,:);  
    pst=p(j,:);  
end
```

**Figure G-2:** 1<sup>st</sup> iteration for unstable model detection

The function *any* is used to find out if all values are greater than unit, *thst* and *pst* are variables used to store the coefficients and covariance of the stable model, respectively.

The second iteration ensures there is always a stable model. Figure G-3 shows that if the model is unstable the previous stable one from the first iteration is used.

```
if any(abvalue > 1)
    thm(j,:) = thst;
    pst(j,:) = pst;
else
    thst = thm(j,:);
    pst = p(j,:);
end
```

**Figure G-3:** 2<sup>nd</sup> iteration for unstable models replacement

With this method, quality of predicted signal is significantly improved.

### ***G.5 The Saturation Detector***

It is known that the estimator only works well with the right excitation and input information; if there is a long period of saturation part from the input signal the estimator may not be trained well. Therefore, a saturation detector was designed in order to find constant outputs and remove them.

Initially the first sample in the output library is compared with its neighbouring one (the second one), if they are same its neighbour's index is stored in a library. Then the second sample is compared with the third one, and so on until all samples are processed. These indices stored are then deleted. One sample in the saturation part is allowed so that the estimator can still estimate the model under the saturation conditions.

# Appendix H: Codes for the MMGS and MMGSD

## H.1 The MMGS

### H.1.1 The AME

```
function
[thm,yhat,epsilon,epsilonhat,epsilonTest,threshold,threshold1,threshold2,pm,phi,psi] = ame(z,nn,adm,adg,th0,p0,phi,psi)
% Model selector based on RARMARX
disp('you are in the estimator');

% default values for the output signals
thm=[]; % initialise the estimates
yhat=[]; % initialize the predicted result
epsilon=[];
epsilonhat=[];
epsilonTest=[];
threshold=[];
threshold1=[];
threshold2=[];
p=[];
phi=[];
psi=[];

% adm and adg are part of forgetting factor, adm is adaptation mechanism,
% and adg is adaptation gain
if nargin < 4
    disp('Usage: MODEL_PARS = RARMARX(DATA,ORDERS,ADM,ADG)')
    disp(['MODEL_PARS,YHAT,COV,PHI,PSI] =
RARMARX(DATA,ORDERS,ADM,ADG,TH0,COV0,PHI,PSI)')
    disp('ADM is one of 'ff', 'kf', 'ng', 'ug'.')
    return
end
adm=lower(adm(1:2));
if ~(adm=='ff'|adm=='kf'|adm=='ng'|adm=='ug')
    error('The argument ADM should be one of 'ff', 'kf', 'ng', or
'ug'.')
end

% new input and output data are analysed
[nz,ns]=size(z); % define the new output data without saturation

[ordnr,ordnc]=size(nn); % define the size of matrix for all parameters

if ns~=3,error('This routine is for double inputs only. Use RPEM
instead!'),end
if ns==1,
    if ordnc~=2,error('For a time series nn should be [na nc]!'),end
else
    %if ordnc~=4, error('the argument nn should be [na nb nc nk]!'),end,
    if ordnc~=5, error('the argument nn should be [na nb nc nk ne]!'),end,
end
if ns==1,
    na=nn(1);nb=0;nc=nn(2);nk=1;ne=0
else
    na=nn(1);nb=nn(2);nc=nn(3);nk=nn(4);nu=1;ne=nn(5);
end
if nk<1,error('Sorry, this routine requires nk>0; Shift input sequence if
necessary!'),end
```

```

d=na+nb+nc+1+ne; % define a parameter for the offset

if ns==1,nb=0;ne=0;end
if nb==0|ne==0,nk=1;end

nam=max([na,nc]);nbm=max([nb+nk-1,nc]);
ndm=max([1,nc]); % extra data for offset

nem=max([ne+nk-1,nc]);

tic=na+nb+1:na+nb+nc;
%tic=na+nb+ne+1:na+nb+ne+nc;
ia=1:na;iac=1:nc;
ib=nam+nk:nam+nb+nk-1;ibc=nam+1:nam+nc;
ic=nam+nbm+1:nam+nbm+nc;
id=nam+nbm+nc+1;
idc=nam+nbm+nc+1:nam+nbm+nc+ndm; % set indices for the offset

ie=nam+nbm+nc+ndm+nk:nam+nbm+nc+ndm+ne+nk-1; % define for the second inputs
iec=nam+nbm+nc+ndm+1:nam+nbm+nc+ndm+nc;

iia=1:nam-1;iib=nam+1:nam+nbm-1;iic=nam+nbm+1:nam+nbm+nc-1;
iid=nam+nbm+nc+1:nam+nbm+nc+ndm-1; % set indices for the offset
%iid=nam+nbm+nc+1; % set indices for the offset

iie=nam+nbm+nc+ndm+1:nam+nbm+nc+ndm+nem-1; % for the second input

dm=nam+nbm+nc+ndm+nem;
%if nb==0,iib=[];end
ii=[iia iib iic iid iie]
i=[ia ib ic id ie]

if nargin<8, psi=zeros(dm,1);end
if nargin<7, phi=zeros(dm,1);end
if nargin<6, p0=10000*eye(d);end
if nargin<5, th0=eps*ones(d,1);end
if isempty(psi),psi=zeros(dm,1);end
if isempty(phi),phi=zeros(dm,1);end
if isempty(p0),p0=10000*eye(d);end
if isempty(th0),th0=eps*ones(d,1);end
if length(th0)~=d, error('The length of th0 must equal the number of estimated
parameters!'),end
[th0nr,th0nc]=size(th0);if th0nr<th0nc, th0=th0';end

if adm(1)=='f', R1=zeros(d,d);lam=adg;end
if adm(1)=='k', [sR1,SR1]=size(adg);
if sR1~=d | SR1~=d,
error('The R1 matrix should be a square matrix with dimension equal to
number of parameters!'),
end
R1=adg;lam=1;
end
if adm(2)=='g',
grad=1;
else
grad=0;
end

% only the last 5000 samples are interested
testinterval=[(nz-10000):nz];

u=z(:,2); % define the input data for estimator (whole data)
u2=z(:,3); % define the input data for estimator (whole data)
%u=[u1 u2];

%utest1=z(testinterval,2); % define the input data for the test
%utest2=z(testinterval,3); % define the input data for the test
utest=z(testinterval,2); % define the input data for the test

```

```

aa=max(max(u)); % find out the maximum value of all inputs
bb=min(min(u)); % find out the minimum value of all inputs

threshold=[bb,aa]; % initialise threshold

%disp('the number of division for the input has to be integer');
division4input=5;

if(division4input==0)
    error('the number of division for the input can not be zero');
end

if(rem(division4input,2)==0)
    error('the number of divisions for the input has to be even');
end

middleIndex=(division4input+1)/2; % define the middle of index for u=0;

inRange=(aa-bb)/division4input; % define the number of ranges for input,
% the division number has to be even
interval=bb:inRange:aa; % define the interval of input
LenInterval=length(interval); % define the length of interval

for j=1:(LenInterval-1) % define a range
    indInterval{j}=find((utest<interval(j+1))&(utest>=interval(j)));
    % find out the index by using accelerator
end
indInterval;

lengthThresh(1)=0; % initialise the size of first threshold
lengthThresh(2)=length(threshold); % initialise the size of threshold

% running the estimator
yhat=[]; % set up the default condition for the output

while (lengthThresh(1)~=lengthThresh(2))
    modt=lengthThresh(2)-1; % define the total No. of models, which is
noofthreshod-1
    for modn=1:modt % define the No. of models, which is noofthreshod-1
        thm(modn,1:d)=th0'; % initialize thm
        pm(modn,1:d*d)=p0(:)'; % initialize pm--covariance
    end

    % disp('display the number of models');
    % decide which range of u is used, result in the value of j,j decide which
model.
    low=threshold(1:modt);
    sizeoflower=size(low);

    high=threshold(2:modt+1);
    sizeofhigher=size(high);

    %threshold=sort(threshold)
    count=zeros(1,modt); % initialization
    %indexthreshold=[];
    psi(id)=1;
    for kcou=1:nz % start the loop for estimator

        % define the threshold index,i.e., where the threshold is
indexthreshold=find((u(kcou)>=low)&(u(kcou)<=high));

        modn=min(indexthreshold); % make sure there is only one index

        count(modn)=count(modn)+1;

        th=thm(modn,1:d)'; % redefine the parameters
        eeee=d*d;
    end
end

```



```

ffff=d;
p=reshape(pm(modn,1:d*d)',d,d); % redefine the covariance
modn;
%pause
phi(id)=1;
%psi(id)=1;
yh=phi(i)'*th;
gggg=phi(i)';
epsi=z(kcou,1)-yh; % define the innovation error
if ~grad,
    K=p*psi(i)/(lam + psi(i)'*p*psi(i));
    p=(p-K*psi(i)'*p)/lam+R1;
else
    K=adg*psi(i);
End

% parameters for unpdating th
if adm(1)=='n', K=K/(eps+psi(i)'*psi(i));end
th=th+K*epsi; % update the innovation error

if nc>0,
    % stabilizes a MONIC polynomial with respect to the unit circle
    c=fstab([1;th(tic)]);
    sizeofc=size(c);
else
    c=1;
end
th(tic)=c(2:nc+1);

epsilon=z(kcou,1)-phi(i)'*th; % define the residual error
if nb>0,
    zb=[z(kcou,2),-psi(IBC)'];
else
    zb=[];
end

if ne>0,
    ze=[z(kcou,3),-psi(IEC)'];
else
    ze=[];
end

ztil=[[z(kcou,1),psi(iac)'];zb:[epsilon,-psi(IC)'];[1,-
psi(idc)'];ze]*c;

% shifting procedure
phi(ii+1)=phi(ii);
psi(ii+1)=psi(ii);

% update parameters for output
if na>0,phi(1)=-z(kcou,1);psi(1)=-ztil(1);end

% update parameters for input
if nb>0
    phi(nam+1)=z(kcou,2);psi(nam+1)=ztil(2);
end
if nb==0,
    zc=ztil(2);
else
    zc=ztil(3);
end

% update noise parameters
if nc>0
    phi(nam+nbm+1)=epsilon;
    psi(nam+nbm+1)=zc;
end

```

```

% update information for the offset
if nb==0,
    zd=ztil(3);
else
    zd=ztil(4);
end
phi(nam+nbm+nc+1)=1;
psi(nam+nbm+nc+1)=zd;

% update information for the second input
if ne>0
    if nb==0,
        phi(nam+nbm+nc+ndm+1)=z(kcou,3);
        psi(nam+nbm+nc+ndm+1)=ztil(4);
    else
        phi(nam+nbm+nc+ndm+1)=z(kcou,3);
        psi(nam+nbm+nc+ndm+1)=ztil(5);
    end
end

% store and update these data
thm(modn,1:d)=th';
pm(modn,1:d*d)=p(:)';
modn;
yhat(kcou)=yh;

if isnan(yh)
    keyboard
end
epsilonhat(kcou)=epsilon;
end % end of for loop

epsilonTest=epsilonhat(testinterval); % inform the range of data we choose

% define the minimum value in each interval
miniSizeInt1(middleIndex)=min(epsilonTest(indInterval{middleIndex}));

% define the maximum value in each interval
maxSizeInt1(middleIndex)=max(epsilonTest(indInterval{middleIndex}));

% define the median range of each interval
mediamRange1(middleIndex)=(maxSizeInt1(middleIndex)-
miniSizeInt1(middleIndex))/2;

% post-estimation: after the estimator, error range can be defined.
for j=1:(LenInterval-1) % define a range
    % define the minimum value in each interval
    miniSizeInt{j}=min(epsilonTest(indInterval{j}));

    % define the maximum value in each interval
    maxSizeInt{j}=max(epsilonTest(indInterval{j}));

    % define the median range of each interval
    mediamRange{j}=(maxSizeInt{j}-miniSizeInt{j})/2;

    % define the critical range of each interval
    criticalRange{j}=(maxSizeInt{j}+miniSizeInt{j})/2;

    % the procedure for adding a new threshold
    if(abs(mediamRange{j}-mediamRange1(middleIndex))>criticalRange{j})

        % define the criteria of range
        % if the index is greater than middle, we use the smaller index
        if(j>middleIndex)

            % add one threshold on lower index
            threshold=[threshold,interval(j)];

```

```

        bbb=1;
    else
        % if the index is less than middle,we use the larger index
        % add one threshold on higher index
        threshold=[threshold,interval(j+1)];
    end
    else
        % show there is change if size of threshold remains same
        threshold=threshold;
    end
end

% sort thresholdtest in the ascending order, and increase
threshold1=sort(threshold);
threshold1=[threshold1 0]; % increase the length by 1

lengthThresh1=length(threshold1); % define the length of threshold

% remove the same thresholds
aaaa=threshold1(1:(lengthThresh1-1));
bbbb=threshold1(2:lengthThresh1);

% comparing numbers between 1st to the one b4 and 2nd to last
tmp=bbbb-aaaa;
% define the index when difference of two numbers are not equal
index=find(tmp~=0);
threshold2=threshold1(index); % find sorted thresholds which are not equal

lengthThresh(1)=lengthThresh(2); % old length of threshold increase
ab=lengthThresh(1);

% new length of threshold goes to second
lengthThresh(2)=length(threshold2);
cd=lengthThresh(2);
threshold=threshold2;
end % end of while loop
count
% display the result for these parameters
thm=thm;
p=pm;
yhat = yhat';
epsilonTest=epsilonTest';
epsiEsSize=size(epsilonTest);

threshold2=threshold'
sizethreshold=size(threshold2)

```

## H.1.2 The AMP

```

function [yhat] = amp(u,nn,thm,threshold)
% Model predictor based on RARMARX
disp('you are in the predictor');

u1=u(:,1);
u2=u(:,2);

% default values for the output signals
yhat=[]; % initialize the predicted result

% only input data is interested
[nz,ns]=size(u);
[ordnr,ordnc]=size(nn);

```

```

% define names for the parameters in the matrix
na=nn(1);nb=nn(2);nc=nn(3);nk=nn(4);nu=1;ne=nn(5);

d=na+nb+nc+1+ne; % define a parameter for the offset

if ns>3,error('Sorry, this routine is double single input only!'),end

if ns==1,nb=0;ne=0;end
if nb==0|ne==0,nk=1;end

nam=max([na,nc]);nbm=max([nb+nk+ne-1,nc]);
ndm=max([1,nc]); % extra data for offset
nem=max([ne+nk-1,nc]);

tic=na+nb+ne+1:na+nb+ne+nc;
ia=1:na;iac=1:nc;
ib=nam+nk:nam+nb+nk-1;ibc=nam+1:nam+nc;
ic=nam+nbm+1:nam+nbm+nc;
id=nam+nbm+nc+1;
idc=nam+nbm+nc+1:nam+nbm+nc+ndm; % set indices for the offset

ie=nam+nbm+nc+ndm+nk:nam+nbm+nc+ndm+ne+nk-1; % define for the second inputs
iec=nam+nbm+nc+ndm+1:nam+nbm+nc+ndm+nc;

iia=1:nam-1;iib=nam+1:nam+nbm-1;iic=nam+nbm+1:nam+nbm+nc-1;
iid=nam+nbm+nc+1:nam+nbm+nc+ndm-1; % set indices for the offset

iie=nam+nbm+nc+ndm+1:nam+nbm+nc+ndm+nem-1; % for the second input

dm=nam+nbm+nc+ndm+nem;
%if nb==0,iib=[];end

ii=[iia iib iic iid iie];i=[ia ib ic id ie];

psi=zeros(dm,1);
phi=zeros(dm,1);

[modt,columnofth]=size(thm);

if d~=columnofth
    error('these two must be the same')
end

low=threshold(1:modt)

```

```

sizeoflower=size(low);

high=threshold(2:modt+1)
sizeofhigher=size(high);

%threshold=sort(threshold)
count=zeros(1,modt);           % initialization

for kcou=1:nz                   % start the loop for estimator
    % define the threshold index,i.e., where the threshold is
    indexthreshold=find((u(kcou)>=low)&(u(kcou)<=high)));
    if ~isempty(indexthreshold)
        modn=min(indexthreshold);           % make sure there is only one index

        count(modn)=count(modn)+1;         % find number of samples on each model
        %kcou
        %thm
        th=thm(modn,1:d)';                 % redefine the parameters
        phi(id)=1;

        yh=phi(i)'*th;

        if nc>0,
            sizeoftic=length(tic);
            sizeofth=length(th);
            % stabilizes a MONIC polynomial with respect to the unit circle
            c=fstab([1;th(tic)])';
        else
            c=1;
        end

        th(tic)=c(2:nc+1);

        epsilon=yh-phi(i)'*th;             % define the residual error

        phi(ii+1)=phi(ii);

        if na>0,phi(1)=-yh;end

        % update parameters for the first input
        if nb>0
            phi(nam+1)=u1(kcou);
        end

        if nc>0,phi(nam+nbm+1)=epsilon;end
    end
end

```

```

    phi(nam+nbm+nc+1)=1;

    % update information for the second input
    if ne>0
        if nb==0,
            phi(nam+nbm+nc+ndm+1)=u2(kcou);
        else
            phi(nam+nbm+nc+ndm+1)=u2(kcou);
        end
    end

    % store and undate these data
    thm(modn,1:d)=th';
    yhat(kcou)=yh;

    if isnan(yh)
        keyboard
    end
else
    fprintf('u(%g)=%g',kcou,u(kcou));
    error('no model available!');
end
end
% end of for loop
yhat = yhat';

```

## ***H.2 The MMGSD***

### **H.2.1 The AME**

```

function
[thm,yhat,epsilon,epsilonhat,epsilonTest,threshold,threshold1,threshold2,pm,ph
i,psi] = ame(z,nn,adm,adg,Ts,th0,p0,phi,psi)

% In this estimator we are going to undeltarise epsilon and epsie
% Model selector based on RARMARX
disp('you are in the estimator');

% default values for the output signals
thm=[]; % initialise the estimates
yhat=[]; % initialize the predicted result
dyh=[];
epsilon=[];
epsilonhat=[];
epsilonTest=[];
threshold=[];
threshold1=[];
threshold2=[];
p=[];
pm=[];
phi=[];
psi=[];
dz=[];
dphi=[];
dphi4y=[];

```

```

dpsi=[];
depsilon=[];
depsi=[];
yh=[];

% adm and adg are part of forgetting factor, adm is adaptation mechanism,
% and adg is adaptation gain
if nargin < 4
    disp('Usage: MODEL_PARS = RARMAX(DATA,ORDERS,ADM,ADG)')
    disp(' [MODEL_PARS,YHAT,COV,PHI,PSI] =
RARMAX(DATA,ORDERS,ADM,ADG,TH0,COV0,PHI,PSI)')
    disp('ADM is one of 'ff', 'kf', 'ng', 'ug'.')
    return
end
adm=lower(adm(1:2));
if ~(adm=='ff'|adm=='kf'|adm=='ng'|adm=='ug')
    error('The argument ADM should be one of 'ff', 'kf', 'ng', or
'ug'.')
end

% new input and output data are analysed
[nz,ns]=size(z); % define the new output data without saturation

[ordnr,ordnc]=size(nn); % define the size of matrix for all parameters

if ns~=3,error('This routine is for double inputs only. Use RPEM
instead!'),end
if ns==1,
    if ordnc~=2;error('For a time series nn should be [na nc]!'),end
else
    %if ordnc~=4, error('the argument nn should be [na nb nc nk]!'),end,
    if ordnc~=5, error('the argument nn should be [na nb nc nk ne]!'),end,
end
if ns==1,
    na=nn(1);nb=0;nc=nn(2);nk=1;ne=0;
else
    na=nn(1);nb=nn(2);nc=nn(3);nk=nn(4);nu=1;ne=nn(5);
end
if nk<1,error('Sorry, this routine requires nk>0; Shift input sequence if
necessary!'),end
d=na+nb+nc+1+ne; % define a parameter for the offset

if ns==1,nb=0;ne=0;end
if nb==0|ne==0,nk=1;end

if nc>1
    tic=na+nb+1:na+nb+nc;
else
    tic=[];
end

% create the array for i
iia=1:na;
iib=na+nk:na+nb+nk-1;
if nc>1
    iic=na+nb+1:na+nb+nc;
else
    iic=[];
end

iidd=na+nb+nc+1;
iie=na+nb+nc+1+nk:na+nb+nc+1+ne+nk-1; % define for the second inputs

% create the array for ii
iia=1:na-1;
iib=na+1:na+nb-1;

if nc>1

```

```

        iic=na+nb+1:na+nb+nc-1;
    else
        iic=[];
    end
    iid=na+nb+nc+1;           % set indices for the offset
    iie=na+nb+nc+1+1:na+nb+nc+1+ne-1;           % for the second input

    ii=[iia iib iic iid iie];
    i=[iiaa iiib iiic iiid iiie];
    sizeofiii=size(i);

    dm=na+nb+nc+1+ne;

    if nargin<9, psi=zeros(dm,1);end
    if nargin<8, phi=zeros(dm,1);end
    if nargin<7, p0=10000*eye(d);end
    if nargin<6, th0=eps*ones(d,1);end
    if isempty(psi),psi=zeros(dm,1);end
    if isempty(phi),phi=zeros(dm,1);end
    if isempty(p0),p0=10000*eye(d);end

    % initialise dpsii
    dpsii=zeros(dm,nc+1);
    sizeofdpsii=size(dpsii);

    % initialise dpsit
    dpsit=zeros(dm,1);
    sizeofdpsit=size(dpsit);

    if isempty(th0),th0=eps*ones(d,1);end
    if length(th0)~=d, error('The length of th0 must equal the number of estimated
parameters!'),end
    [th0nr,th0nc]=size(th0);if th0nr<th0nc, th0=th0';end

    if adm(1)=='f', R1=zeros(d,d);lam=adg;end
    if adm(1)=='k', [sR1,SR1]=size(adg);
        if sR1~=d | SR1~=d,
            error('The R1 matrix should be a square matrix with dimension equal to
number of parameters!'),
            end
            R1=adg;lam=1;
        end
    if adm(2)=='g',
        grad=1;
    else
        grad=0;
    end

    % only the last 5000 samples are interested
    testinterval=[(nz-8000):nz];

    u=z(:,2);           % define the input data for estimator (whole
data)
    u2=z(:,3);           % define the input data for estimator (whole
data)

    %utest=z(testinterval,2);           % define the input data for the test

    aa=max(max(u));           % find out the maximum value of all inputs
    bb=min(min(u));           % find out the minimum value of all inputs

    threshold=[bb,aa];           % initialise threshold

    [modt,modtcolumn]=size(threshold);

    division4input=5;

    if(division4input==0)

```



```

    error('the number of division for the input can not be zero');
end

if(rem(division4input,2)==0)
    error('the number of divisions for the input has to be even');
end

middleIndex=(division4input+1)/2; % define the middle of index for u=0;

inRange=(aa-bb)/division4input % define the number of ranges for input,
% the division number has to be even
interval=bb:inRange:aa % define the interval of input
LenInterval=length(interval) % define the length of interval

for k=1:(LenInterval-1) % define a range
    % find out the index by using accelerator
    indInterval{k}=find((u<interval(k+1))&(u>=interval(k)));
end
sizeofindInterval=size(indInterval)

lengthThresh(1)=0; % initialise the size of first threshold
lengthThresh(2)=length(threshold); % initialise the size of threshold

% running the estimator
yhat=[]; % set up the default condition for the
output
while (lengthThresh(1)~=lengthThresh(2))
    modt=lengthThresh(2)-1; % define the total No. of models, which is
noofthreshod-1
    for modno=1:modt % define the No. of models, which is
noofthreshod-1
        thm(modno,1:d)=th0'; % initialize thm
        pm(modno,1:d*d)=p0(:)'; % initialize pm--covariance
    end

    % disp('display the number of models');
    % decide which range of u is used, result in the value of j,j decide which
model.
    low=threshold(1:modt);
    sizeoflower=size(low);

    high=threshold(2:modt+1);
    sizeofhigher=size(high);

    %threshold=sort(threshold)
    count=zeros(1,modt); % initialization
    %indexthreshold=[];

    th=th0;
    psi(iiid)=1;
    for kcou=1:nz % start the loop for estimator
        %kcou
        % define the threshold index,i.e., where the threshold is
        indexthreshold=find((u(kcou)>=low)&(u(kcou)<=high));

        modn=min(indexthreshold); % make sure there is only one index

        count(modn)=count(modn)+1;

        % Preventing the jump while the models are switching
        if count(modn)~=1
            th=thm(modn,1:d)'; % redefine the parameters
        end
        p=reshape(pm(modn,1:d*d)',d,d); % reshape the covariance

        % increase the size for the offset
        phi(iiid)=1;
        psi(iiid)=1;
    end
end

```

```

% the operation for the delta operator
phi4y=phi(iiia);
dphi4y = delta4y([z(kcou,1) phi4y']',Ts);
sizeofphi4y=length(dphi4y);

dz(kcou,1)=dphi4y(1);
dphiy=dphi4y(2:sizeofphi4y);

phi4u=phi(iiib);
dphi4u = delta4y(phi4u',Ts);
sizeofphi4u=length(dphi4u);

%phi(iiic)=0;
dphi4c = delta4y(phi(iiic)',Ts);
sizeofphiic=size(phi(iiic)');

dphi4d = delta4y(phi(iiid)',Ts);
sizeofphiid=size(phi(iiid)');

phi4e=phi(iiie);
dphi4e = delta4y(phi4e',Ts);
sizeofphi4e=length(dphi4e);

% Make sure the delay is always 1
dphi = [-dphiy dphi4u dphi4c dphi4d dphi4e]';

dyh=dphi(i)'*th;

depsi=dz(kcou,1)-dyh; % define the innovation error

epsi=undelta([depsi dphi(iiic)'],Ts,0);
%depsi1=undelta([depsi dphi4c],Ts,1)

if nc>0,
    % stabilizes a MONIC polynomial with respect to the unit circle
    c=[1;th(tic)];
    sizeofc=size(c);
    %disp('im here!');
    th(tic)=c(2:nc+1);

    dpsit=[dphi -dpsi(1:dm,2:nc+1)]*c;
    dpsi(1:dm,1)=dpsit;
    %dpsi(1:dm,1)=[dphi -dpsi(1:dm,2:nc+1)]*c
    sizeofdpsidm1=size(dpsi(1:dm,1));
    psi=undelta(dpsi,Ts,0);
    dpsi=[zeros(dm,1) undelta(dpsi,Ts,1)];
    %pause
else
    c=1;
    dpsit=dphi*c;
    dpsi=dpsit;
    psi=dpsi;
    dpsi=zeros(dm,1);
end

if ~grad,
    K=p*dpsit(i)/(lam + dpsit(i)'*p*dpsit(i));
    sizeofK=size(K);
    p=(p-K*dpsit(i)'*p)/lam+R1;
else
    K=adg*dpsit(i);
end
% parameters for unpdating th
if adm(1)=='n', K=K/(eps+dpsit(i)'*dpsit(i));end

th=th+K*depsi; % update the innovation error

```

```

% define the residual error
depsilon=dz(kcou,1)-dphi(i)'*th;

%epsilon=undelta([depsilon dphi(iiic)'],Ts,0)
epsilon=undelta([depsilon dphi(iiic)'],Ts,0);

% undeltarise the output
yh=undelta([dyh -dphi(iiia)'],Ts,0);
%yh=undelta([dyh dphiy],Ts,0);

% shifting procedure for phi
phi(ii+1)=phi(ii);
%psi(ii+1)=psi(ii);
phi(iid)=1;

phi(1)=z(kcou,1);
phi(na+1)=u(kcou);
phi(na+nb+1)=epsilon;
phi(na+nb+nc+1)=1;
phi(na+nb+nc+1+1)=u2(kcou);

% store and undate these data
thm(modn,1:d)=th';
pm(modn,1:d*d)=p(:)';

yhat(kcou)=yh;

if isnan(yh)
    keyboard
end

%epsilonhat(kcou)=depsi;
epsilonhat(kcou)=depsilon;
if isnan(depsilon)
    keyboard
end

end % end of for loop

%epsilonTest=epsilonhat(testinterval);% inform the range of data we choose
epsilonTest=epsilonhat; % inform the range of data we choose

% post-estimation: after the estimator, error range can be defined.
for j=1:(LenInterval-1) % define a range
    % define the minimum value in each interval
    miniSizeInt(j)=min(epsilonTest(indInterval{j}));

    % define the maximum value in each interval
    maxSizeInt(j)=max(epsilonTest(indInterval{j}));

    % define the mediam range of each interval
    mediamRange(j)=(maxSizeInt(j)-miniSizeInt(j))/2;

    % define the critical range of each interval
    criticalRange(j)=(maxSizeInt(j)+miniSizeInt(j))/2;
end
oldmiddleIndex=middleIndex

% Detect the index for the minimum range
middleIndex=min(find(min(mediamRange)==mediamRange))

%for m=(LenInterval-1):-1:1
for m=1:(LenInterval-1)
    % the procedure for adding a new threshold
    if(abs(mediamRange(m)-mediamRange(middleIndex))>criticalRange(m))

% define the criteria of range
% if the index is greater than middle,we use the smaller index

```

```

        if(m>middleIndex)
        indtest1=find(interval(m)==threshold)
        if (isempty(indtest1))
            % add one threshold on lower index
            threshold=[threshold,interval(m)]
            break
        end
    else % if the index is less than middle,we use the larger index
        indtest2=find(interval(m+1)==threshold)
        if (isempty(indtest2))
            % add one threshold on higher index
            threshold=[threshold,interval(m+1)]
            break
        end
    end
end
else % show there is change if size of threshold remains same
    threshold=threshold;
end
end
    threshold1=sort(threshold); % sort thresholdtest in the ascending
order, and increase
    threshold1=[threshold1 0]; % increase the length by 1

    lengthThresh1=length(threshold1); % define the length of threshold

    % remove the same thresholds
    aaaa=threshold1(1:(lengthThresh1-1));
    bbbb=threshold1(2:lengthThresh1);
    tmp=bbbb-aaaa; % comparing numbers between 1st to the one b4 and
2nd to last

    index=find(tmp~=0); % define the index when difference of two numbers
are not equal

    threshold2=threshold1(index); % find sorted thresholds which are not
equal

    lengthThresh(1)=lengthThresh(2); % old length of threshold increase
ab=lengthThresh(1)
    lengthThresh(2)=length(threshold2); % new length of threshold goes to
second
    cd=lengthThresh(2)
    threshold=threshold2

end
count

% display the result for these parameters
thm=thm;
pm=pm;
yhat = yhat';
epsilonTest=epsilonTest';
epsiEsSize=size(epsilonTest);

threshold2=threshold'
sizethreshold=size(threshold2)

```

## H.2.2 The AMG

```
function [yhat] = amp(u,nn,thm,threshold,Ts)
```

```

% In this estimator we are going to undeltarise epsilon and epsie
% Model selector based on RARMARX
disp('you are in the predictor.');
```

```
yhat=[];
```

```

% default values for the output signals
u1=u(:,1);
sizeofu1=size(u1);
u2=u(:,2);
sizeofu2=size(u2);

% new input and output data are analysed
[nz,ns]=size(u); % define the new output data without saturation
%[ordnr,ordnc]=size(nn); % define the size of matrix for all parameters

if ns>=3,error('This routine is for double inputs only. '),end
na=nn(1);nb=nn(2);nc=nn(3);nk=nn(4);nu=1;ne=nn(5);

if nk<1,error('Sorry, this routine requires nk>0; Shift input sequence if
necessary! '),end
d=na+nb+nc+1+ne; % define a parameter for the offset

if ns==1,nb=0;ne=0;end
if nb==0|ne==0,nk=1;end

tic=na+nb+1:na+nb+nc;

% create the array for i
iia=1:na;
iib=na+nk:na+nb+nk-1;
iic=na+nb+1:na+nb+nc;
iid=na+nb+nc+1;
iie=na+nb+nc+1+nk:na+nb+nc+1+ne+nk-1; % define for the second inputs

% create the array for ii
iia=1:na-1;
iib=na+1:na+nb-1;
iic=na+nb+1:na+nb+nc-1;
iid=na+nb+nc+1 % set indices for the offset
%iid=na+nb+nc+1:na+nb+nc+1-1 % set indices for the offset
iie=na+nb+nc+1+1:na+nb+nc+1+ne-1; % for the second input

ii=[iia iib iic iid iie]
i=[iia iib iic iid iie]
sizeofiii=size(i);

dm=na+nb+nc+1+ne;

% initialise phi
phi=zeros(dm,1);

dphi4y=zeros(1,na);

[modt,columnofth]=size(thm)

if d~=columnofth
error('these two must be the same')
end

% decide which range of u is used, result in the value of j,j decide which
model.
low=threshold(1:modt)
sizeoflower=size(low);

high=threshold(2:modt+1)
sizeofhigher=size(high);

%threshold=sort(threshold)
count=zeros(1,modt); % initialization

for kcou=1:nz % start the loop for estimator
% define the threshold index,i.e., where the threshold is
indexthreshold=find((u1(kcou)>=low)&(u1(kcou)<=high));

```

```

if ~isempty(indexthreshold)
    modn=min(indexthreshold);           % make sure there is only one index

    count(modn)=count(modn)+1;         % find number of samples on each model

    th=thm(modn,1:d)';                 % redefine the parameters

    % increase the size for the offset
    phi(iiid)=1;

    % the operation for the delta operator
    phi4y=phi(iiia);
    dphi4y = delta4y(phi4y',Ts);
    sizeofphi4y=length(dphi4y);

    phi4u=phi(iiib);
    dphi4u = delta4y(phi4u',Ts);
    sizeofphi4u=length(dphi4u);

    phi(iiic)=0;
    dphi4c = delta4y(phi(iiic)',Ts);
    sizeofphiic=size(phi(iiic)');

    dphi4d = delta4y(phi(iiid)',Ts);
    sizeofphiid=size(phi(iiid)');

    phi4e=phi(iiie);
    dphi4e = delta4y(phi4e',Ts);
    sizeofphi4e=length(dphi4e);

    %dphi = [dphiy dphiu dphi4c dphi4d dphie]'
    dphi = [-dphi4y dphi4u dphi4c dphi4d dphi4e]';

    dyh=dphi(i) '*th;

    % fully undeltarise prediction error in order to achieve the
    yh=undelta([dyh -dphi(iiia)'],Ts,0);

    % shifting procedure
    phi(ii+1)=phi(ii);
    phi(iid)=1;

    % Assign the output to the vector array
    phi(1)=yh;
    phi(na+1)=u1(kcou);
    phi(na+nb+1)=0;
    phi(na+nb+nc+1)=1;
    phi(na+nb+nc+1+1)=u2(kcou);

    % store and undate these data
    thm(modn,1:d)=th';

    yhat(kcou)=yh;

    if isnan(yh)
        keyboard
    end
else
    fprintf('u(%g)=%g',kcou,u1(kcou));
    error('no model available!');
end
end                                     % end of for loop
yhat = yhat';

```



# ***Appendix I: Analytical Systems in MATLAB***

## ***I.1 Analytical System in Signal Processing Toolbox***

```
% load input signals from the PRBSG

load d:\likun\AfterTransferReport\AutomatedFaultModelGeneration\
softwareDesign\Library_PWP\MATLAB\Rarmax\Data\LeadLagNewfl159fl15krf10
kr410k.txt -ASCII

vout=LeadLagNewfl159fl15krf10kr410k;
LeadLagNewfl159fl15krf10kr410k=[];

% define the size of data, m,n are the row and column, respectively
[m,n]=size(vout);

% define the output-input data
u1=vout(:,1);
u2=vout(:,2);
u=[u1 u2];
y=vout(:,3);
z=[y u1 u2];

% preprocessing for detection and deleting saturation data
znew=cleanSat(z);

ynew=znew(:,1);
unew1=znew(:,2)';
unew2=znew(:,3)';

unew=[unew1; unew2];
sizeOfunew=size(unew);

% define the size of new output data sets
[zewrow,zewcolumn]=size(znew);

intervals=[(zewrow-10000):zewrow];

% define the System
% define the sampling rate
t=1:zewrow;
T_s=140.0E-6;
f_s=1/T_s;

ts=T_s*t;
sizeOfTs=size(ts);

% define the low pass filter for vip
f_lpvip=10;
R3=10.0E+3;
C4=1/(2*pi*R3*f_lpvip);

% define the low pass filter for vin
```



```

f_lpvin=f_lpvip*10;
Rf=10.0E+3;
Cf=1/(2*pi*Rf*f_lpvin);

R1=Rf/10;

% define coefficients for the system coefficients for vin
dd=R1*Rf*Cf;
dd1=Rf*C4*R3;

% coefficients for vip
ee=R1*Rf*Cf*C4*R3;
%ee1=(R1+Rf)*Cip*R4;
ee1=R1+Rf;
ee2=dd+R1*R3*C4;

% define the transfer functions for two inputs with
% common denominators
h1=tf([-dd1 -Rf],[ee ee2 R1]);
h2=tf([dd ee1],[ee ee2 R1]);

% define the two systems in continuous time
sysc=[h1 h2];

% obtain the output data with the lsim function
sizeOfsysc=size(sysc);
sizeOfunew=size(unew);

ysim=lsim(sysc,unew,ts);
sizeOfy=size(y);

% compare the original signal with the generated signal
subplot(2,1,1); plot(ynew(intervals));
subplot(2,1,2); plot(ysim(intervals));

% convert the Laplace transform into z transform
sysd=c2d(sysc,T_s,'zoh')

% write out these signals
sizeOfysim=size(ysim)
sizeOfunew1=size(unew1')
sizeOfunew2=size(unew2')

znew=[ unew1' unew2' ysim];
sizeOfznew=size(znew);

fid =
fopen('d:\likun\AfterTransferReport\AutomatedFaultModelGeneration\soft
wareDesign\Library_PWP\MATLAB\Rarmax\Data\prbs42inputsLeadLagNewfilter
test2.txt ','w');

fprintf(fid,'% -20.10f % -20.10f % -20.10f\n',znew');
fclose(fid);

```

## ***1.2 Analytical System in System Identification Toolbox***

```

% loading input signals from the PRBSG

```

```

load e:\likun\ AfterTransferReport\ AutomatedFaultModelGeneration\
softwareDesign\Library_PWP\MATLAB\Rarmax\Data\LeadLagNewfl159fl15krf10
kr410k.txt -ASCII

vout=LeadLagNewfl159fl15krf10kr410k;
LeadLagNewfl159fl15krf10kr410k=[];

% define the size of data, m,n are the row and column, respectively
[m,n]=size(vout);

% define the output-input data
u1=vout(:,1);
u2=vout(:,2);
u=[u1 u2];
y=vout(:,3);
z=[y u1 u2];

% preprocessing for detection and deleting saturation data
znew=cleanSat(z);

ynew=znew(:,1);
unew1=znew(:,2)';
unew2=znew(:,3)';

unew=[unew1; unew2];
sizeOfunew=size(unew);

% define the size of new output data sets
[zewrow,zewcolumn]=size(znew);

intervals=[(zewrow-10000):zewrow];

% define the System
% define the sampling rate
t=1:zewrow;
T_s=140.0E-6;
f_s=1/T_s;

ts=T_s*t;
sizeOfTs=size(ts);

% define the low pass filter for vip
f_lpvip=10;
R3=10.0E+3;
C4=1/(2*pi*R3*f_lpvip);

% define the low pass filter for vin
f_lpvin=10*f_lpvip;
Rf=10.0E+3;
Cf=1/(2*pi*Rf*f_lpvin);

R1=Rf/10;
% define coefficients for the system coefficients for vin
dd=1/(R1*Cf);
ddl=1/(R1*Cf*C4*R3);

ff1=1/(R3*C4);
ff2=(R1+Rf)/(R1*Rf*Cf*C4*R3);

ee1=(1/(Rf*Cf))+ff1;
ee2=1/(Rf*Cf*R3*C4);

```

```

% define the transfer functions for two inputs with
% common denominators
h1=[-dd -dd1; ff1 ff2];
h2=[1 ee1 ee2;1 ee1 ee2];
m = idpoly(1,h1,1,1,h2,1,0);
sysd= c2d(m,T_s)
sizeOfsysd=size(sysd);
y = sim(sysd,[unew1' unew2']);

% redesign the relationship between inputs and output
znew=[unew1' unew2' y];
sizeOfznew=size(znew);

% open a file and then write signals into the file and then close it
fid =
fopen('e:\likun\AfterTransferReport\AutomatedFaultModelGeneration
\softwareDesign\Library_PWP\MATLAB\Rarmax\Data\prbs42inputsLeadLagNewf
iltertest3.txt','w');

fprintf(fid,'% -20.10f % -20.10f % -20.10f\n',znew');
fclose(fid);

```

# ***Appendix J: The AME System Validation Using the HDL Simulator***

## ***J.1 Introduction***

It is known that the models from the MMGSD are used for the analogue simulation, so in this section we validate the MMGSD using a known model implemented in SystemVision. The process comprises two steps:

1. The analogue simulation is implemented using a linear model. Both input data and output data are stored in a text file.
2. The MMGSD generates the model based on these data

The model used is a linear model given in Eq. J-1.

$$V_o = \frac{-(20s + 500)V_{in} + (10s + 250)V_{ip} + 250V_{offset}}{s^2 + 20s + 500} \quad \text{Eq. J-1}$$

The stimulus is a 0.2V, 100Hz triangle waveform with a 0.05V, 100kHz PRBS superimposed on it for the inverting input, the second input is a similar signal but with lower amplitude and frequency for the non-inverting input displayed in Figure 8-5 (14,000 samples).

This section consists of two subsections: J.2 introduces the analogue simulation. The estimator is tested in subsection J.3.

## ***J.2 Analogue Simulation***

### ***J.2.1 VHDL-AMS Model Structure***

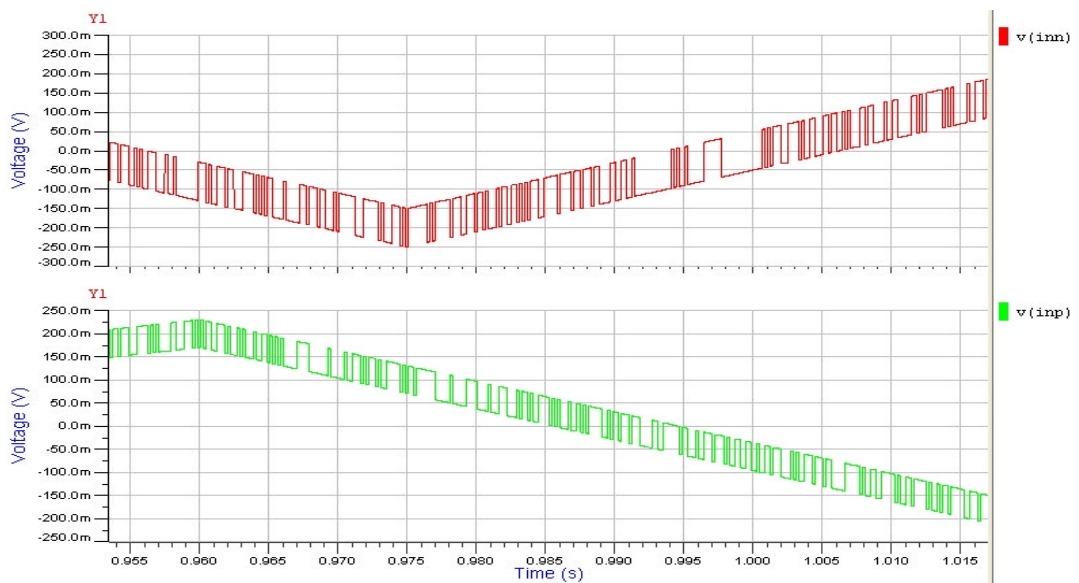
The linear model in Eq. J-1 is converted into a VHDL-AMS model in Eq. J-2 using the attribute 'ltf', which is for a Laplace transfer function [Ashenden03].

$$v_o == v_{in} 'ltf(num\_one,den) + v_{i_p} 'ltf(num\_two,den) + v_{offset} 'ltf(num\_three,den)$$

where  $num\_one$  includes coefficients (-20 -500) of  $v_{in}$ ,  $num\_two$  has coefficients (10 250) of  $v_{ip}$ ,  $num\_three$  is 250 for  $v_{offset}$ , and  $den$  includes the output coefficients (1 20 500) of  $v_o$ .

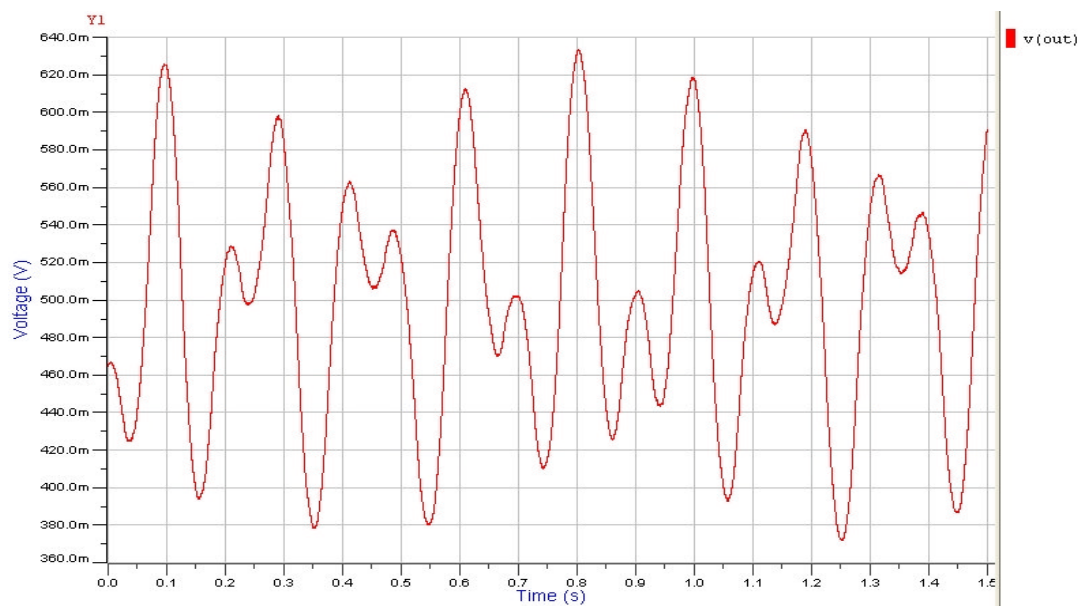
### J.2.2 Output Signal Using Data Loading and Writing Functions

We use the data loading function mentioned in section 6.2.2 to load the triangle PRBS as stimuli, which is shown in Figure J-1 by the analogue simulator SystemVision.



**Figure J-1:** The triangle PRBS

The output signal is shown in Figure J-2:

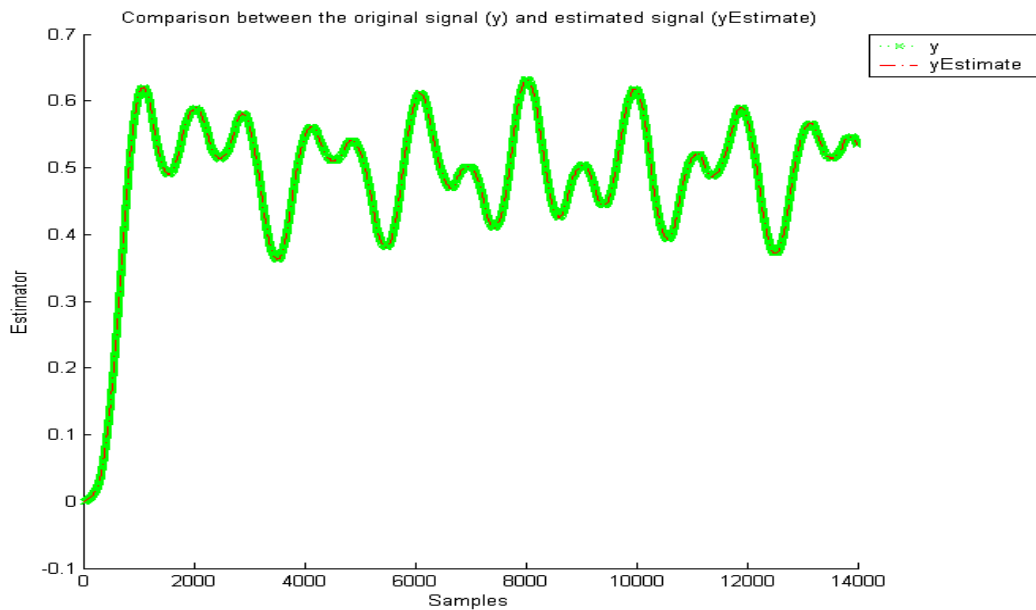


**Figure J-2:** The output signal

Both inputs and output signals are be written to a text file by using data writing process discussed in section 6.2.2.

### J.3 Test for the AME system

The AME system generates a model based on the data in the text file. The estimated signal is illustrated in Figure J-3:



**Figure J-3:** The estimated signal

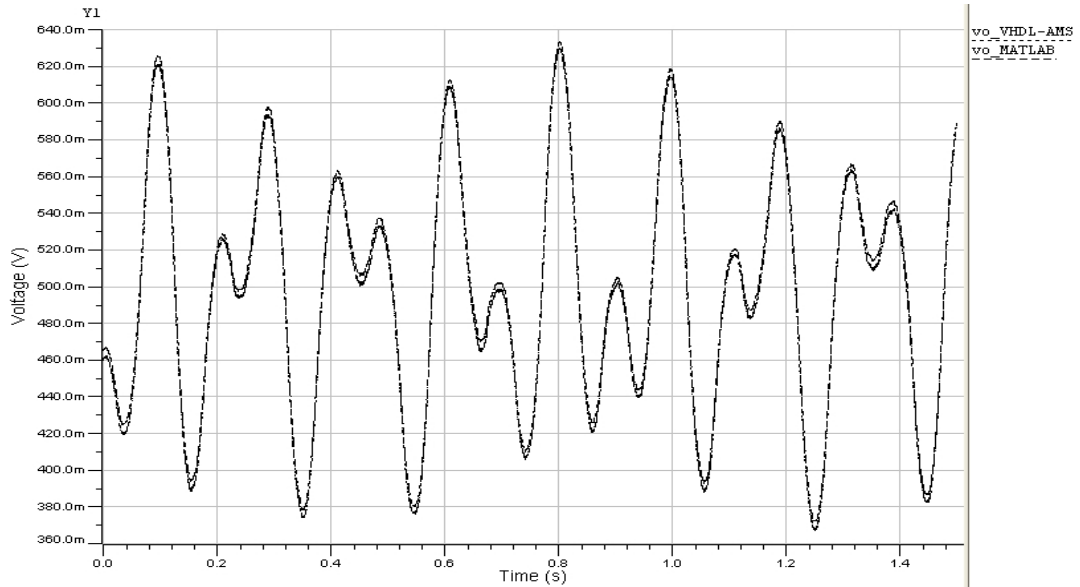
It is seen that the signal from the analogue simulator  $y$  matches the estimated signal  $yEstimator$ . The average difference between two signals is  $5.21e-8\%$  using Eq. 5-2.

During the simulation the coefficients are obtained as shown in Eq. J-3.

$$V_o = \frac{-(20.16s + 518.22)V_{in} + (10.08s + 259.095)V_{ip} + 250.259V_{offset}}{s^2 + 20.969s + 524.716} \quad \text{Eq. J-3}$$

It is shown this model is close to the linear model in Eq. J-1 by comparing the coefficients.

To prove the model generated is correct, we convert it into a VHDL-AMS model for analogue simulation using the same stimuli in Figure J-1. Signals are illustrated in Figure J-4:



**Figure J-4:** The predicted signal in the analogue system

It is seen that the predicted signal in VHDL-AMS (*vo\_VHDL\_AMS*) matches the original signal (*vo\_MATLAB*) in terms of amplitude and shape.

# ***Appendix K: Comparison of Various HDLs and Simulators***

## ***K.1 Introduction***

With the development of hardware description languages (HDLs) based on design methodologies, IC design and simulation have become simpler. For the digital IC design HDLs including VHDL, Verilog, etc. are already well established. Many books have been published to assist them such as [Zwo2000]. Unfortunately, for analogue and mixed-signal design, textbooks or references on HDLs such as MAST [Saber04], Verilog-AMS, SpectreHDL [Spec97] and VHDL-AMS [Ashenden03] are limited. [Ashenden03] is one of few books to introduce VHDL-AMS in detail, others may only supply part of information based on a particular system (e.g. [Getreu93], [Nikitin07], [Pecheux05]).

A number of commercial simulators for HDLs are currently available from several electronic design automation (EDA) companies. They include Cosmos in Saber from Synopsys [Synopsys], Virtuoso AMS Designer from Cadence [Cadence], Simplorer from Ansoft [Ansoft], SMASH from Dolphin [Dolphin], SystemVision from Mentor Graphics [Mentor], etc. Each of them has individual features in terms of simulation speed, ease of use. In this chapter we briefly compare two types of HDLs (MAST, VHDL-AMS) and simulators (Cosmos, Smash, SystemVision) based on a behavioural operational amplifier (op amp) model.

The objective of the section is to compare two or three HDLs and individual simulators using a linear model in terms of complexity and accuracy.

The following section is outlined: in section K.2 different HDLs are introduced; multiple simulators for these HDLs are presented in section K.3. In section K.4 high level modelling is implemented. The conclusion is supplied in section K.5.



## ***K.2 Brief Introduction to Different HDLs***

In this subsection, MAST and VHDL-AMS are introduced. Since 1987, MAST has been enhanced to include mixed-signal modelling constructs such that it is able to improve simulation speed and support top-down design methods for analogue and mixed signal designs [Saber04]. MAST language files use the *.sin* extension. More information can be found in [Saber04].

In 1999, the analogue and mixed signal (AMS) extension to VHDL (VHSIC Hardware Description Language) was standardized as VHDL-AMS [SMASHR05]. It inherits all advantages from VHDL, handles several levels of design hierarchy and provides behavioural modelling capability for both digital and analogue systems [Frey98]. VHDL-AMS files usually have *.vhd* or *.vhdl* extensions.

## ***K.3 Introduction to Different Simulators***

This subsection introduces three simulators: SMASH, SystemVision and Cosmos in Saber.

### **K.3.1 Introduction to SMASH**

The SMASH simulator requires two mandatory files: the netlist file abbreviated as *.nsx*, and a pattern file *.pat*. The former contains the circuit description. The latter is associated with but separated from the netlist file and provides the stimulus descriptions and simulation directives. These two files must have the same name and be located in the same directory. In the netlist file, various languages such as Spice, Verilog, as well as VHDL are allowed to describe both the analogue part and the logic part, at any level of refinement from behavioural to transistor level [SMASHU05]. Each part must be preceded by a specific language identifier `>>>`, which allows the switch from one language to another such as `>>> VERILOG` for Verilog, `>>> SPICE` for Spice and `>>> VHDL` for VHDL. Library files resulting from model compilations are stored in a work directory created by SMASH at the same level as netlist and pattern files. Note: in the pattern file the order of multiple specifications of this directive is important. If the compilation order does not correspond to the order required by the VHDL source files, an error message will be displayed [SMASHR05]. This pattern file comprises directories of all components from the netlist. The name of the entity and optionally of the corresponding architecture or configuration at the top level is also indicated.

### K.3.2 Introduction to SystemVision

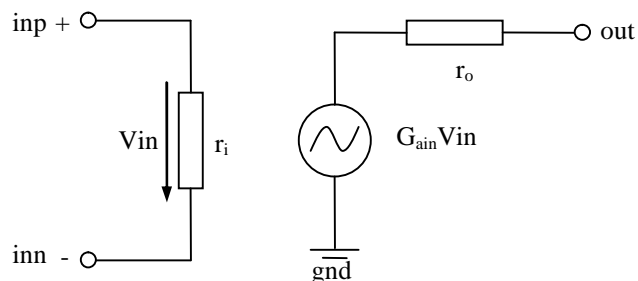
The SystemVision simulator provides a virtual lab for creating and analyzing analogue, digital and mixed-signal systems and allows design verification of hierarchical schematic and circuit elements. Industry-standard languages such as VHDL-AMS, SPICE, and C are supported by this software. Moreover, this simulator provides concept verification through block diagrams and transfer functional blocks [SystemVision]. There is a large built-in library in SystemVision, so the user is able to select many models or even schematics for any design. The nestlist file needs a project that prefers to be separated from library project.

### K.3.3 Introduction to Cosmos in Saber

Cosmos is the simulator for the MAST language in *Saber*. Similar to other two simulators it can handle analogue and mixed mode circuitry. It includes a large digital and analogue model library and allows different levels of modelling [Ana94]. The model in Cosmos can be described using a traditional netlist (the normal SPICE method) or in the differential equations directly [Saber04]. The simulator can recognise the SPICE signals extension such as *.tr0* for transient simulation and displays them. *SystemVision* can not display these signals even though it is able to recognise these extensions because of different configuration. Furthermore, *Saber* has interfaces with MATLAB and C libraries.

## K.4 Experimental Results

In this section, we compare both HDLs and simulators using a linear behavioural op amp) model shown in Figure K-1:



**Figure K-1:** The linear op amp

The model comprises two parts: the input stage has a resistor  $r_i$  representing the input impedance; the output stage consists of the gain  $G_{ain}$  and the resistor  $r_o$  representing the output impedance. The code in the MAST and VHDL-AMS are shown in Figure K-2 and Figure K-3, respectively.

```
#... behavioural model of an op amp
template opamp out inn inp gnd= gain, ri, ro
electrical out, inn, inp, gnd

#... parameters values
number gain = 50k
number ri = 1k
number ro = 1k
{
    val v vin, vip, vi, vout
    val i iR1, iRo

    values{
        #... define all connections
        vin = v(inn)-v(gnd)
        vip = v(inp)-v(gnd)
        vout = v(out)-v(gnd)
        #... equation between input and output
        vi = vin-vip
        #... define the current for the input resistor
        iR1 = vi/ri
        #... output current
        iRo = (gain*vi+vout)/ro
    }
    equations{
        #... current in input stage
        i(inn->inp)+= iR1
        #... current in output stage
        i(gnd->out)+= iRo
    }
}
```

**Figure K-2:** The op amp model written in MAST

```

library ieee;
use ieee.electrical_systems.all;
use ieee.math_real.all;
use work.all;

entity op is
    generic(gain,ri,ro: real);
    port(terminal inn,inp,outp: electrical); --interface ports
end entity op;

architecture opampb of op is
    quantity vo across io through outp to ground;
    quantity v_in across i_in through inn to inp;
begin
    i_in==v_in/ri;
    io==(vo+gain*v_in)/ro;
end architecture opampb;

```

**Figure K-3:** The model written in VHDL-AMS

This op amp is configured as an inverting amplifier with a gain of -4. The input stimulus is a sine wave with the amplitude of 0.1mV at 100Hz. The netlist structured in MAST and VHDL-AMS are shown in Figure K-4 and Figure K-5, respectively.

```

#... define the voltage source
v.sourceN inn0 0 = tran = (sin=(va=0.0001,f=100))

#... define value of R
r.Rin inn0 inn1 = 10k
r.Rf out1 inn1 = 40k

#... assign variables for the op amp
opamp.rc out1 inn1 0 0 = 500k, 50k, 100

```

**Figure K-4:** The top level in MAST

```

-- .nsx file
>>> VHDL
library ieee;

```

```

use ieee.math_real.all;
use ieee.electrical_systems.all;

entity inverter is
    port (terminal inn,ou:electrical);
end inverter;

architecture behav of inverter is
    terminal innn:electrical;
    constant FREQ: real := 1.0e2;
    constant ampt: real := 0.0001;

    quantity V across inn to ground;
    quantity vo across io through ou to ground;
begin
    V == ampt*sin(MATH_2_PI*FREQ*NOW);

    opamp_behav: entity work.op(opampb)
generic map (gain=>5.0e+5, ri=>5.0e4, ro=>100.0)
port map (inn=>innn,inp=>ground, outp=>ou);

    rin: entity work.resistor(behav_r)
        generic map (r=>1.0e+4)
        port map (p1=>inn, p2=>innn);

    rf: entity work.resistor(behav_r)
        generic map (r=>4.0e+4)
        port map (p1=>innn, p2=>ou);
end behav;

-- .pat file
.VHDL set kind=ams
.VHDL compile library=work source=e:/VHDL-AMS/opamp.vhdl
.VHDL compile library=work source=e:/VHDL-AMS/resistor.vhdl
.VHDL elaborate entity=inverter unit=behav

.Eps 1m 100m 500
.Tolerance DEFAULT_TOLERANCE 100m
.H 100fs 100fs 10ns 125m 2
.Tran 100ps 10ms 1.5us noise=no noisestep=10ns traceBreak=yes

```

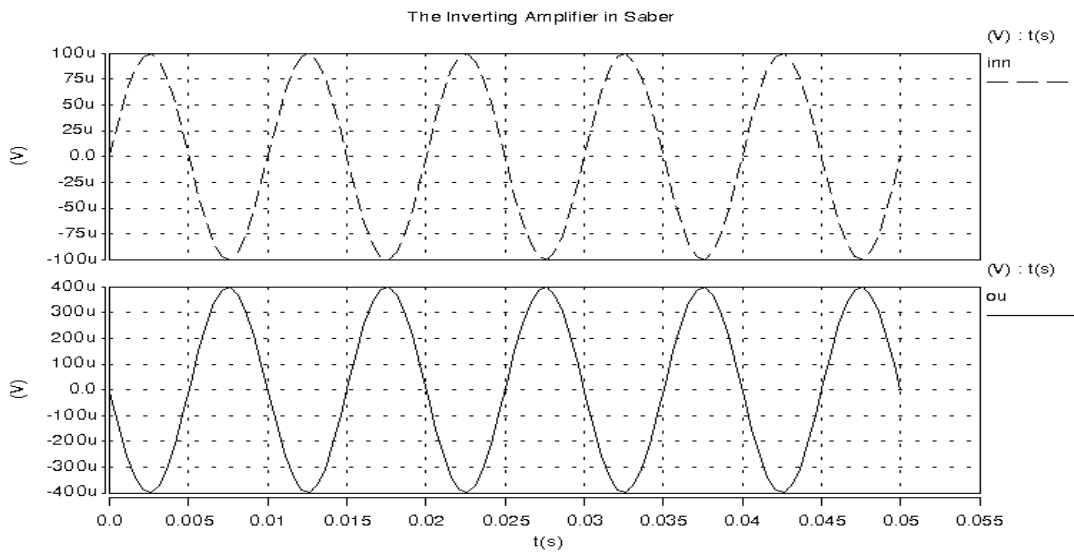
```

.Method BDF sync=lockstep current=yes global=yes
.Trace Tran INN_REFERENCE Min=-7.2000E-004 Max=7.2000E-004
.Trace Tran OU_REFERENCE Min=2.4940487E+000 Max=2.5110145E+000

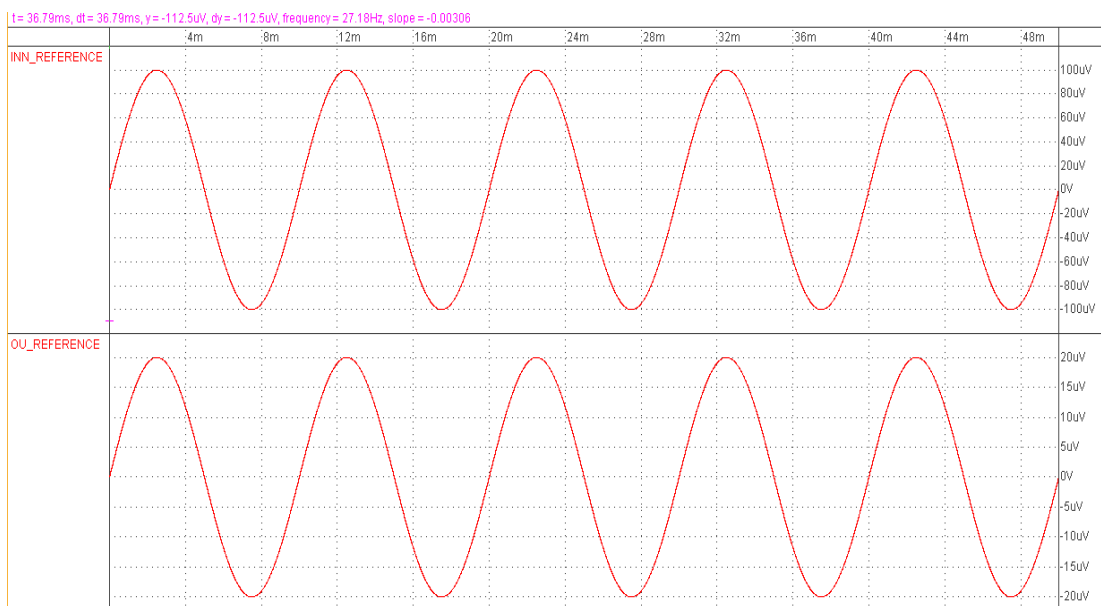
```

**Figure K-5: VHDL-AMS top level in SMASH**

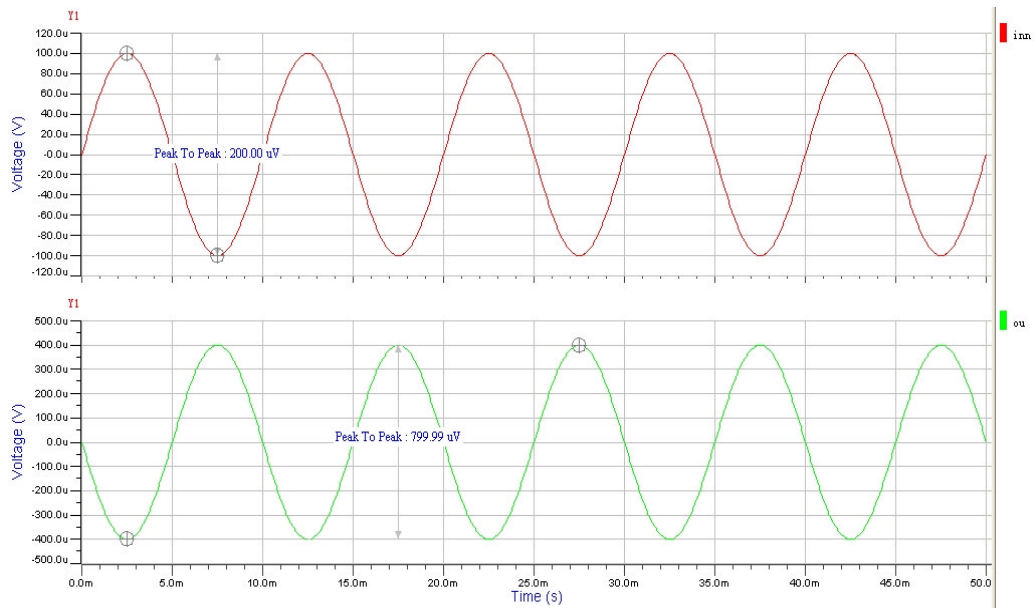
The signal produced by MAST in Saber simulator is shown in Figure K-6; signals created by VHDL-AMS in both SMASH and SystemVision simulators are depicted in Figure K-7 and Figure K-8, respectively.



**Figure K-6: The output signal from Cosmos**

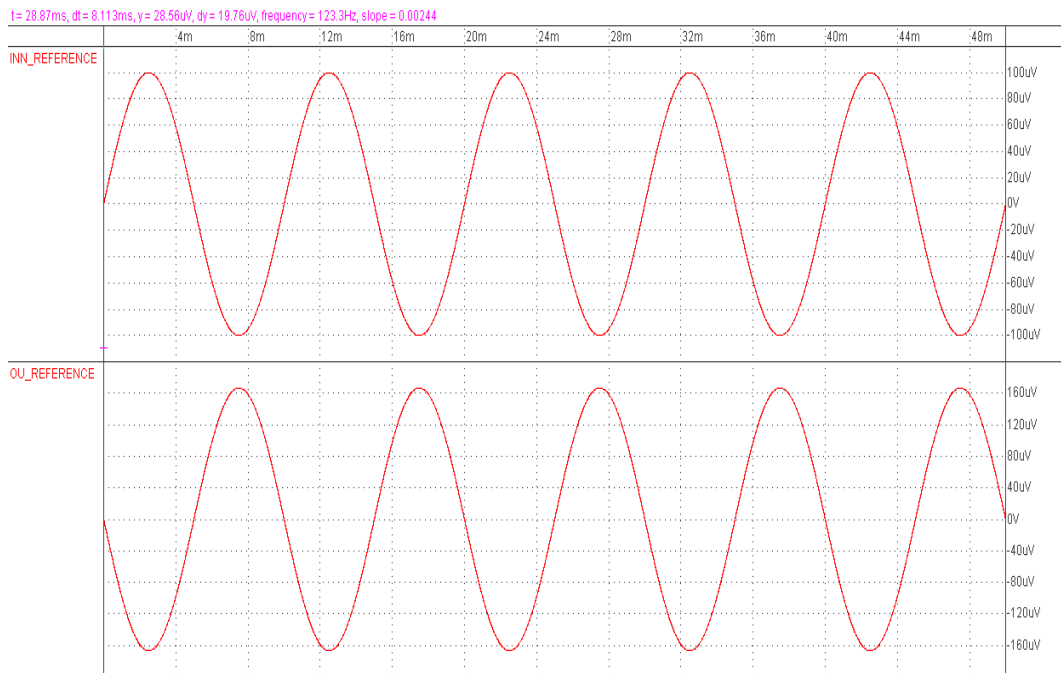


**Figure K-7: The signal in VHDL-AMS from SMASH**



**Figure K-8:** The signal in VHDL-AMS from SystemVision

It is seen that with the equivalent set up both Figure K-6 and Figure K-8 supply the correct solutions. However, in Figure K-7 the correct output signal does not have the correct amplitude. One way to improve it is to change the input resistance  $r_i$  to  $15k\Omega$ . The signal is plotted in Figure K-9:



**Figure K-9:** The signal from SMASH with  $r_i=15k\Omega$

Furthermore, when the gain of amplifier is changed to -5, with the SMASH simulator the correct results are not obtained until the parameters  $r_i$  is changed again. With other simulators the correct response can be achieved.

It is seen that the structure of the model in the MAST language is more complex than VHDL-AMS because the latter does not require many sections. Moreover, it provides attributes such as *'slew* for slew rate and *'zoh* for sampling and hold, but in MAST *'zoh* has to be defined in the *when* section, in which the statements such as *schedule\_event* are used. Additionally in SMASH there is not a way to *export* signals, the only way to obtain them is to use the *Prt Sc* key on the keyboard, whereas the Saber and SystemVision simulators can save signals with the *export* option.

In Saber the schematics design entry is provided by the SaberSketch software that is separated from Cosmos, whereas both SMASH and SystemVision can handle HDLs and schematics. However, in SystemVision a projector is required for each simulation, whereas the Saber simulator does not need this.

### ***K.5 Conclusion***

In this section two popular HDLs are reviewed: MAST and VHDL-AMS, and simulators for them: Cosmos in Saber, SMASH and SystemVision are compared in terms of accuracy and convenience using a linear op amp model. Results show that structure of VHDL-AMS language is simpler than MAST. Cosmos and SystemVision can achieve accurate solution more easily than SMASH.



# **Appendix L: The RML Estimation Algorithm Updates Equations for Both $z$ and delta Transforms**

## ***L.1 Estimation in $z$ Transform***

$$\begin{aligned}\varepsilon(t) &= y(t) - \varphi^T(t)\theta(t-1) \\ C^{t-1}(z^{-1})\varphi^T(t) &= \psi(t) \\ L(t) &= \frac{P(t-1)\varphi(t)}{\lambda(t) + \varphi^T(t)P(t-1)\varphi(t)} \\ P(t) &= \frac{1}{\lambda(t)} \left[ P(t-1) - \frac{P(t-1)\varphi(t)\varphi^T(t)P(t-1)}{\lambda(t) + \varphi^T(t)P(t-1)\varphi(t)} \right] \\ \theta(t) &= \theta(t-1) + L(t)\varepsilon(t) \\ \bar{\varepsilon}(t) &= y(t) - \varphi^T(t)\theta(t)\end{aligned}$$

where,

$\varepsilon(t)$  is the innovation error sequence.

$\bar{\varepsilon}(t)$  is the residual error sequence.

$P(t)$  is the covariance matrix.

$L(t)$  is the gain vector.

Estimator Observation Vector

$$\psi(t) = [-y(t-1) \dots -y(t-na), u(t-1) \dots u(t-nb), \bar{\varepsilon}(t-1) \dots \bar{\varepsilon}(t-nc), 1, u(t-1) \dots u(t-nb)]$$

Estimated Parameter Vector

$$\theta(t) = [a_1 \dots a_{na}, b_1 \dots b_{nb}, c_1 \dots c_{nc}, d, f_1 \dots f_{ne}]^T$$

Define the  $c$  polynomial for the prefilter

$$c^t(z^{-1}) = 1 + c_1^t z^{-1} + \dots + c_{nc}^t z^{-nc}$$

Pre-whitened Estimator Observation Vector

$$\varphi(t) = \begin{bmatrix} -y(t-1)/c^{t-1}(z^{-1}) \\ \vdots \\ -y(t-na)/c^{t-1}(z^{-1}) \\ u(t-1)/c^{t-1}(z^{-1}) \\ \vdots \\ u(t-nb)/c^{t-1}(z^{-1}) \\ \bar{\varepsilon}(t-1)/c^{t-1}(z^{-1}) \\ \vdots \\ \bar{\varepsilon}(t-nc)/c^{t-1}(z^{-1}) \\ 1/c^{t-1}(z^{-1}) \\ v(t-1)/c^{t-1}(z^{-1}) \\ \vdots \\ v(t-ne)/c^{t-1}(z^{-1}) \end{bmatrix} = \begin{bmatrix} -y^C(t-1) \\ \vdots \\ -y^C(t-na) \\ u^C(t-1) \\ \vdots \\ u^C(t-nb) \\ \bar{\varepsilon}^C(t-1) \\ \vdots \\ \bar{\varepsilon}^C(t-nc) \\ 1^C \\ v^C(t-1) \\ \vdots \\ v^C(t-ne) \end{bmatrix}$$

## L.2 Estimation in Delta Transform

$$\begin{aligned} \varepsilon(t) &= \delta^{na} y(t) - \varphi^T(t)\theta(t-1) \\ C^{t-1}(\delta)\varphi^T(t) &= \psi(t) \\ L(t) &= \frac{P(t-1)\varphi(t)}{\lambda(t) + \varphi^T(t)P(t-1)\varphi(t)} \\ P(t) &= \frac{1}{\lambda(t)} \left[ P(t-1) - \frac{P(t-1)\varphi(t)\varphi^T(t)P(t-1)}{\lambda(t) + \varphi^T(t)P(t-1)\varphi(t)} \right] \\ \theta(t) &= \theta(t-1) + L(t)\varepsilon(t) \\ \bar{\varepsilon}(t) &= \delta^{na} y(t) - \varphi^T(t)\theta(t) \end{aligned}$$

where,

$\varepsilon(t)$  is the innovation error sequence.

$\bar{\varepsilon}(t)$  is the residual error sequence.

$P(t)$  is the covariance matrix.

$L(t)$  is the gain vector.

The deltarise function and undeltarise function are necessary to update other vectors. Examples for both are given.

### L.2.1 Deltarise Function

The deltarise function is used to obtain deltarised value using the delta operator given in Eq. L-1, where delta ( $\delta$ ) is related to both the present and future values,  $T_s$  is the sampling rate,  $q$  is the forward shift operator used to describe discrete models, which is shown in Eq. L-2.

$$\delta = \frac{q - 1}{T_s} \cong \frac{d}{dt} \quad \text{Eq. L-1}$$

$$qx_k = x_{k+1} \quad \text{Eq. L-2}$$

The equivalent form of Eq. L-2 is given in Eq. L-3, the relationship between  $\delta$  and  $q$  is a simple linear function, so  $\delta$  can offer the same flexibility in the modelling of discrete-time systems as  $q$  does.

$$\delta x_k = \frac{x_{k+1} - x_k}{T_s} = \frac{x(kT_s + T_s) - x(kT_s)}{T_s} \cong \frac{dx}{dt} \quad \text{Eq. L-3}$$

The use of delta operator and its relationship is illustrated in the following example. Imagine there is a vector array for  $y$ , see in Eq. L-4. Initially each vector is subtracted from the one next to it seen in Eq. L-5, and is then divided by  $T_s$ , so the deltarised value is obtained, as shown in Eq. L-6. However, the last one highlighted by the rectangle is not involved in the calculation.

$$y(t) \quad y(t-1) \quad y(t-2) \quad \boxed{y(t-3)} \quad \text{Eq. L-4}$$

$$y(t-1) \quad y(t-2) \quad y(t-3) \quad \text{Eq. L-5}$$

$$\delta y(t-1) \quad \delta y(t-2) \quad \boxed{\delta y(t-3)} \quad \text{Eq. L-6}$$

$$\delta y(t-2) \quad \delta y(t-3) \quad \text{Eq. L-7}$$

To obtain  $\delta^2 y(t-3)$ , Eq. L-6 is subtracted by Eq. L-7, and then divided by  $T_s$ . The same procedure is used to obtain  $\delta^3 y(t-3)$ .

$$\delta^2 y(t-2) \quad \delta^2 y(t-3) \quad \text{Eq. L-8}$$

$$\delta^2 y(t-3) \quad \text{Eq. L-9}$$

$$\delta^3 y(t-3) \quad \text{Eq. L-10}$$

Therefore, the deltarised version of Eq. L-4 is obtained as shown in Eq. L-11.

$$\delta^3 y(t-3) \quad \delta^2 y(t-3) \quad \delta^1 y(t-3) \quad \delta^0 y(t-3) \quad \text{Eq. L-11}$$

The same procedure is also used for other vectors such as the inputs vectors  $u$ ,  $e$  and noise vector  $c$ . Delay is not included here.

### L.2.2 Undeltarise Function

This function is based on Eq. L-1 but with modification, that is,  $q = \delta T_s + 1$ , in order to model at the current time. An example is also used to demonstrate this reverse algorithm. It is a model in delta transform, but only output vectors  $y$  are shown in Eq. L-12. Firstly each vector, except for last one highlighted by the rectangle because it is already undeltarised, is multiplied by  $T_s$  in Eq. L-13, and then adds the terms in Eq. L-14, so undeltarised vectors are obtained in Eq. L-15, i.e.,  $y(t-2)$  is obtained.

$$\delta^3 y(t-3) \quad \delta^2 y(t-3) \quad \delta^1 y(t-3) \quad \delta^0 y(t-3) \quad \text{Eq. L-12}$$

$$T_s \delta^3 y(t-3) \quad T_s \delta^2 y(t-3) \quad T_s \delta^1 y(t-3) \quad \text{Eq. L-13}$$

$$+ \quad + \quad + \quad \text{Eq. L-14}$$

$$\delta^2 y(t-3) \quad \delta^1 y(t-3) \quad \delta^0 y(t-3)$$

$$\parallel \quad \parallel \quad \parallel$$

$$\delta^2 y(t-2) \quad \delta^1 y(t-2) \quad y(t-2) \quad \text{Eq. L-15}$$

To achieve  $y(t-1)$ , Eq. L-15 is multiplied by  $T_s$ , and then adds ones in Eq. L-17

$$T_s \delta^2 y(t-2) \quad T_s \delta^1 y(t-2) \quad \text{Eq. L-16}$$

$$+ \quad +$$

$$\delta^1 y(t-2) \quad \delta^0 y(t-2) \quad \text{Eq. L-17}$$

$$\parallel \quad \parallel$$

$$\delta^1 y(t-1) \quad y(t-1) \quad \text{Eq. L-18}$$

Finally  $y(t)$  is obtained using the same procedure as above.

$$T_s \delta^1 y(t-1) \quad \text{Eq. L-19}$$

+

$$y(t-1) \quad \text{Eq. L-20}$$

||

$$\boxed{y(t)} \quad \text{Eq. L-21}$$

Therefore, the undeltarised version of Eq. L-12 is obtained as shown in Eq. L-22.

$$y(t) \quad y(t-1) \quad y(t-2) \quad y(t-3) \quad \text{Eq. L-22}$$

The number of iterations is dependent on a variable *numb*. If a fully deltarisation is required, *numb* is set to 0, otherwise a number is selected. If the number is greater than the size of the vector array an error message is displayed.

After knowing the procedure for obtaining deltarise or undeltarise vectors, more details about vectors in RML are introduced.

Estimator Observation Vector

$$\psi(t) = [-\delta^{na-1}y(t) \dots -y(t), \delta^{nb-1}u(t) \dots u(t), \delta^{nc-1}\bar{\varepsilon}(t) \dots \bar{\varepsilon}(t), 1, \delta^{ne-1}v(t) \dots v(t)]$$

Estimated Parameter Vector

$$\theta(t) = [a_1 \dots a_{na}, b_1 \dots b_{nb}, c_1 \dots c_{nc}, d, f_1 \dots f_{ne}]^T$$

Define the *c* polynomial for the prefilter

$$c^t(\delta) = 1 + \delta^0 c_1^t + \dots + \delta^{nc-1} c_{nc}^t$$

Pre-whitened Estimator observation vector

$$\delta^{nc-1}\psi(t-nc) = \delta^{nc-1}\varphi(t-nc) - c_1\delta^{nc-2}\psi(t-nc) - \dots - c_{nc}\delta^0\psi(t-nc)$$

The relationship between delta  $\psi(t)$  and  $\varphi(t)$  is shown in Eq. L-2 when the size of vector array  $nn$  is set to [3 4 3 1 3].

$$\delta^2\psi(t-3) = \delta^2\varphi(t-3) - c_1\delta^1\psi(t-3) - c_2\delta^0\psi(t-3)$$

# References

[**Abra95**] J.A. Abraham, M. Soma, Mixed-Signal Test-Tutorial C, *The European Design and Test Conference*, 1995.

[**Aktouf05**] C. Aktouf, Why Haven't EDA Vendors Given Us DFT at the Register Transfer Level?, DeFacTo Technologies, <http://www.soccentral.com/results.asp?CatID=488&EntryID=13071> [Accessed on 12/08/08].

[**Allen87**] P.E. Allen, D.R. Holberg, *CMOS Analog Circuit Design* (Holt, Rinehart and Winston, Inc., 1987).

[**Ana94**] Oregon: Saber Reference Manual, *Analogy Inc. Beaverton*, release 3.3, 1994.

[**Ansoft**] <<http://www.ansoft.com>>.

[**Ashenden03**] P. J. Ashenden, G. D. Peterson and D. A. Teegarden, *The System Designer's Guide to VHDL-AMS* (Morgan Kaufmann Publishers, Elsevier Science, 2003).

[**Bartsch96**] E.K. Bartsch, I.M. Bell, High-Level Analogue Fault Simulation using Linear and Non-Linear Models, *Radioengineering*, 8(4), April 1996, 32-34.

[**Bartsch99**] E.K. Bartsch, *Improved Analogue Fault Simulation Through High Level Modelling*, MSc thesis, University of Hull, 1999.

[**Batra04**] R. Batra, P. Li, L.T. Pileggi, Yu-Tsun Chien, A Methodology for Analog Circuit Macromodeling, *BMAS*, 21-22 October, 2004, 41-46.

[**Bell96**] I.M. Bell, S.J. Spinks and J.Machado da Silva, Supply Current Test of Analogue and Mixed Signal Circuits, *IEEE Proceedings on Circuits Devices and Systems*, 143(6), December 1996, 399-407.

[**Bissell94**] C.C. Bissell, *Control Engineering* (Chapman & Hall, 1994).

[**Boyle74**] G.R. Boyle, D.O. Pederson, B.M. Cohn, and J.E. Solomon, Macromodeling of Integrated Circuit Operational Amplifiers, *IEEE Journal of Solid State Circuits*, 9(6), Dec. 1974, 353-67.

[**Bratt95**] A.H. Bratt, A.M.D. Richardson, R.J.A. Harvey, A.P. Dorey, A Design-For-Test Structure for Optimising Analogue and Mixed Signal IC Test, *Proceedings of the European Design and Test Conference (ED&TC)*, Paris, March 1995, 24-33.

[**Breiman96**] L. Breiman, Stacked Regression, *Machine Learning*, 24(1), 1996, 49-64.

[**Broyden65**], C.G. Broyden, A class of methods for solving nonlinear simultaneous equations, *Mathematics of Computation*, 19, 577-593, 1965.

[**Burden85**] R.L. Burden and J.D. Faires, *Numerical Analysis* (Prindle, Weber and Schmidt, 1985).

[**Cadence**] <[http://www.cadence.com/products/cusmom\\_ic/ams\\_designer](http://www.cadence.com/products/cusmom_ic/ams_designer)>.

[**Caunegre95**] P. Caunegre, C. Abraham, Achieving Simulation-Based Test Program Verification and Fault Simulation Capabilities for Mixed-signal Systems, *Proceedings of the European Design and Test Conference (ED&TC)*, 1995, 469-477.

[**Chang00**] Y.J. Chang, C.L. Lee, A Behaviour Level Fault Model for the Closed-Loop Operational Amplifier, *Journal of Information Science and Engineering*, 16, 2000, 751-766.



[**Chiprout94**] E. Chiprout and M.S. Nakhla, *Asymptotic Waveform Evaluation and Moment Matching For Interconnect Analysis* (Kluwer Academic Publisher, Norwell, MA, 1994).

[**Chirlian82**] P.M. Chirlian, *Analysis and Design of Integrated Electronic Circuits* (Harper & Row Ltd., 1981).

[**DataSheet93**] Saber-Industry Standard for Multi-Technology and Mixed-Signal Simulation, *Synopsys*, 2003.

[**Davalo91**] E. Davalo, P. Naïm, *Neural Networks* (Macmillan Education Ltd., 1991).

[**Dolphin**] <[http://www.dolphin.fr/medal/smash/smash\\_overview.html](http://www.dolphin.fr/medal/smash/smash_overview.html)>.

[**Dong03**] N. Dong and J. Roychowdhury, Piecewise Polynomial Nonlinear Model Order Reduction, *Proceedings Design Automation Conference*, 2003, 484-489.

[**Dong04**] N. Dong and J. Roychowdhury, Automated extraction of broadly applicable nonlinear analog macromodels from SPICE-level descriptions, *CICC*, 2004, 117-120.

[**Dong05**] N. Dong and J. Roychowdhury, Automated Nonlinear Macromodelling of Output Buffers for High-Speed Digital Applications, *DAC*, 2005, 51-56.

[**Elias79**] N.J. Elias, The Application of Statistical Simulation to Automated Analog Test Development, *IEEE Trans. on Circuits and Systems*, 26(7), 1979, 513-517.

[**Fang01**] L. Fang, F. Fronthoud, H.G. Kerkhoff, Reducing Analogue Fault-Simulation Time by Using High-Level Modelling in Dotss for an Industrial Design, *Proceedings of the IEEE European Test Workshop*, 2001, 61.

[**Feldmann95**] P. Feldmann and R.W. Freund, Efficient linear circuit analysis by Padé approximation via the Lanczos process, *IEEE Trans. CAD*, 14(5), May 1995, 639-649.

**[Ferguson88]** F. Joel Ferguson, F. P. Shen, Extraction and Simulation of Realistic CMOS Faults using Inductive Fault Analysis, *Proceeding of International Test Conference*, 1988, 475-484.

**[Frey98]** P. Frey, K. Nelayappan, V. Shanmugasundaram, R.S. Mayiladuthurai, C.L. Chandrashekar, H.W. Carter, SEAMS: Simulation Environment for VHDL-AMS, *proceeding of simulation conference, 1*, 1998, 539-546.

**[Getreu93]** I. Getreu, D. Teegarden, An Introduction to Behavioural Modelling, *Microelectronics Journal*, 24(7), 1993, 708-716.

**[Gielen05]** G. Gielen, T. McConaghy, T. Eechelaert, Performance Space Modeling for Hierarchical Synthesis of Analog Integrated Circuits, *DAC*, June 2005, 881-886.

**[Grimme97]** E.J. Grimme, *Krylov Projection Methods for Model Reduction*, doctoral thesis, University of Illinois, USA, 1997.

**[Grout00]** I.A. Grout and K. Keane, A Matlab to VHDL conversion toolbox for digital control, *IFAC Symposium on Computer Aided Control Systems Design (CACSD)*, Salford, UK, 11th – 13th September 2000, 13-18.

**[Grout01]** I.A. Grout, Modelling, simulation and synthesis: From Simulink to VHDL generated hardware, *Systemics, Cybernetics and Informatics (SCI)*, Orlando, Florida, USA, July 22<sup>nd</sup>-25<sup>th</sup>, 2001, 443-448.

**[Grout04]** I.A. Grout, C. Wegener and M.P. Kennedy, Reducing Fault Simulation Effort by Sensitivity Analysis of Circuit Structure, *IEEE IC Test Workshop*, Limerick, Ireland, 13-14 September 2004.

**[Grout05]** I.A. Grout, J. Ryan and T. O'Shea, Configuration and debug of field programmable gate arrays using MATLAB/SIMULINK, *Journal of Physics: Conference Series*, 15, 2005, 244–249.

[Harvey95] R.J.A. Harvey, A.M.D. Richardson, H.G. Kerkhoff, Defect Oriented Test Development Based on Inductive Fault Analysis, *IEEE Int'l Mixed Signal Testing Workshop*, Grenoble, 1995, 2-9.

[Healy05] J.T. Healy, The New DFT at 90nm, LogicVision, Inc., <http://www.soccentral.com/results.asp?CatID=488&EntryID=13086> [Accessed on 12/08/08].

[Hong03] X. Hong, P.M. Sharkey, K. Warwick, A Robust Nonlinear Identification Algorithm Using PRESS statistic and Forward Regression, *IEEE Trans. Neural Networks*, 14(2), March 2003, 454-458.

[Hsu04] C. Hsu, Control and Observation Structure for Analog Circuits with Current Test Data, *J. of Electronic Testing*, 20, 2004, 39-44.

[Huang03] X. Huang, C.S. Gathercole, and H.A. Mantooth, Modeling nonlinear dynamics in analog circuits via root localization, *IEEE Trans. CAD*, 22(7), July 2003, 895-907.

[Jaworski97] Z. Jaworski, M. Niewczas and W. Kuzmich, Extension of Inductive Fault Analysis to Parametric Faults in Analog Circuits with Application to Test Generation, *IEEE VLSI Test Symposium (VTS)*, 1997, 172-176.

[Joannon06] Y. Joannon, V. Beroulle, R. Khouri, C. Robach, S. Tedjini and J. Carbonero, Behavioral Modeling of WCDMA Transceiver with VHDL-AMS Language, *IEEE Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, 2006, 113-118.

[Jiang99] T. Jiang, C. Kellon and R.D. Blanton, Inductive Fault Analysis of a Microresonator, *Proc. Modeling and Simulation Microsystems*, 1999, 498-501.

[Jiang06] T. Jiang and R.D. Blanton, Inductive Fault Analysis of Surface-Micromachined MEMS, *IEEE Transactions on Computed-aided Design of Integrated Circuits and Systems*, 25(6), 2006, 1104-1116.

[**Joannon08**] Y. Joannon, V. Berouille, C. Robach, S. Tedjini and J. Carbonero, Choice of a High-Level Fault Model for the Optimization of Validation Test Set Reused for Manufacturing Test, *Journal of VLSI Design*, 2008.

[**Johnson03**] D.R. Johnson, *Conformability analysis for the control of quality costs in electronic systems*, doctoral thesis, University of Hull, 2003.

[**Kaehler**] K.D. Kaehler, FUZZY LOGIC - AN INTRODUCTION, <http://www.ece.rochester.edu/research/wcng/meetings/FUZZY%20LOGIC%20-%20Presented.doc> [Accessed on 09/07/08].

[**Kalpana04**] P. Kalpana, K. Gunavathi, Behavioral Modeling and Fault Simulation of System on Chips, *Academic Open Internet Journal*, 13, 2004, 1-6.

[**Kamon00**] Mattan Kamon, Frank Wang and Jacob White, Generating nearly optimally compact models from Krylov-subspace based reduced-order models, *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 47(4), April 2000, 239-248.

[**Kerns95**] K.J. Kerns, I.L. Wemple and A.T. Yang, Stable and efficient reduction of substrate model networks using congruence transforms, *IEEE/ACM Pro. ICCAD*, Nov. 1995, 207-214.

[**Khouas00**] A. Khouas, A. Derieux, Fault Simulation for Analog Circuits Under Parameter Variations, *Journal of Electronic Testing: Theory and Applications*, 16, 2000, 269-278.

[**Kilic04**] Y. Kilic, M. Zwolinski, Behavioral Fault Modeling and Simulation Using VHDL-AMS to Speed-Up Analog Fault Simulation, *Journal of Analog Integrated Circuits and Signal Processing*, 39(2), 2004, 177-190.

[**Kundert90**] K.S. Kundert, J.K. White and A. Sangiovanni-Vincentelli, *Steady-state Methods for Simulating Analog and Microwave Circuits* (Kluwer Academic Publishers, 1990).

[Li03] P. Li and L.T. Pileggi, NORM: Compact Model Order Reduction of Weakly Nonlinear Systems, *Proceeding of ACM/IEEE DAC*, 2003, 472-477.

[Li05] P. Li and L.T. Pileggi, Compact Reduced-order Modeling of Weakly Nonlinear Analog and RF Circuits, *IEEE Transactions on computer-aided design of integrated circuits and systems*, 23(2), February 2005, 184-203.

[Ljung75] L. Ljung, T. Soderstrom and I. Gustavsson, Counterexamples to the General Convergence of a Commonly Used Recursive Identification Method, *IEEE Transactions on Automatic Control*, 20(5), 1975, 643-652.

[Ljung99] L. Ljung, *System Identification-Theory for the User* (Prentice-Hall, Inc., 1999).

[Maly88] W. Maly, W.R. Moore and A.J. Strojwas, Yield Loss Mechanisms and Defect Tolerance, *Research Report No. CMU-CAD-88-18*, 1998.

[MATLAB6.5] Help File, *MATLAB6.5*.

[McConaghy05] T. McConaghy, T. Eeckelaert, and G. G. E. Gielen, CAFFEINE: Template-Free Symbolic Model Generation of Analog Circuits via Canonical Form Functions and Genetic Programming, *Proceedings Design Automation and Test in Europe Conference*, 2, Mar 2005, 1082-1087.

[McConaghy05a] T. McConaghy and G. Gielen, Analysis of Simulation-Driven Numerical Performance Modeling Techniques for Application to Analog Circuit Optimization, *ISCAS*, May 2005, 1298-1301.

[Mentor]

<[http://www.mentor.com/products/ic\\_nanometer\\_design/simulation/advance\\_ms](http://www.mentor.com/products/ic_nanometer_design/simulation/advance_ms)>.

[Middleton90] R.H. Middleton, G.C. Goodwin, *Digital Control and Estimation – A Unified Approach* (Prentice-Hall, Inc., 1990).

[**Moore81**] Bruce Moor, Principal component analysis in linear systems: Controllability, observability, and model reduction, *IEEE Transactions on Automatic Control*, 26(1), February 1981, 17-32.

[**Mosis**] The MOSIS Service, [http://www.mosis.com/cgi-bin/cgiwrap/umosis/swp/params/ibm-013/t82h\\_8rf\\_8lm\\_dm-params.txt](http://www.mosis.com/cgi-bin/cgiwrap/umosis/swp/params/ibm-013/t82h_8rf_8lm_dm-params.txt) [Accessed on 05/11/08].

[**Mutnury03**] B. Mutnury, M. Swaminathan, and J. Libous, Macro-modelling of non-linear I/O drivers using spline functions and finite time difference approximation, *Proc. Electrical Performance of Electronic Packaging*, 2003, 273-276.

[**Nagi92**] N. Nagi, Jacob A. Abraham, Hierarchical Fault Modeling for Analog and Mixed-Signal Circuits, *IEEE VLSI test symposium*, 1992, 96-101.

[**Nagi93**] N. Nagi, A. Chatterjee, Jacob A. Abraham, DRAFTS: Discretized Analog Circuit Fault Simulator, *30<sup>th</sup> ACM/IEEE Design Automation Conference*, 1993, 509-514.

[**Nayfeh95**] A. Nayfeh and B. Balachandran, *Applied Nonlinear Dynamics: Analytical, Computational, and Experimental Methods* (Wiley, 1995).

[**Nikitin07**] Pavel V. Nikitin, C.-J. Richard Shi, VHDL-AMS based modeling and simulation of mixed-technology Microsystems: a tutorial, *Integration, the VLSI journal*, 40, 2007, 261-273.

[**Odabasioglu97**] A. Odabasioglu, M. Celik and L. Pileggi, Prima: Passive reduced-order interconnect macromodeling algorithm, *International Conference on Computer Aided-Design*, San Jose, California, November 1997, 58-65.

[**Ohletz91**] M.J. Ohletz, Hybrid Built-In Self-Test (HBIST) for Mixed Analogue/Digital Integrated Circuits, *Proceedings of European Test Conference*, 1991, 307-316.

[**Olbrich96**] T. Olbrich, J. Pèrez, I.A. Grout, A.M.D. Richardson, C. Ferrer, Defect-Oriented VS Schematic-Level Based Fault Simulation for Mixed-Signal ICs, *Proceedings of the International Test Conference (ITC)*, 1996, 511-520.

[**Olbrich97**] T. Olbrich, I.A. Grout, Y. Eben Aimine, A.M. Richardson and J. Contensou, A New Quality Estimation Methodology for Mixed-Signal and Analogue ICs, *European Design and Test Conference (ED&TC)*, Paris, March 1997, 573-580.

[**Pan96**] C.Y. Pan and K.T. Cheng, Fault Macromodeling and A Testing Strategy for Opamps, *Journal of Electronic Testing: Theory and Applications*, 9, 1996, 225-235.

[**Pan97**] C.Y. Pan and K.T. Cheng, Fault Macromodeling for Analog/Mixed-Signal Circuits, *International Test Conference*, 1997, 913-922.

[**Pecheux05**] F. Pecheux, C. Lallement, A. Vachoux, VHDL-AMS and Verilog-AMS as alternative hardware description languages for efficient modeling of multidiscipline systems, *IEEE Trans. Comput. Aided Des. Integrated Circuits Syst*, 24(2), 2005, 204-205.

[**Pella97**] F. Pellandini, V. Moser and H.P. Amann, *Behavioral Modelling of Analogue Systems with absynth* (Kluwer Academic Press, 1997).

[**Phillips98**] J. Phillips, Model reduction of time-varying linear systems using approximate multipoint Krylov-subspace projectors, *International Conference on Computer Aided-Design*, Santa Clara, California, November 1998, 96-102.

[**Phillips00**] J. Phillips, Automated extraction of nonlinear circuit macromodels, *Proceeding of IEEE CICC*, 2000, 451-454.

[**Phillips02**] J. Phillips, L. Daniel and L.M. Silveira, Guaranteed passive balancing transformations for model order reduction. *Proc. IEEE DAC*, June 2002, 52-57.

[**Pillage90**] L.T. Pillage and R.A. Rohrer, Asymptotic waveform evaluation for timing analysis, *IEEE Trans. CAD*, 9, April 1990, 352-366.

**[Regression]** A place in history: a guide to using GIS in historical research, <http://ahds.ac.uk/history/creating/guides/gis/sect101.html> [Accessed on 05/04/08].

**[Renovell96]** M. Renovell, F. Azaïs and Y. Bertrand, Analog Signature Analyzer for Analog Circuits: BIST Implementations, *In 2nd Intl. Mixed Signal Testing Workshop*, Quebec, Canada, 1996, 233-238.

**[Rewiński01]** M. Rewiński and J. White, A Trajectory Piecewise-Linear Approach to Model-Order Reduction and Fast Simulation of Nonlinear Circuits and Micromachined Devices, *Proceedings IEEE/ACM International Conference on Computer Aided Design*, 2001, 252–257.

**[Rich92]** A.M.D. Richardson, A.P. Dorey, Reliability Indicators, *European Symposium on reliability of electron devices, failure physics and analysis (ESREF)*, Schwabisch Gmund, Germany, 5-8<sup>th</sup> October 1992, 277-283.

**[Rosen98]** R. Rosenberger, S.A. Huss, A Systems Theoretic Approach to Behavioural Modeling and Simulation of Analog Functional Blocks, *Proceedings of DATE*, 1998, 721-728.

**[Roychowdhury99]** J. Roychowdhury, Reduced-Order Modelling of Time-Varying Systems, *IEEE Transactions on circuits and systems-II: Analog and Digital Signal Processing*, 46(10), October 1999, 1273-1288.

**[Roychowdhury01]** J. Roychowdhury, Analysing Circuits with Widely-separated Time Scales using Numerical PDE Methods, *IEEE Trans. on Circuits and Systems-I*, 48, May 2001, 578-594.

**[Roychowdhury03]** J. Roychowdhury, Automated Macromodel Generation for Electronic Systems, *IEEE Behavioral Modeling and Simulation Workshop*, San Jose, CA, Oct. 2003, 11-16.



[**Roychowdhury04**] J. Roychowdhury, Algorithmic Macromodelling Methods for Mixed-Signal Systems, *Proceedings of the 17<sup>th</sup> International Conference on VLSI Design (VLSID)*, 2004, 141-147.

[**Russell93**] D.S. Learmonth, G. Russell, BIST Schemes for Testing Analogue Circuits, *IEE Colloquium on Testing-the Gordian Knot of VLSI Design*, 1993, 7/1-7/4.

[**Saber03**] SaberBook reference, 2003.

[**Saber04**] Saber Library and Model User Guide, *Synopsys, Inc.*, 2004.

[**Sachdev95**] M. Sachdev, A Realistic Defect Oriented Testability Methodology for Analog Circuits, *Journal of Electronic testing: Theory and Applications*, 6, 1995, 265-276.

[**Schetzen80**] M. Schetzen, *The Volterra and Wiener Theories of Nonlinear Systems* (New York: John Wiley, 1980).

[**Sebeke95**] C. Sebeke, J.P. Teixeira, and M.J. Ohletz, Automatic Fault Extracting and Simulation of Layout Realistic Faults for Integrated Analogue Circuits, *Proc. European Design and Test Conference*, 1995, 464-468.

[**Silveira96**] L. Miguel Silveira, M. Kamon, I. Elfadel, and J. White, A Coordinate-Transformed Arnoldi Algorithm for Generating Guaranteed Stable Reduced-Order Models of RLC Circuits, *Proc. ICCAD*, November 1996, 288-294.

[**Simeu05**] E. Simeu, S. Mir, Parameter Identification Based Diagnosis in Linear and Non-linear Mixed-Signal Systems, *11<sup>th</sup> International Mixed-Signals Testing Workshop*, June 27-29, 2005.

[**SMASHR05**] SMASH™ Reference Manual, 5.6, December 2005.

[**SMASHU05**] SMASH™ User Manual, 5.6, December 2005.

[**Smith96**] D. Smith, VHDL and Verilog Compared and Contrasted-Plus Modeled Example Written in VHDL, Verilog and C, *Proceedings of the 33rd Design Automation Conference*, June 1996, 771-776.

[**Spec97**] SpectreHDL Reference Manual, Product Version 4.4.1, *Cadence Design Systems, Inc.*, San Jose (USA), 1997.

[**Spiegel95**] Jan Van der Spiegel, SPICE Models for Selected Devices and Components, *University of Pennsylvania*, 1995, <http://www.seas.upenn.edu/~jan/spice/spice.models.html#mosis1.2um> [Accessed on 01/10/08].

[**Spinks97**] S.J. Spinks, C.D. Chalk, I.M. Bell, M. Zwolinski, Generation and Verification of Tests for Analogue Circuits Subject to Process Parameter Deviations, *International Symposium on Defect and Fault Tolerance in VLSI Systems*, Paris, France, October 1997, 100-108.

[**Spinks98**] S.J. Spinks, *Fault Simulation for Structural Testing of Analogue Integrated Circuits*, doctoral thesis, University of Hull, 1998.

[**Spinks04**] S. Spinks, I. Bell, ANTICS 1.3 User Guide, *VLSI Design and Test Group, Department of Electronic Engineering, University of Hull*, 2004.

[**Stopja04**] V. STOPJAKOVÁ, P. MALOŠEK, D. MIČUŠIČ, M. MATEJ, M. MARGALA, Classification of Defective Analog Integrated Circuits Using Artificial Neural Networks, *Journal of Electronic Testing: Theory and Applications*, 20, 2004, 25-37.

[**Synopsys**] <<http://www.synopsys.com>>.

[**SystemVision**] Getting Started with SystemVision, [http://www.mentor.com/products/sm/systemvision/demos/getting\\_started.cfm/](http://www.mentor.com/products/sm/systemvision/demos/getting_started.cfm/) [Accessed on 15/08/06].

[**Tan03**] S.X.-D Tan and C.J.-R Shi, Efficient DDD-based term generation algorithm for analog circuit behavioral modeling, *Proc. IEEE ASP-DAC*, January 2003, 789-794.

[**Tietze93**] U. Tietze, C. Schenk and E. Gamm, *Halbleiter-Schaltungstechnik* (Springer-Verlag, Berlin, 1993).

[**Uppal05**] F.J. Uppal and R.J. Patton, Neuro-fuzzy uncertainty de-coupling: a multiple-model paradigm for fault detection and isolation, *Int. J. Adapt. Control Signal Process*, 2005, 281-304.

[**Vasiyev03**] D. Vasiyev, M. Rewiński, J. White, A TBR-based Trajectory Piecewise-Linear Algorithm for Generating Accurate Low-order Models for Nonlinear Analog Circuits and MEMS, *DAC*, June 2-6, 2003, 490-495.

[**Volterra**] Volterra Series and Volterra Kernel, [http://ctas.east.asu.edu/chnam/ASE\\_Book/Volterra%20Theory.htm](http://ctas.east.asu.edu/chnam/ASE_Book/Volterra%20Theory.htm) [Accessed on 06/08/05].

[**Voo97**] R. Voorakaranam, S. Chakrabarti, J. Hou, A. Gomes, S.Cherubal, A.Chatterjee, Hierarchical Specification-Driven Analog Fault Modeling for Efficient Fault Simulation and Diagnosis, *International Test Conference*, 1997, 903-912.

[**Watkins**] S. Watkins and K. Wong, Mixed-Signal Modeling with Vanilla VHDL and Verilog, *Blue Pacific Computing, Inc.*, San Diego, California, <http://www.bluepc.com/mixpap.html>, [Accessed on 08/05/06].

[**Wei05**] Y. Wei and A. Daboli, Systematic Development of Analog Circuit Structural Macromodels through Behavioral Model Decoupling, *DAC*, June, 2005, 57-62.

[**Wilkins86**] B.R. Wilkins, *Testing Digital Circuits: An Introduction* (Chapman and Hall, 1986).

[**Wilkinson91**] A.J. Wilkinson, S. Roberts, P.M. Taylor, G.E. Taylor, "Real time plant monitoring using recursive identification", *Proceedings of COMADEM*, 1991, 310-315.

[**Wilson01**] P.R. Wilson, Y. Kilic, J.Neil Ross, M. Zwolinski, A.D.Brown, Behavioural Modelling of Operational Amplifier Faults using Analogue Hardware Description Languages, *BMAS*, 2001, 106-112.

[**Wilson02**] P.R. Wilson, Y. Kilic, J. Neil Ross, M. Zwolinski, A.D.Brown, Behavioural Modelling of Operational Amplifier Faults using VHDL-AMS, *Proceedings of Design, Automation and Test in Europe (DATE)*, 2002, 1133-.

[**Xia08a**] L. Xia, I.M. Bell and A.J. Wilkinson, Automated Macromodel Generation for High Level Modelling, *Design and Technology Integrated Systems (DTIS)*, Tozeur, Tunisia, March 25-28, 2008.

[**Xia08b**] L. Xia, I.M. Bell and A.J. Wilkinson, A Novel Approach for Automated Model Generation, *International Symposium on Circuits and Systems (ISCAS)*, Seattle, WA, 2008, pp. 504-507.

[**Xia08c**] L. Xia, I.M. Bell and A.J. Wilkinson, An Automated Model Generation Approach for High Level Modeling, *World Congress on Engineering (WCE)*, London, UK, July 2-4, 2008.

[**Xing98**] Y. Xing, Defect-Oriented Testing of Mixed-Signal ICs: Some Industrial Experience, *Proc. IEEE International Test Conference (ITC)*, 1998, 678-687.

[**Yang98**] Z.R. Yang, M. Zwolinski, A Methodology for Statistical Behavioral Fault Modeling, *BMAS*, 1998.

[**Yang04**] B. Yang and B. MacGaughy, An Essentially Non-Oscillatory (ENO) High-Order Accurate Adaptive Table Model for Device Modeling, *Proc. IEEE DAC*, 2004, 864-867.

[**Yeredor00**] A. Yeredor, The Extended Least Squares Criterion: Minimization Algorithms and Applications, *IEEE Transactions on signal processing*, 49(1), January 2000, 74-86.

[Zadeh63] L.A. Zadeh and C.A. Desoer, *Linear System Theory: The State-space Approach* (McGraw-Hill, New York, 1963).

[Zhang00] Q.J. Zhang and K.C. Gupta, *Neural Networks for RF and Microwave Design* (Boston: Artech House, 2000).

[Zorzi02] M. Zorzi, N. Speciale, G. Masetti, A New VHDL-AMS Simulation Framework in Matlab, *BMAS*, Santa Rosa, CA, 6-8 October 2002, 185-190.

[Zorzi03] M. Zorzi, F. Franze, N. Speciale, Construction of VHDL-AMS simulator in Matlab, *BMAS*, 2003, 113-117.

[Zwo96] M. Zwolinski, C. Chalk and B.R. Wilkins, Analogue Fault Modelling and Simulation for Supply Current Monitoring, *Proceedings of European Design and Test Conference*, 1996, 547-552.

[Zwo97] M. Zwolinski, C. Chalk and A.J. Perkins, Multi-Level Fault Modeling of Analog Circuits, *IEEE/VIUF International Workshop on Behavioral Modeling and Simulation (BMAS)*, 1997, 107-113.

[Zwo00] M. Zwolinski, *Digital System Design with VHDL* (Pearson Education Ltd., 2000).

# *List of Publications*

[1] Likun Xia, I.M. Bell, A.J. Wilkinson, Automated Macromodel Generation for High Level Modelling, *Design and Technology Integrated Systems (DTIS)*, Tozeur, Tunisia, March 25-28, 2008.

[2] Likun Xia, I.M. Bell, A.J. Wilkinson, A Novel Approach for Automated Model Generation, *International Symposium on Circuits and Systems (ISCAS)*, Seattle, USA, May 18-21, 2008, pp. 504-507.

[3] Likun Xia, I.M. Bell, A.J. Wilkinson, An Automated Model Generation Approach for High Level Modelling, *World Congress on Engineering (WCE)*, London, UK, July 2-4, 2008.

## *Submitted Papers*

[1] Likun Xia, Ian M. Bell, Antony J. Wilkinson, High Level Fault Modelling using an Automated Model Generation Algorithm, *Journal of IEEE Trans. CAD*. (Submitted in July, 2008)

[2] Likun Xia, Ian M. Bell, Antony J. Wilkinson, Analogue Simulation using Automated Model Generation based on SPICE-level Description, *International Journal of Modelling and Simulation, ACTA*. (Submitted in June, 2008)

[3] Likun Xia, Ian M. Bell, Antony J. Wilkinson, A Robust Approach for Automated Model Generation, *Design, Automation and Test in Europe (DATE)*, Nice, France, April 20-24, 2009. (Submitted)