Behavioural Simulation of Mixed Analogue/Digital Circuits

David Ian Long

A thesis submitted in partial fulfilment of the requirements of Bournemouth University for the degree of Doctor of Philosophy

April 1996

Bournemouth University

ABSTRACT

Continuing improvements in integrated circuit technology have made possible the implementation of complex electronic systems on a single chip. This often requires both analogue and digital signal processing. It is essential to simulate such IC's during the design process to detect errors at an early stage. Unfortunately, the simulators that are currently available are not well-suited to large mixed-signal circuits.

This thesis describes the design and development of a new methodology for simulating analogue and digital components in a single, integrated environment. The methodology represents components as behavioural models that are more efficient than the circuit models used in conventional simulators. The signals that flow between models are all represented as piecewise-linear (PWL) waveforms. Since models representing digital and analogue components use the same format to represent their signals, they can be directly connected together.

An object-oriented approach was used to create a class hierarchy to implement the component models. This supports rapid development of new models since all models are derived from a common base class and inherit the methods and attributes defined in their parent classes. The signal objects are implemented with a similar class hierarchy.

The development and validation of models representing various digital, analogue and mixed-signal components are described. Comparisons are made between the accuracy and performance of the proposed methodology and several commercial simulators.

The development of a Windows-based demonstration simulation tool called POISE is also described. This permitted models to be tested independently and multiple models to be connected together to form structural models of complex circuits.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Acknowledgement

This research was sponsored by the Department of Electronics at Bournemouth University.

The author would like to acknowledge the help and support given by his supervisors: Professor Sa'ad Medhat, Professor John Lidgey and Dr. Randeep Soin.

The author would also like to acknowledge the support of the fellow researchers and members of staff of the Department of Electronics at Bournemouth University.

Author's Declaration

The critical review of the literature presented in Chapter 2 is based on published material. The views expressed are the author's own except where indicated by references.

The investigation into PWL waveform representation and the development of modelling techniques presented in Chapter 3 are based on the authors own work.

The application of object-oriented techniques to mixed-signal simulation, the model development and the experiments presented in Chapter 4 are the authors own work. The validation of the PWL models made use of component library parts where suitable models were supplied with the commercial simulators.

The Windows simulation demonstration system presented in Appendix A made use of several utility classes supplied with the compiler used. These are indicated in the text. The development of the demonstration system is the author's own work

Glossary of Terms

ASIC	Stands for Application Specific Integrated Circuit. This is usually taken to refer to an integrated circuit (IC) that is designed to implement the functionality required for a particular product as opposed to standard ICs that implement simpler functions but can be used in a wide range of products. An ASIC solution reduces the number of ICs required for a product and so can reduce costs. ASICs are one of the major growth areas in electronics. ASICs are sometimes referred to as "Custom ICs".
CASE	Computer Aided Software Engineering. Computer-based tools to assist in the application of formal approaches to the design and development of software.
IC	Integrated Circuit. An electric circuit manufactured on a single semi-conductor "substrate" - usually silicon.
PC.	Personal Computer. (Usually taken to mean one compatible with an IBM Personal Computer).
Mixed-Level Simulator.	A simulator that can evaluate systems consisting of components at more than one abstraction level (e.g. a mixture of behavioural and circuit-level models).
Mixed-Signal System	A system consisting of both analogue and digital signals.
Mixed-Signal Simulator.	A simulator capable of evaluating a mixed-signal system.
Piece-Wise Constant (PWC)	A method of representing a discrete (discontinuous) signal as a set of points joined by constant magnitude straight line segments.
Piece-Wise Linear (PWL)	A method of representing a continuous signal as a set of points joined by straight line segments.
UNIX.	A multi-tasking operating system traditionally used on mainframe and mini-computers but now also used for workstations.

VHDL — A language for describing digital systems that has been defined as a standard (1076) by the Institute of Electrical and Electronics Engineers (IEEE). This language can be simulated and can be translated into a physical circuit layout by synthesis tools. VHDL was originally developed as part of the US Department of Defense's Very High Speed Integrated Circuit (VHSIC) programme. The letters stand for VHSIC Hardware Description Language.

VHDL-A — A superset of the VHDL syntax to cover the description of analogue and mixed-signal systems. An IEEE sponsored committee has been working on the standardisation of VHDL-A since 1992 and has almost completed its task. VHDL-A is likely to be issued as IEEE standard 1076.1 in 1996.

VLSI — Very Large-Scale Integration. The technology that enables integrated circuits (ICs) containing hundreds of thousands of transistors to be fabricated.

Workstation. — A powerful, multi-tasking, networked computer (e.g. Sun, HP-Apollo). Typically uses a variant of the UNIX operating system and a graphical user interface (GUI) such as X-Windows or Motif.

1. Overview and Requirements Specification.

1.1 Introduction.

This thesis describes a project entitled "Behavioural Simulation of Mixed Analogue-Digital Circuits". This chapter describes the background to the project and defines the project objectives. It also contains a taxonomy of the other chapters.

1.2 Rationale.

As IC technology has improved, allowing higher integration and performance, it has become possible to implement complete electronic systems on a single chip. In the majority of cases, the system functions are mostly implemented with digital circuits. However, many systems also require some analogue signal processing capability. This can range from simple analogue to digital converters for interfacing external transducers to complex filtering and wave-shaping circuits. There are a number of advantages to be gained from integrating the analogue functions into the same chip as the rest of the system. These include a smaller product, lower power consumption, quicker assembly, lower component counts and increased reliability.

The design of mixed analogue and digital custom integrated circuits (mixed-signal ASICs) is one of the main growth areas in the field of electronics. Despite the availability of new products and technologies, the number of new devices that have been designed is lower than expected. This can be attributed to the lack of good computer-aided design tools for mixed-signal systems. This project investigates the computer simulation tools that are available for mixed-signal circuits and aims to develop a better simulation methodology.

Computer simulation is vital to the design of any integrated circuit since it is impossible to correct design errors once a chip has been fabricated. It is therefore vital to establish that a design is good before manufacture. The majority of simulation tools that are

currently commercially available are not ideally suited to the design of mixed-signal circuits: a simulator that can evaluate both analogue and digital functions is required. Traditional simulators cannot combine the efficiency required to simulate a complex VLSI device (with tens or hundreds of thousands of digital gates) with the accuracy required to simulate low-level analogue functions. A new class of simulator has therefore been developed to address this problem. These simulators are known as "Mixed-Signal" simulators since they attempt to combine analogue and digital simulation into a single process. A number of experimental and commercial mixed-signal simulators have been announced since the early 1980's when the need first became apparent. They are generally a combination of two or more of the methods found in existing simulators.

An ideal mixed-signal simulator should be capable of simultaneously processing the analogue and digital components of a large, complex system. It should also facilitate modern design methodologies such as hierarchical (e.g. "top-down") design and use of *'hardware description languages'* (HDLs). These require a mixed-signal simulator that also supports "mixed-mode" modelling: i.e. it must be able to evaluate component models existing at different levels of abstraction. The levels of abstraction typically required are compared in Table 1-1. A mixed-mode approach enables trade-offs to be made between the time taken for the simulation to run and the accuracy of the results: each simulation level in Table 1-1 is approximately 10 times less efficient than the level above it, except for electrical simulation, that is about 100 times less efficient than timing simulation.

The approach taken in most mixed signal design environments is to use separate analogue and digital simulators coupled together. This requires that a mixed signal design is partitioned into analogue and digital sections before a simulation is run. The methods used to describe the component connectivity ('*netlists*') within the analogue and digital regions is often different and usually incompatible. The analogue connectivity is typically expressed as a SPICE [1] netlist whilst the digital connectivity could be expressed in a proprietary digital simulator netlist format or with a hardware description language such as VHDL [2] or Verilog. A problem with this approach is that the boundaries between analogue and digital partitions are likely to change during the design process. The need for multiple types of netlist at different levels of abstraction complicates the design

Simulation Level	Model Representation	Type of Analysis
Behavioural	Algorithms	Functional verification
RTL	RTL (Register Transfer Level) primitives e.g. registers, counters	Functional verification
Gate	Boolean Algebra, State Tables	Functional verification and 1st order timing
Switch	Signal/node strengths and switch position tables	Functional verification and 1st order timing
Timing	Resistance-Voltage Graphs	Detailed PWL waveform timing
Electrical	Non-linear Algebraic Equations and ordinary differential equations (ODE's)	Detailed analogue waveforms, electrical loading, circuit stability, etc.

Table 1-1. Levels of circuit simulation.

process and provides a potential source of errors for the design integrity. Work is continuing by an IEEE committee (1076.1) to develop analogue extensions to VHDL to address these issues (see Appendix B). The new language (VHDL-A) will provide a mechanism to describe analogue, digital and mixed signal components in a compatible format. It will support hierarchical design and facilitate multiple views of individual components (as found in VHDL).

The demand for mixed signal simulation is increasing as the number of mixed signal IC's designed each year grows. This trend looks likely to continue. However, the best way to implement a multi-level mixed signal simulator is not clear. Research and development of mixed signal simulators and simulation methodologies are therefore continuing in both academic and commercial sectors. Issues that must be resolved include:

- How the behaviour of analogue and digital components is described and evaluated.
- How the connectivity of components is described to the simulator.
- The interfaces required between different types of component.
- Representation of signals.

- Representation of time in analogue, digital and mixed-signal simulation algorithms.
- How the different simulation algorithms are initialised and kept in synchronisation.

Object-oriented methods have been used in many fields of simulation since the physical model of a system can often be best represented in software as a set of objects [3,4,5,6]. This suggests that an object-oriented approach could be well-suited to the development of a mixed signal simulator. There are two potential advantages of using an object-oriented approach for simulating mixed signal circuits. The first is the ability to define a set of reusable objects to describe generic models. This is significant when simulating circuits that consist of standard components or standard types of component. Each electrical component could be associated with an object. The object would be an "instance" of a part from a library of "standard" objects, with additional parameters to reflect the properties of that particular component. This design philosophy is consistent with the automated generation of a simulation model from a circuit description. The second advantage is due to a characteristic of object-oriented programming languages known as "overloading". This is a mechanism for identifying the operation a particular function is going to perform, according to the type of the parameters that invoke the function. It could be used to automate the selection of the most appropriate model or methodology in a simulator capable of working concurrently at multiple levels of abstraction.

This project is concerned with behavioural modelling. It uses a more abstract and therefore efficient type of model than the circuit level models used in most commercial simulators. Consequently, it should be better suited to simulating large mixed-signal systems. Behavioural models are inevitably less accurate than circuit level models or else are only valid over a limited range of operation. To achieve acceptable levels of both accuracy and efficiency with behavioural models, new techniques of representing signals and solving circuit equations are proposed. Simulation models based upon these new techniques are developed using an object-oriented approach. These are used to construct a demonstration mixed signal simulation environment to validate the proposed methodology.

1.3 Aims and Objectives.

To design a new methodology for simulating circuits containing both analogue and digital components that is consistent with the proposed analogue extensions to the IEEE standard hardware description language (VHDL-A).

To develop behavioural models of standard components typically used in mixed-signal ASICs in order to investigate the accuracy and performance of the proposed simulation methodology.

1.4 Taxonomy of Chapters.

1.4.1 Chapter 1.

This chapter introduces the work that has been carried out towards this Project. It describes the aims and objectives of the research and gives a brief rationale for the approach taken.

1.4.2 Chapter 2.

A review of the techniques and approaches that have been used for simulating integrated circuits is presented in this chapter. This review is based on material published in a wide range of technical books, journals and conference proceedings.

1.4.3 Chapter 3.

This chapter describes the modelling techniques developed during this research project. These techniques are based around a piece-wise linear (PWL) representation of all signals (both analogue and digital).

1.4.4 Chapter 4.

This chapter describes how an object-oriented approach was applied to this research project and the validation of the simulation techniques and component models.

1.4.5 Chapter 5.

This chapter presents overall conclusions of the outcome of this research project. It also makes recommendations for areas requiring further work.

1.4.6 Appendix A.

This describes an experimental simulation system that has been developed to demonstrate how a simulator based on the methods proposed in this thesis could be implemented.

1.4.7 Appendix B.

This provides an overview of proposed extensions to VHDL to support mixed-signal circuits.

2. Review of Mixed-Signal Simulation.

2.1 Introduction.

A review of the techniques and approaches that have been used for simulating integrated circuits is presented in this chapter. This review is based on material published in a wide range of technical books, journals and conference proceedings. It commences with a discussion of the development of simulators for integrated circuits and their shortcomings. The development of the most important mixed-signal simulators and mixed-signal simulation methodologies within the commercial and academic sectors is then discussed. The simulation approaches taken reflect the different objectives held by these two sectors. Consequently, the commercial and academic developments in the area of mixed-signal simulation are considered in separate sections.

2.2 Background to Computer Simulation of ICs.

Computer simulation has been extensively used since the early 1970's to verify the behaviour of integrated circuits (IC's) prior to manufacture. It was initially feasible to simulate the behaviour of a complete IC by modelling the currents and voltages around every transistor in the circuit. This is known as analogue or **circuit-level** simulation. The early analogue simulators could only model circuits with a few hundred transistors, even on the most powerful computers available. Advances in mathematical methods led to more advanced simulators that could process larger circuits. The best known of these were the SPICE1 [1] and SPICE2 [7] simulators developed at the University of Berkeley. The original SPICE programs were designed to simulate circuits containing a hundred or so transistors, although they have since been adapted to work with much larger circuits. SPICE1 and SPICE2 were written in the Fortran programming language: later versions are almost 18,000 lines long [8]. SPICE3 [9] was written in the C programming language to increase efficiency and was released in 1986. All of the SPICE simulators were placed in the public domain. SPICE2 and SPICE3 have since become the basis for most commercial circuit simulators currently in use

SPICE uses a modified "nodal analysis" approach to obtain the value of unknown vectors from the set of excitation vectors and circuit coefficients. The circuit coefficients are arranged in a (sparse) matrix that describes the branch admittance between every node in the system. SPICE provides several types of analysis including non-linear dc analysis, non-linear transient (time domain) analysis and linear small signal (frequency) analysis. Transient analysis is the most important verification method for the majority of circuits: it can be compared to using a signal generator to excite a physical circuit and observing the results on an oscilloscope. Unfortunately, transient analysis is also the most time consuming. It uses numerical integration methods to convert the (non-linear) differential equations describing the system into a set of linear algebraic equations that it solves using Gaussian elimination. These equations are only valid at the instance in simulation time about which the integrations were performed (known as the current "time-step"). When the simulator advances to the next time-step, the integrations must be repeated to obtain a new set of linear equations. If the signals are changing rapidly, very small time-steps must be used to ensure the integrations converge to the correct solution. Transient analysis can therefore require a large number of mathematical operations to be performed.

The simulation of large circuits using SPICE is very computationally demanding and so is time consuming and expensive. There are two reasons why the SPICE approach to transient analysis becomes inefficient for large circuits. The first is that the time taken to solve the matrix equations grows (approximately exponentially) as the size of the matrix is increased until it dominates the simulation time [10]. The second is due to the direct method used to solve for the unknowns: i.e. they are all found at the same time. This forces every differential equation in the system to be linearised using a common time-step. The integration time-step has to be small enough to represent the fastest changing signal for all nodes. This becomes inefficient when there are rapidly and slowly changing signals in a single circuit (as is often the case in a large system).

As the levels of IC integration increased, a point was reached when it was no longer viable to simulate the behaviour of a complete IC chip using a circuit-level simulator. Alternative techniques therefore had to be found. Since most ICs only contained digital functions, the approach generally taken was to model the chip as a collection of

interconnected logic gates. This is known as **gate level** simulation. Each logic gate input is assumed to only recognise two states: logic '1' and logic '0'. An additional state is often used in gate-level simulators to model the effects of open circuit inputs and conflicting short circuit outputs: the indeterminate state 'X'. A fourth state 'Z' is sometimes used to represent a high impedance tri-state output. This simplification of the models enabled the simulation of ICs with thousands of transistors to be performed, at the expense of small-signal accuracy. The outputs generated by each model are derived via Boolean operations from input states. These operations are simpler than the arithmetic operations required to evaluate the voltages and currents associated with transistors. Once the output of a logic gate has been set to a particular state it is assumed to remain in that state until new input states are received. A gate-level simulator therefore only needs to evaluate a gate model at the instant when its inputs change state. This means that each model is only required to be invoked at discrete time-steps within the simulation. This is very different from circuit level simulation where every model needs to be evaluated at every time-step in the simulation. The approach used in digital simulators is known as 'event-driven' while that used in analogue simulators is known as 'continuous'. The event-driven approach together with the simplified models enables gate-level simulators to run hundreds of times faster than the most efficient analogue simulator. Including four logic states instead of two increases the ability of the simulator to detect error conditions at the expense of efficiency. Some gate-level simulators can associate a drive strength with each logic gate output. A range of drive strengths enables the state of short circuit outputs to be resolved to a recognised value. The resolution functions increase the number of circuit nodes whose state can be determined but reduce the simulator efficiency still further. The combination of multiple logic states with several drive strengths has led to the development of digital simulators that can resolve this “multi-valued” logic. A typical present generation digital simulator might work with 28-state logic: i.e. 4 logic states each with 7 possible drive strengths to represent different types of technologies and connections. Gate-level simulators provide limited timing information by modelling the propagation delays associated with each logic gate. This is used to detect hazards, glitches and race conditions.

The majority of integrated circuits produced since the early 1970's have used MOS technology. It is possible to model the MOS transistors in a logic gate as voltage-controlled switches. The resulting logic gate model can be almost as accurate as the transistor-based model used in circuit-level simulators. If a drive strength is associated with each switch in the "ON" state the voltage waveforms can be determined by considering the parasitic capacitance associated with each node. The state of a switch is determined by the node voltage on its control port (MOS gate terminal). The efficiency of this approach is lower than gate-level simulators but still much higher than circuit-level simulators. It is known as **switch-level** simulation and has become the preferred form of simulation for MOS digital IC design. Switch-level simulators are not suitable for bipolar technologies since the behaviour of bipolar transistors cannot be accurately modelled by voltage-controlled switches. They are also unsuitable for simulating analogue functions since the switch models only possess two different states (ON and OFF) - simulation of analogue MOS circuits requires the operating point of each transistor to be determined since the transistors in an analogue circuit are generally acting as transconductance amplifiers.

Simulators have also been developed that work at a higher level of abstraction. These are known as **behavioural-level** simulators because the system is modelled as a collection of functional blocks. Behavioural-level simulators support simulation of both combinational and sequential digital logic. Analogue functions are also supported in some behavioural simulators. A hardware description language (HDL) is often used to describe the operation of the system to the simulator. The level of abstraction used for this description can vary from "black boxes" containing a list of equations to structural representations that describe the connectivity and timing relationships between collections of functional blocks.

2.3 The Need for Mixed Signal Simulation.

The design of an IC that contains both analogue and digital functions (a mixed-signal IC) requires a simulator that can evaluate both analogue and digital functions. None of the simulator types described in the previous section combine the efficiency required to simulate a complex VLSI device (containing tens or hundreds of thousands of digital

gates) with the accuracy required to simulate low-level analogue functions. A new class of simulator has therefore been developed to address this problem. These simulators are known as "Mixed-Signal" simulators since they attempt to combine analogue and digital simulation into a single process. A number of experimental and commercial mixed-signal simulators have been announced since the early 1980's when the need first became apparent. They are generally a combination of two or more of the methods found in existing simulators. This is an attempt to arrive at the best trade-off between accuracy and efficiency for both analogue and digital simulation. There is no universally accepted solution to this trade off. Commercially available mixed signal simulators have tended to be based on a combination of existing digital and existing analogue simulators. Researchers have explored new algorithms and techniques that can work with both analogue and digital functions to produce experimental simulators. The demand for mixed signal simulation is increasing as the number of mixed signal IC's designed each year grows. This trend looks likely to continue. Research and development of mixed signal simulators and simulation methodologies are therefore continuing in both academic and commercial sectors. A brief review of the major contributions to these areas is given in the following sections.

2.4 Commercial Mixed Signal Simulators and Simulation Methodologies.

Mixed signal simulators can be grouped according to how the analogue and digital simulation methodologies are combined. This gives the following four categories: manual; coupled; extended; and integrated. Alternatively, they can be classified by the architecture of the combined analogue and digital simulators. There are five different architectures in common use. These are described as: sequential; paired; stand alone; nested; and framework-based. There is some overlap between these categories. A sequential architecture must be used for a manual simulation approach but can also be used with a coupled approach. Coupled simulators can also have paired, nested or framework-based architectures. Extended and integrated simulators have a stand alone architecture. These terms are all described below.

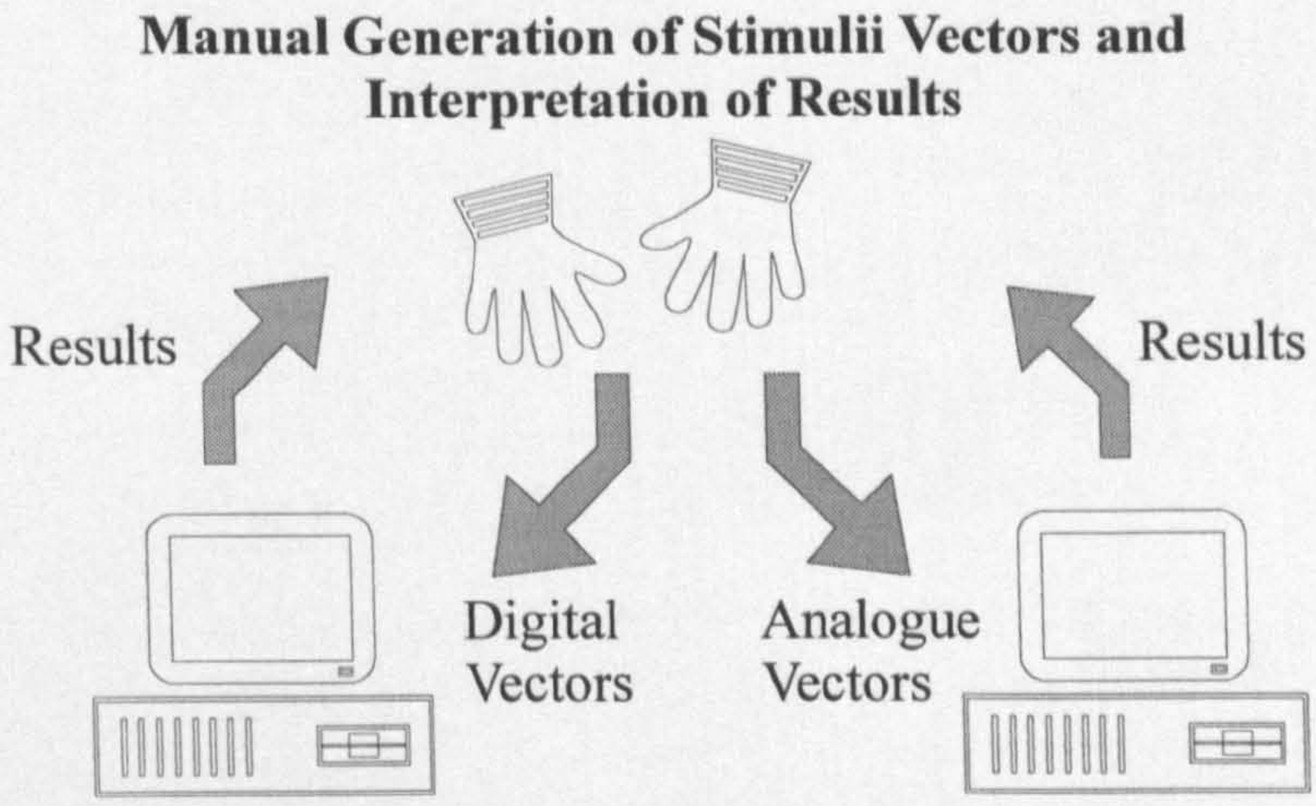


Figure 2-1. Manual Mixed-Signal Simulator Approach

A **manual approach** to mixed signal simulation uses separate analogue and digital simulators. The logic simulator generates output waveforms from the digital part of the circuit. These waveforms are then used to produce input signals for the circuit-level simulator that simulates the analogue part of the circuit. The outputs from the analogue circuit simulation are used to write input vectors for any following digital circuits. The logic simulator is then run again with these new inputs. This is illustrated in Figure 2-1. The manual approach is tedious and unreliable since the signal conversion and simulator control are both done manually. This becomes even worse if there are feedback paths between the analogue and digital parts of the IC. This methodology is not used very much except with a few entry-level ASIC design tools. An example is the Bx design system from MCE. This uses a SPICE derivative simulator (HSPICE) and its native logic simulator to process the analogue and digital parts of the MCE mixed signal Gate Arrays respectively.

The **coupled approach** implements a mixed signal simulator by connecting a circuit-level and a logic simulator together. Coupled simulators can be grouped into several categories depending on the strength and nature of the coupling.

The weakest form of coupling is found in **sequential simulators**. These are similar to the manual mixed signal simulators described above except the transfer of data between the analogue and digital simulators is automated. Each simulator only considers a forward

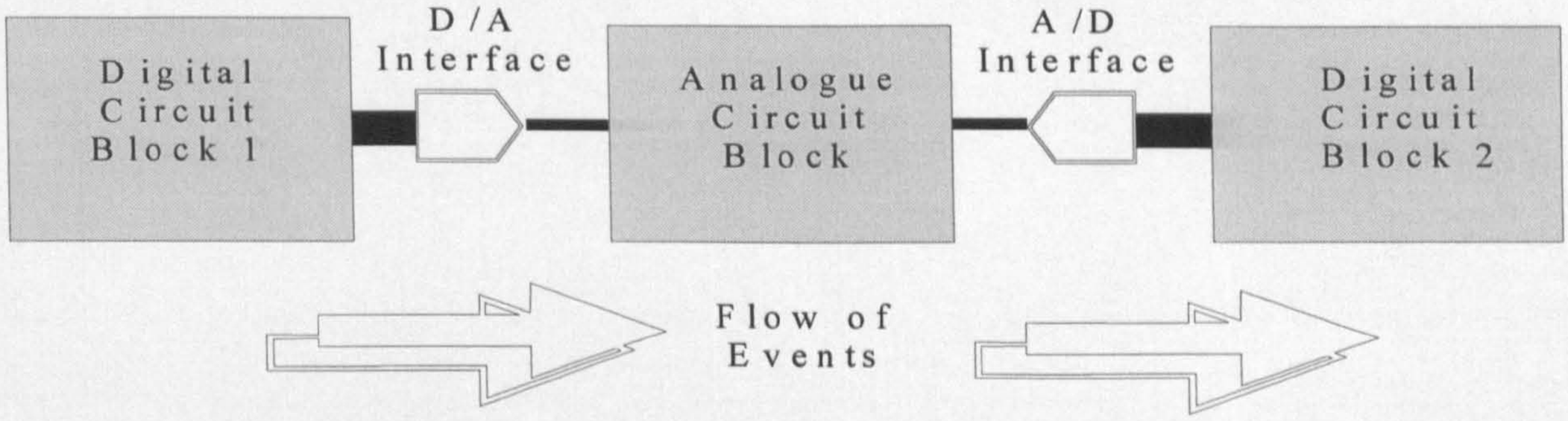


Figure 2-2. Data Flow in sequential Mixed-Signal Simulator

circuit path: A sequence of input vectors is processed to produce a sequence of output results. These results are used as the input vectors for the next circuit block as shown in Figure 2-2. This approach is valid provided there is no risk of the input vectors being altered by feedback from a later stage. Sequential simulators therefore work best where there is no feedback between analogue and digital parts or vice versa. This approach is rarely used in practice since most mixed signal IC's include some feedback paths between analogue and digital sections. A commercial sequential simulator called 'A/D Lab' was released by Daisy [11] as part of their suite of design tools but is no longer available.

The **paired approach** couples the analogue and digital simulators together more tightly to enable feedback to be simulated. The main difference between this and the sequential approach is that the analogue and digital simulators are run concurrently (i.e. parallel

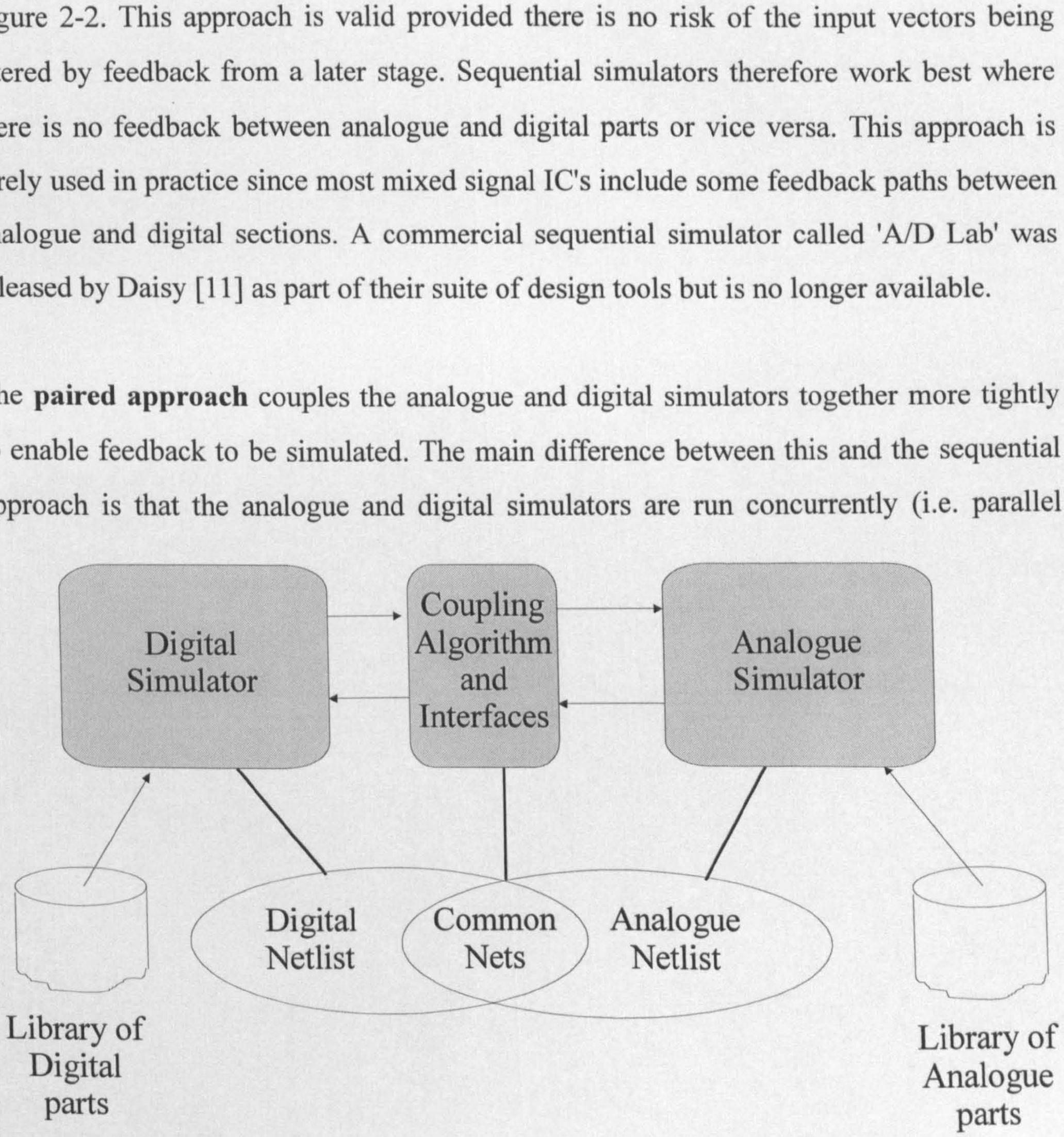


Figure 2-3. Paired Mixed-Signal Simulator Architecture.

processes). This requires a computing environment that can support parallel processing: usually a UNIX based operating system running on a workstation (e.g. Sun, HP-Apollo). The structure of a typical paired mixed-signal simulator is shown in Figure 2-3.

Since the analogue and digital simulators are both running at the same time, they must be synchronised whenever data needs to be transferred from one to the other. One simulator will normally tend to run ahead of the other. Synchronisation therefore requires the fastest simulator to be stopped and its simulation time reset to the same point as the slower simulator. The simulator that will process a particular circuit most efficiently depends on the size and nature of the analogue and digital parts. It is generally easier to stop and back-track a digital simulator since lists of digital states are simpler to regenerate than analogue voltage and current waveforms. The ability to back-track requires all intermediate results to be stored. This can require large amounts of disk storage and can produce a very inefficient simulator if a lot of back-tracking is required. There are two main synchronisation methods used in commercial mixed-signal simulators to address this problem. One method is known as **'lock step'** synchronisation. This locks the timebases of both analogue and digital simulators tightly together so that neither one can get substantially ahead of the other. This method is best where there is a large amount of interaction between the analogue and digital components, i.e. a large number of results must be passed between the two simulators. It is used in coupled mixed-signal simulators from Cadence (Verilog + Cadence SPICE), Viewlogic (ViewSim + PSPICE) and Genrad (SHADO: System Hilo + Eldo). The other method of synchronisation is known as **'leap frog'**. This allows each simulator greater independence: each simulator is allowed to run until it encounters an 'event' from the other one. This looser coupling enables circuits with less interaction between the digital and analogue sections to be simulated more efficiently. It also simplifies the integration of analogue and digital simulators (often from different CAD vendors) into a single process. The problems associated with back-tracking limit this method's performance when there is significant interaction between the analogue and digital sections. The best known example of this technique is the patented 'Calaveras' algorithm [12] that saves information about the previous states of analogue nodes to reduce the amount of analogue matrix re-evaluation required during back-tracking. It is used in mixed-signal simulators based on the Saber analogue simulator

from Analogy (e.g. Saber + ViewSim (Viewlogic), Saber + Verilog (Cadence or Mentor)).

The **nested approach** couples two or more simulator 'engines' together under the control of a single manager process. The simulator engines each implement different algorithms so that the various parts of a mixed-signal system can be analysed using the most appropriate method. The main difference between this approach and the paired approach is that the simulation manager processes the circuit description and waveforms and invokes the most appropriate engine only when it is required. The simulation manager therefore controls the simulation, passing and receiving data from the different algorithms in the same manner as subroutines are called from a main program in conventional high-level programming languages. A nested simulator can be implemented as a self-contained program with the algorithms and data transfer completely hidden. Alternatively, one can be implemented with looser coupling between the algorithms by using UNIX 'sockets' for data transfer. The simulation engines are then implemented as separate programs although still invoked and controlled by the simulation manager. The structure of a nested mixed-signal simulator is shown in Figure 2-4.

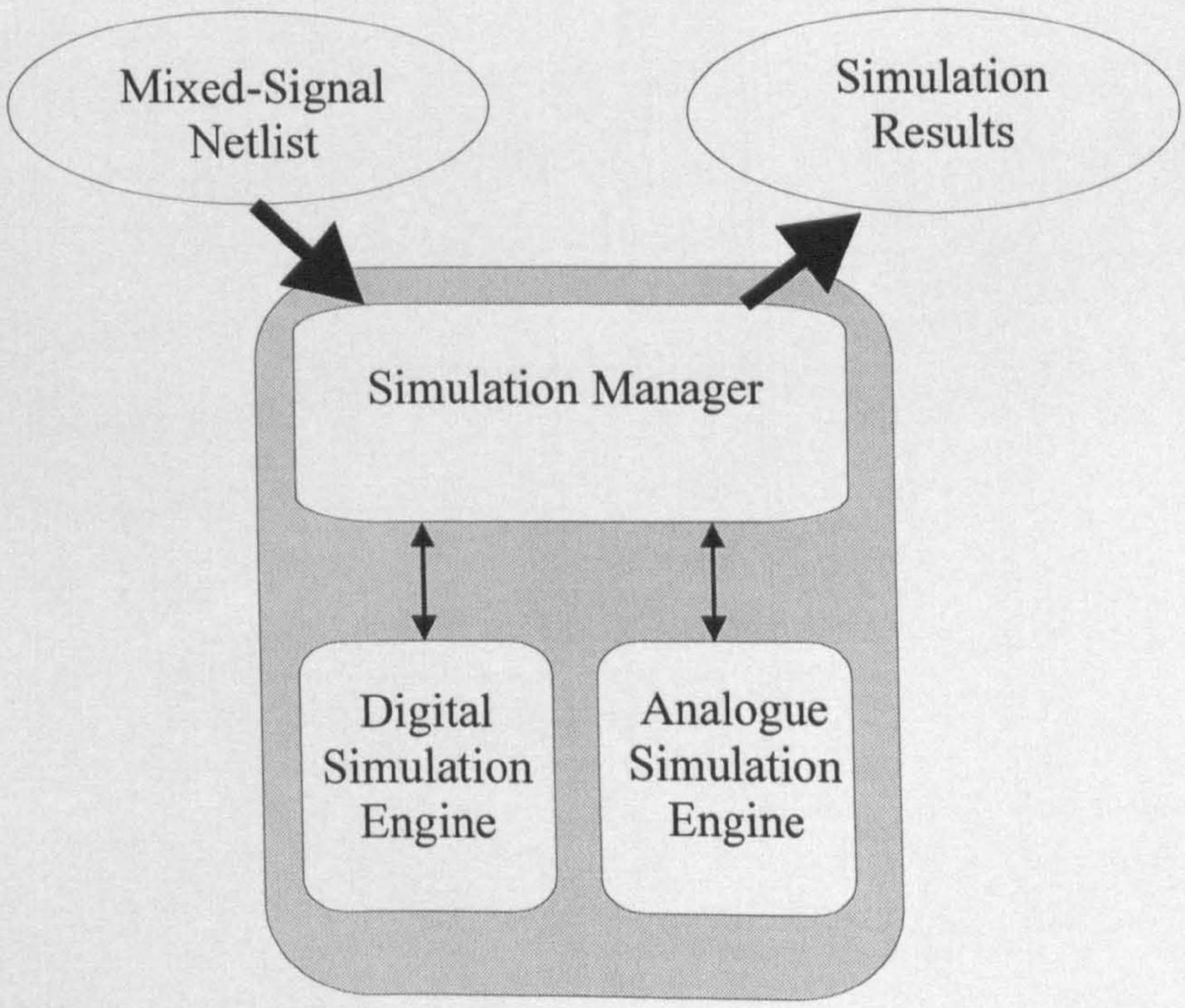


Figure 2-4. Nested Mixed-Signal Simulator Architecture.

The use of sockets produces a more flexible nested simulator as it is relatively simple to add, remove or update individual simulation engines. It can also significantly increase the simulation speed if the simulation engines exist across a network of computers. Different simulation engines can then process different parts of the system simultaneously on separate computers. Sockets can also be used to interface hardware modelers and hardware accelerators to the simulator. These are devices that allow physical (rather than software) models of components to be used. This can result in huge increases in simulation speed where large, complex standard parts (such as a CPU) are included in a system. This is significant since silicon masks for an increasing number of CPU cores are available for inclusion in ASIC designs from several vendors.

The main disadvantage of using sockets is the time penalty involved whenever data is transferred. The effect of this is even greater if the data has to be transmitted over a network. This method is therefore best where the amount of data that needs to be transferred between the different circuit sections is small, i.e. there is little global feedback between sections.

An example of a nested mixed-signal simulator is the combination of Meta-Software's HSPICE analogue simulator with Silicon Compiler Systems' (SCS - now part of Mentor Graphics) LSIM extended digital core simulator [13]. LSIM includes algorithms for analysing behavioural digital and analogue models. Running LSIM and HSPICE in parallel on separate CPUs enables the simulation manager to maintain a level of accuracy comparable to coupled simulators but with a large improvement in execution time. This technique is therefore better suited to the simulation of entire VLSI devices where the higher cost of the simulation tools is offset by the reduction in (expensive) CPU time.

A **framework** approach is the latest technology to appear in commercial CAD tools. The concept of a framework is that all of the CAD tools required for the design process are integrated into a single environment and address a common design database [14]. The ideal framework would allow the end-user to select and integrate tools from any vendor into a customised design environment with a consistent look and 'feel'. The concept is similar to the window-based environments found on personal computers but much more

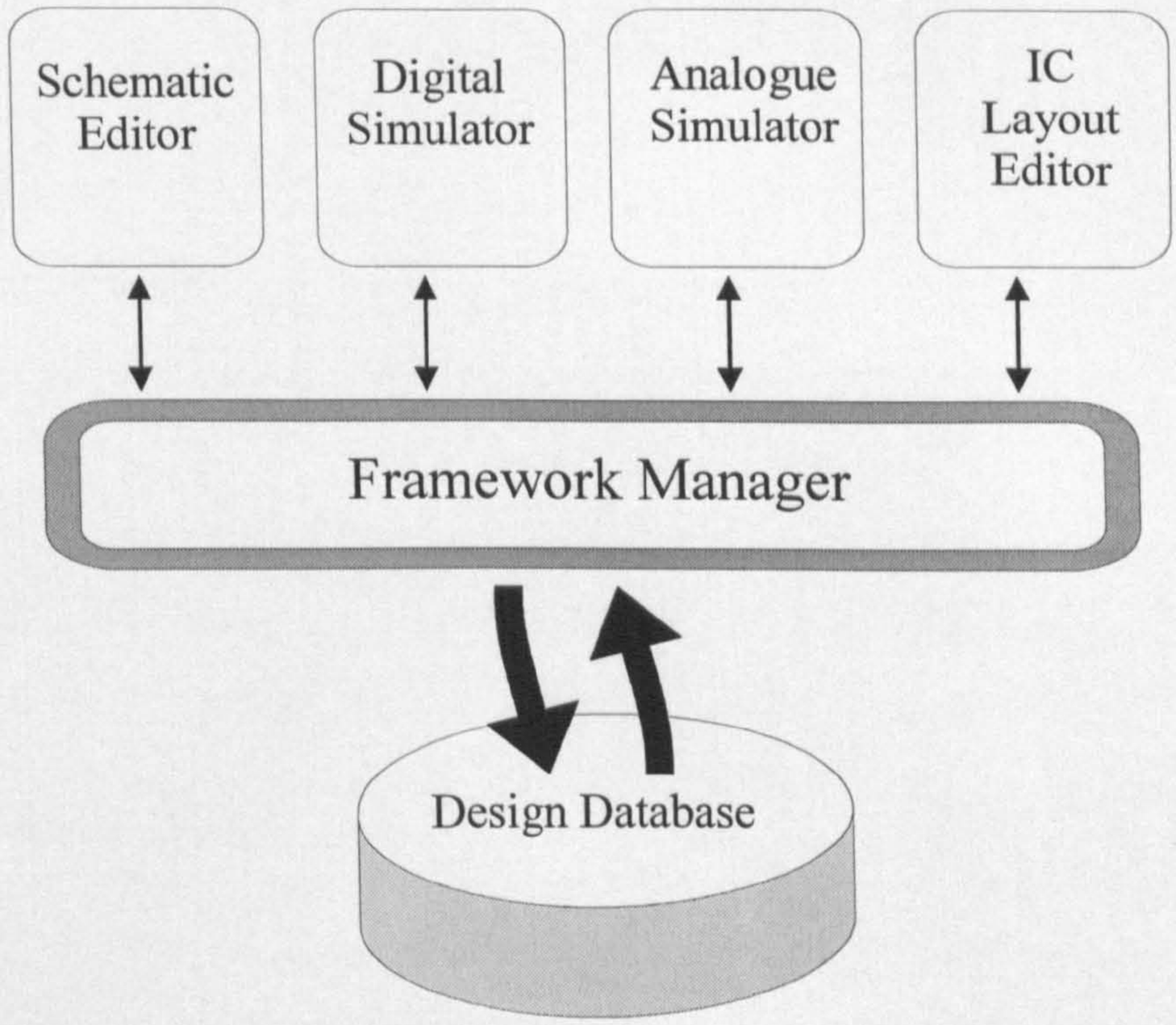


Figure 2-5. A CAD Framework.

powerful. Access to the design database and control of the tools is controlled by a framework manager process. This can be viewed as an extension of the simulation manager process found in nested simulators. The architecture of a framework is shown in Figure 2-5.

A framework should support the integration of a variety of simulators. The framework manager should automatically select the most appropriate one to simulate a particular design. A CAD tool must possess standard control and data interfaces before it can be integrated into a framework. Unfortunately there are several incompatible framework standards currently in use and a certain reluctance of CAD vendors to adopt the standards used by their competitors.

The two largest electronic CAD companies each have their own framework, Falcon Framework from Mentor Graphics and Analog Artist from Cadence. Mentor Graphics has a mixed-signal simulator known as "Continuum" that runs under their framework. It is based on their QuickSim II digital simulator and AccuSim II analogue simulator (acquired from Anacad). Continuum supports switch-level to behavioural-level (VHDL) digital models and circuit-level (SPICE) to behavioural-level (HDL-A) analogue models. The

use of a framework means that Continuum has a common user interface for both simulation engines and automates the partitioning of a mixed-signal design into analogue and digital sections. There are currently only a few third party tools that are compatible with either the Mentor Graphics or Cadence frameworks. This situation is likely to improve as framework standards are approved and more widely adopted.

Applications that contain predominately analogue or predominately digital circuitry may be simulated more efficiently by **extended core simulators**. These are analogue or digital simulators that have been "extended" with algorithms to process the other domain as shown in Figure 2-6.

Circuits that are predominantly analogue may be simulated more efficiently by extended analogue core simulators. These use analogue behavioural models of the digital circuitry and so offer improved performance over pure analogue simulators whilst still maintaining the same degree of accuracy. Analogue simulators have the ability to perform frequency domain analysis, which can be useful for evaluating certain mixed signal designs. The usefulness of the frequency domain analysis is dependent on the accuracy of the models of the analogue-to-digital interface. The simulation will become less efficient as the proportion of digital to analogue circuitry increases. This is because each digital model will have to be evaluated at every time-step even when no changes of state have taken place. The time-steps will normally be set to a relatively small value (compared to those used in digital simulators) to keep the analogue simulation accurate.

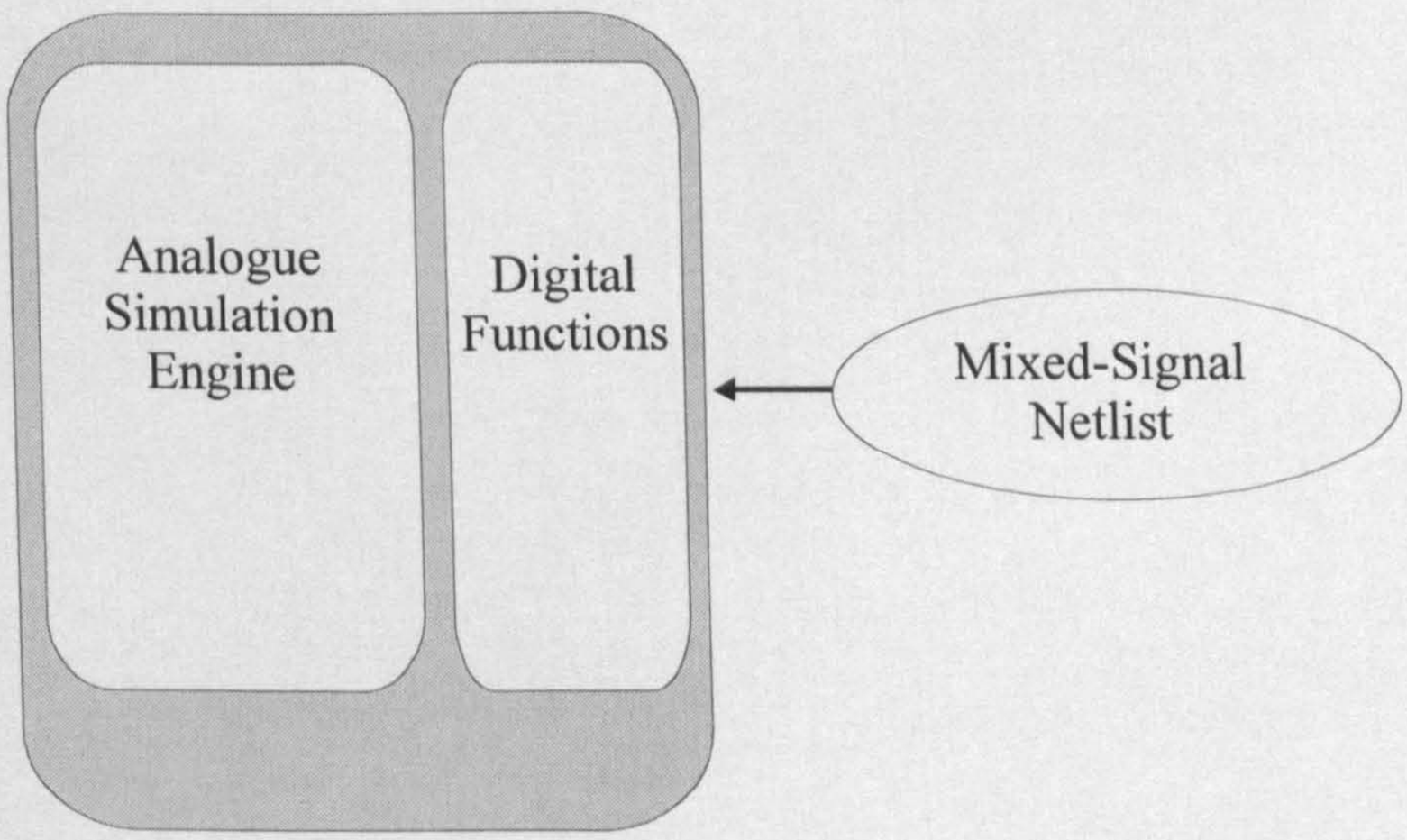


Figure 2-6. Extended Analogue Core Mixed-Signal Simulator.

Saber (from Analogy) is an extended analogue core simulator that was first released in 1985. Saber can operate at a behavioural level as well as a SPICE compatible level. The performance of Saber as a mixed-signal simulator is therefore closer to that of coupled simulators. Saber models consist of differential equations rather than the voltage sources, current sources and semiconductor devices used in SPICE. The models are defined using a proprietary analogue hardware description language (HDL) MAST [15]. Since the models are defined in terms of integral and differential equations they are not limited to electronic components but can also represent other system components whose behaviour can be expressed in such terms. The behaviour of an integrated circuit in its intended environment can therefore be investigated at the design stage using this tool. Saber and the MAST language support simulation of digital components at the behavioural, structural and gate levels so it can claim to be an integrated simulator. Despite this capability, it is used in several coupled mixed-signal simulators as the analogue engine (e.g. Saber-Verilog, Saber-ViewSim and Saber-QuickSim). This reflects the higher efficiency that is provided by coupled simulators when evaluating large, predominately digital circuits.

Extended analogue core simulators based on SPICE have become more common as the processing power of personal computers (PC's) has approached that of UNIX workstations. PSPICE (from Microsim) is one of best known of this category of simulator. It features digital models with A/D and D/A interfaces. As a mixed-signal simulator it is best suited to small sub-systems such as phase-locked loops and data converters. For larger circuits, it is better to couple PSPICE to a digital simulator (such as ViewSim).

IsSPICE4 (from Intusoft) is another extended core analogue simulator based on the SPICE 3F [16] algorithms. It has recently been enhanced with an analogue hardware description language capability. This HDL enables models to be written using the "C" programming language. The Intusoft HDL enables IsSPICE4 to process behavioural models of analogue and digital components. It is based on the XSPICE program [17]

developed at the Georgia Institute of Technology rather than on the proposed VHDL-A standard.

The most recently introduced extended analogue core simulator is SMASH from Dolphin Integration [18]. It is available for both PC's and UNIX workstations. SMASH also has a proprietary hardware description language based on the C programming language. It supports analogue behavioural level models based on transfer functions (e.g. Laplace) as well as standard SPICE models. SMASH can perform transient analysis using direct methods (as used in SPICE) or more efficient (but less accurate) relaxation methods. Digital sub-circuits are evaluated using an integrated 12-state, event-driven algorithm. The digital simulation works at the switch-level for MOS circuits or can use behavioural models written in C. Dolphin Integration have announced that digital models written in the Verilog hardware description language will be supported in future releases.

Extended digital core simulators can hold voltage and current values on their time wheel as well as digital events. Analogue waveforms are therefore represented as a series of discrete steps. As previously mentioned, LSIM is an extended digital core simulator, it includes a circuit-level simulation algorithm called 'Adept' that uses SPICE 2G.6 level II and level III models [19]. Adept trades the accuracy of SPICE simulation against improved execution time. LSIM also features a behavioural language 'M' based on the C programming language. This can be used to model both analogue and digital circuits. LSIM has been used by Sierra Semiconductors (a silicon vendor) as the basis for their Montage simulator. Montage forms part of their "in-house" design system and has been used for behavioural simulation of complete ASICs containing over 200,000 gates [20].

The extended digital core approach is better suited to high level behavioural simulation than the detailed electrical level simulation offered by extended analogue core simulators. Trading analogue accuracy against execution speed makes chip-level simulation possible. The approximations inherent in the analogue behavioural models are less significant if this type of simulator is restricted to standard cell type devices.

An ideal **integrated** mixed-signal simulator processes a single circuit description and generates a single output file. All of the processing is done within a single program so no time is wasted in synchronising separate analogue and digital simulation processes or programs. This is therefore the most efficient type of mixed-signal simulator. Unfortunately such a simulator has not yet been built and the techniques required to implement one are still in the research domain. The main problem lies in the different nature of analogue and digital models. The algorithms chosen to realise an integrated simulator normally give a preference to either analogue or digital models, at the expense of the other type. Since the models must use an integrated data format, models from existing simulators will not be compatible with integrated simulators. This means that a large effort would have to be spent on creating libraries of component models before a commercial simulator could be released. A commercial integrated simulator ANDI was released by Silvar-Lisco in the mid 1980's. The performance and capabilities of ANDI were surpassed when coupled simulators became available. As a result, this simulator is not widely used any more. CAD vendors have since concentrated on coupling existing simulators together rather than developing new integrated simulators because of reduced development costs and greater flexibility [21].

2.5 Experimental Mixed Signal Simulators and Simulation Methodologies.

There has been a substantial amount of research in the field of mixed-signal simulation since the mid 1970's. Much of the research work has been performed by post-graduate students working towards Masters degrees and Doctorates in universities and research laboratories. This has led to a number of new simulation methodologies and experimental mixed-signal simulators. Comparatively few of these methodologies have been incorporated into commercial tools. This is a result of the long development times and huge programming effort required to launch a new commercial simulator rather than a reflection on the quality of the published research. The major developments in this area of research are reviewed below.

Several researchers have attempted to produce an integrated mixed-signal simulator by expanding a standard SPICE simulator [1]. Allen and Zuberek [22] expanded a SPICE-compatible circuit simulator to allow the description of parameterized analogue to digital and digital to analogue interfaces. This was then enhanced by including gate-level event-driven digital simulation algorithms. The circuit was described to the simulator using an enhanced version of the SPICE netlist language that included functions for basic 2- and 3-input logic gates. This technique produced a significant reduction in simulation time compared to conventional circuit simulation for a number of analogue to digital converter circuits. It is not suitable for the simulation of large mixed-signal circuits because of the limitations of the SPICE algorithms.

Chain [23] has integrated a SPICE-type simulator with a switched capacitor simulator (Spice-SCAN). This is an 'in-house' tool that has been developed for silicon vendor Harris Semiconductor. It performs transient analysis of switched capacitor circuits. Although switched capacitor circuits are widely used as building blocks in mixed-signal ASICs they are not modelled well in conventional mixed-signal simulators. Spice-SCAN uses algorithms that are much more efficient than SPICE to model charge conservation between the switched capacitors without loss of accuracy. It is designed to simulate circuits with non-deterministic clocks that can't be simulated by other switched capacitor simulators. The example application given in the reference simulated several orders of magnitude faster than a transistor level SPICE simulation and about ten times faster than a SPICE simulation using macro-models. Chain claims it is possible to integrate Spice-SCAN into a Cadence framework using the Verilog digital simulator to give a complete mixed analogue/digital/switched capacitor simulator.

Other researchers have investigated alternatives to SPICE-type nodal analysis for performing circuit simulation. This has resulted in new methodologies known as **relaxation techniques** [24]. These techniques make use of the unidirectional characteristics of MOS transistors - the gate terminal (input) is insulated from the source and drain terminals (outputs). This means that the current in the gate circuit is independent of the voltages at the other terminals (neglecting parasitic capacitance

effects). The gate and source-drain terminals can therefore be considered as belonging to separate circuits that can be evaluated independently.

Relaxation techniques overcome the problems of the direct solution methods used in SPICE by allowing a large circuit to be split into smaller blocks that can be processed separately. The matrices for the separate blocks will be solved more efficiently. The integration time step is not required to be the same for all blocks since it is only dependent on those signals within a block. The optimum step size can therefore be chosen for each block without affecting the efficiency of the integration in other blocks. This makes relaxation techniques more efficient at evaluating large circuits than SPICE.

The first circuit simulator to use relaxation techniques was MOTIS [25] developed by Chawla, Gummel and Kozah. This simulator also featured look-up tables to implement transistor models and was up to two orders of magnitude faster than SPICE-type simulators. It was designed to verify the timing of signals in MOS circuits and was therefore referred to as a 'timing simulator'. The efficiency of this technique was improved in the SPLICE series of simulators [26]. SPLICE introduced a selective-trace algorithm that automatically bypassed inactive circuit nodes during the integration process. The integration methods were also modified to improve the convergence properties. This class of relaxation based simulation is known as **iterated timing analysis** (ITA).

Research into ITA techniques led to a variation known as one step relaxation (OSR) [27]. This was used in the commercial Eldo analogue simulator from Anacad. Benkoski et al. have integrated Eldo with an industrial multi-level digital simulator to form an experimental tightly coupled mixed signal simulator Mozart-MM [28]. This uses a lockstep algorithm to synchronise the analogue and digital simulation engines and therefore falls into the same category as the SHADO commercial simulator already mentioned (although it predates SHADO and doesn't directly support VHDL).

Another class of relaxation techniques attempts to directly solve the differential equations associated with each circuit node. This is known as **waveform relaxation** (WR) [29]. This method assumes that the circuit equations can be broken into blocks such that if the blocks are solved in the proper order, a good approximate solution to the entire system is

obtained. The effect of transitions at the inputs of a circuit ripple through successive blocks and can be considered as a series of wavefronts propagating through the circuit. The accuracy of the initial approximation can usually be increased by a second iteration. Subsequent iterations will tend to converge rapidly to the exact solution. Since relaxation methods assume loose coupling between nodes, circuits with strong coupling (e.g. high gain feedback) will converge very slowly. In such cases, the closely coupled nodes are best evaluated using direct methods. Waveform relaxation simulators such as RELAX2 [24] therefore use partitioning algorithms to group large MOS circuits into suitable blocks before performing the simulation.

The efficiency of waveform relaxation simulators at simulating digital functions is still significantly less than that achieved by conventional digital simulators. They are normally used for large scale analogue simulation and where the higher accuracy is required for digital circuits (e.g. for timing analysis) rather than as the basis for integrated mixed-signal simulators.

Relaxation based techniques are well suited for parallel processing since the decomposed circuit blocks are loosely coupled and can be simulated independently. Waveform Relaxation and Waveform Relaxation Newton (a combination of WR with conventional Newton numerical integration) algorithms have been combined by Odent et al. for use with a general purpose parallel computer [30]. This resulted in a substantial reduction in execution time compared with standard sequential algorithms. Parallel processing is desirable for large mixed-signal systems with a significant proportion of analogue circuitry or digital circuitry requiring accurate timing analysis. This would however require that the architecture of the mixed-signal simulator was well suited to a parallel processing environment. These techniques are therefore most likely to be of use in framework based mixed-signal simulators.

Several researchers [26][31] have developed timing simulators to a point where they could be used as mixed-signal simulators for MOS circuits. Timing simulators are designed to provide accurate simulation of digital circuits by using more accurate models for the transistors than switch-level simulators. They make use of the event-driven nature

of digital circuits to increase the performance over SPICE type circuit simulators. The experimental SPLICE simulators and the ADEPT algorithm in the commercial LSIM simulator have already been mentioned. They both fall into this category.

The MOTIS timing simulator has undergone two major revisions known as MOTIS2 and MOTIS3 [26]. Each revision has improved the algorithms to increase performance and the range of circuits that can be simulated. Chadha et al. have developed a unified multilevel mixed-signal simulation system known as M^3 based upon MOTIS3 [31]. This system consists of a set of tools to create simulator models together with the simulator control program. M^3 is designed to model analogue circuitry at the behavioural level. The models are written in the C programming language. Since this provides considerable scope for error, a model verification tool ACME was also developed to test the model behaviour prior to use in the simulator. Two automatic model generation tools were developed to generate models from *s*-domain and *z*-domain descriptions since the behaviour of analogue filters is often specified in the frequency domain. The behavioural models can have voltage, current and Boolean terminals. The voltage and current terminals are bi-directional so coupling and loading effects can be simulated. Boolean terminals are unidirectional, i.e. they are either inputs or outputs. The models consist of controlled and independent voltage and current sources that are evaluated using the relaxation algorithms in MOTIS3.

Event-EMU is an experimental timing simulator developed by Ackland and Clark [32]. It combines a mixture of event driven and relaxation techniques to provide a higher performance than circuit simulators with no significant loss of accuracy. Event-EMU doesn't directly support analogue components but could be adapted to do so since it includes models for MOS transistors, resistors, capacitors and current sources. Events occur when a node voltage changes by more than some specified threshold. Including an analogue modelling capability would therefore reduce the efficiency of Event-EMU since the threshold would inevitably have to be made smaller to preserve the accuracy.

Overhauser et al. developed IDSIM2, a timing simulator that could operate at multiple levels of abstraction and included an analogue modelling capability [33]. Analogue

circuits are simulated using detailed circuit simulation. Digital circuits are simulated using 'fast timing simulation'. Two methods of fast timing simulation are implemented: the first uses a ramp representation of waveforms while the second uses a more detailed piecewise linear (PWL) representation. The ramp representation is up to one order of magnitude faster than the PWL representation. A significant feature of this simulator is that when the waveforms in any subcircuit are detected to be "not accurate enough" within a certain time interval, the subcircuit is dynamically changed to a more accurate model. This means that at the end of a simulation, only those subcircuits that required the greatest accuracy would have been modelled at that level: the rest would have been modelled at the most efficient level. IDSIM2 uses waveform relaxation for simulation of both analogue and digital functions. PWL waveforms are used to interface subcircuits having different levels of analysis to each other. Since only (unidirectional) MOS circuits are considered, feedback loops can be solved more efficiently than with SPICE type simulators. Feedback loops are processed using one of three windowing techniques: partial waveform convergence; dynamic windowing; or a combination of both. The windowing techniques are chosen to provide the most efficient forward processing path whilst keeping the number of iterations required when feedback occurs to a reasonable level. The authors [33] report significant increases in performance compared to SPICE2.

ILLIADS is a timing simulator developed by Shih and Kang [34] that takes a different approach to IDSIM2. ILLIADS uses a new MOS circuit model and approximates electrical waveforms using PWL segments. The new model has a major advantage over those used in other timing simulators: parallel circuit branches (such as found in CMOS NAND and NOR gates) are not required to be merged to preserve accuracy. Improved waveform relaxation techniques are used to overcome the inefficiencies of standard WR methods when applied to feedback between strongly coupled components. Windowing techniques similar to IDSIM2 are also used (ILLIADS and IDSIM2 were both developed at around the same time in the University of Illinois). The authors report significant increases in performance compared to SPICE simulations for several digital circuits. No mention is made as to the suitability of this simulator for processing mixed-signal circuits.

A mixed-signal simulator based on a timing simulation approach has been developed by Jun and Hajj [35], also at the University of Illinois. This simulator features logic, timing and circuit simulation algorithms. The overall architecture is based on waveform relaxation with dynamic windowing to handle feedback (like IDSIM2 and ILLIADS). The logic simulation uses two logic states and user defined delay parameters. Timing simulation uses either empirical or analytical macromodels based on NMOS transistors. If the timing simulator detects glitches, it can automatically switch to using circuit simulation for a block in the same way as IDSIM2. The circuit simulation can use standard SPICE models for analogue blocks or can work at a higher level for digital blocks using a simplified MOSFET model. The simulator automatically inserts digital to analogue and analogue to digital converters between analogue and digital circuit blocks. The digital to analogue conversion is based on ramp waveforms. The analogue to digital conversion is based around four threshold voltages to simulate noise margins. If an analogue waveform is in between the thresholds for the two logic states, the digital block is automatically simulated at a circuit level. The authors report that this simulator runs up to three orders of magnitude faster than SPICE.

Several of the simulators mentioned above make use of **piecewise linear (PWL)** waveforms to improve the efficiency of signal representation. Piecewise linear techniques provide a generic and powerful approach to the modelling of electronic components. They have been used by several other researchers to create mixed-level and mixed-signal simulators. PWL waveforms support the use of macro models and mixed-levels of abstraction since all components and signals are modelled in a uniform manner.

Early work on PWL simulation formed a similar set of circuit equations to the nodal analysis (NA) methods used in SPICE type simulators [36]. These equations were solved using sparse matrix techniques and multirate integration methods. PWL simulators have two main advantages over SPICE type simulators. They have better convergence characteristics and support component modelling at all levels of abstraction.

Van Stiphout et al. implemented a PWL mixed-signal simulator called PLATO that used an event-driven approach for transient analysis [37]. The events are grouped into two

classes: PL events and dynamic events. PL events occur when a vector reaches the end of a PWL waveform segment. Dynamic events are generated if the integration step size for a particular model becomes invalid (the integration step sizes are adjusted to maintain accuracy and efficiency as the simulation progresses). The efficiency of this event-driven approach was increased by discretization of the event times thereby reducing the number of separate events. Further improvements to efficiency were made by forcing related circuit blocks to use the same minimum step size, reducing the recomputation of step sizes required. The authors present example applications together with program statistics but make no comparisons to the performance or accuracy of other simulators.

Kevenaar and Leenaerts use similar methods to PLATO in a PWL simulator called PLANET that exploits system hierarchy to run more efficiently [38]. Previous PWL simulators solved the circuit equations using a single, large matrix. PLANET partitions a large system into a set of subcircuits that can each be represented by a small matrix of PWL equations. Each subcircuit can be solved independently allowing the optimum integration timesteps to be used. The subcircuits are connected together by sets of topological equations that describe the system hierarchy. This approach also enables subcircuits to be replaced by behavioural models to further increase the efficiency. The advantages of this approach over non-hierarchical methods increase with the complexity of the simulated system. The authors show how the accuracy of an op amp macromodel can be comparable to SPICE.

Griffith and Nakhla have used piecewise linear waveforms in a novel simulator for non-linear frequency dependent circuits [39]. This simulator is designed for transient analysis of high speed circuits where improperly terminated connections can adversely affect the transmission of signals. These connections cannot be simulated correctly by conventional lumped impedance interconnect models and must use distributed transmission line models instead. Unfortunately, these models are only defined in the frequency domain. The simulator replaces non-linear terms in the circuit equations by a set of piecewise linear time-dependent waveforms. This reduces the non-linear equations to a linear equation that can be solved in the frequency domain. The transient response is obtained from the frequency domain solution using inverse Laplace transforms.

Cottrell used piecewise linear waveforms to create a behavioural mixed-signal simulator [40,41]. This simulator was based on LSI Logic's proprietary event-driven behavioural logic simulator BSIM. The analogue models represent the transfer functions associated with analogue circuit blocks such as gain blocks, filter sections, comparators and digital to analogue converters. The model ports are unidirectional and are either classified as inputs or outputs. Passive components such as resistors and capacitors can not be modelled by this simulator since their connections cannot be classified as inputs or outputs. Analogue signals are classified as voltage or current PWL waveforms and can be differential or referred to a common potential (ground). Transfer functions that are specified in the frequency domain must be converted to the time domain using Laplace transforms prior to simulation. They are then solved using the Forward Euler numerical integration method. The timesteps used by each model are controlled by monitoring the truncation errors in the PWL signal representations. Each model can therefore use the most appropriate timestep. The truncation errors arise since the forward Euler formulae are not perfect approximations of the Taylor series that give the correct results. The truncation error is proportional to the second derivative of the PWL waveforms. The simulator was tested with a successive approximation analogue to digital converter circuit. The results were compared with a simulation of the same circuit using HSPICE. The HSPICE simulation provided more detailed waveforms but took over 350 times as long to run. The simulator was also used to test an FM receiver circuit with good results.

Visweswariah and Rohrer have used piecewise linear and piecewise constant waveforms in a prototype event driven circuit simulator called SPECS [42]. This simulator uses empirical table models of I-V characteristics to represent electronic devices. Voltages are represented by PWL waveforms while currents are represented by piecewise constant waveforms. The simulator assumes that circuits only consist of independent sources, linear capacitors and non-energy storing two-port elements like resistors and diodes. A circuit consists of a tree of the independent sources and linear capacitors. Each link of the tree is unidirectional so a table model can be built to represent its I-V characteristics. All the table models are evaluated with a single event queue. Events correspond to the times that current waveforms change level and voltage waveforms change gradient. Feedback

loops cause events on the queue to be rescheduled in a similar way to event-driven digital simulators. Rescheduling may require a number of iterations before the circuit reaches a steady state. The methodology guarantees that the circuit will always converge to the correct state. The number of iterations required to converge is dependent on the coarseness of the steps (number of segments) in the table models. The size of the steps used in different parts of a circuit can be defined by the user to trade-off accuracy against efficiency. Simulation results from SPECS compare favourably with SPICE for digital MOS circuits. It is shown to run up to 200 times faster than SPICE, depending on the number of segments used in the models.

Ruan et al. used PWL waveforms to represent voltages in a functional simulator that formed part of an experimental multi-level simulator [43]. This functional simulator was designed to operate with both analogue and digital functional blocks. It was intended to provide the interface between a logic simulator and a circuit simulator in their multi-level simulation system. It uses an event-driven approach and predicts the time of each new event from the gradient of signal waveforms. Logic gates are modelled using a combination of AND, OR and NOT operations. These operations are designed to work with an arbitrary set of logic states: they operate directly on the voltage waveforms. The output of the AND operation corresponds to the minimum voltage in a set of input signals at any instant in time. The OR operation is similarly defined to give the maximum voltage from a set of input signals. The NOT operation subtracts the signal voltage from the sum of the logic 0 and logic 1 voltage levels (e.g. 5V). The voltage gain of a logic gate is assumed to amplify the rate of change of the output waveform until it reaches a limiting value. Capacitance in the gate fan out is assumed to attenuate the rate of change of the output waveform. The gate models generate an output waveform using the logic operations above. Rate amplification, limiting and attenuation functions are then applied. The models also apply a delay to the output waveform. This delay is a function of the rate of change of the input signal waveforms. Conventional digital simulators normally use a logic state 'X' to represent an unknown initial value at a logic gate output. Unknown states are usually created by an incorrect initialisation procedure. It is not possible to represent an unknown state in a simulator that only recognises voltage values. An algebra based on interval computation was developed to overcome this. An unknown state is

represented by a range of voltage values (an interval). The authors prove that Boolean algebra remains valid if a single value is replaced by an interval. This method has the added advantage that the intervals can be processed by arithmetic operations. The outputs of the models can therefore always be directly passed to analogue functional models. This overcomes another problem with mixed-signal simulators: how analogue models interpret 'X' states from digital models. The simulator was tested with several high level analogue and digital problems. The results were in close agreement with SPICE but ran up to several orders of magnitude faster.

Rsim [44] is an experimental switch-level simulator that can simulate large digital MOS circuits up to three orders of magnitude faster than SPICE. Piece-wise linear models have been added to Rsim by Kao and Horowitz to form a new simulator called Mom [45]. This simulator preserves the efficiency of Rsim for digital circuits but improves the accuracy so that the simulation results for certain "difficult" circuits approach those given by SPICE. The circuits considered included CMOS dynamic RAM sense amplifiers, emitter-coupled logic gates and BiCMOS buffers. These are all cases where the analogue characteristics of the transistors must be considered and so would usually require a mixed-signal simulator to be used. Mom is not suitable for circuit-level analysis since the methods used to evaluate the circuit equations loose their efficiency (compared to SPICE-type numerical integration methods) as the complexity of the piecewise linear models is increased to match the accuracy of SPICE models.

Several researchers have investigated the use of behavioural models of analogue functions in mixed-signal simulators. Behavioural models can be simulated much faster than the transistor level models used in both SPICE-type and relaxation based circuit simulators. Several of the simulators described above make use of analogue behavioural models [31, 40, 41, 15]. The ability to create behavioural models has been available in SPICE since the mid 1970's and is known as macro-modelling [46,47,48]. However, the efficiency of this approach is far less than is possible using a purpose designed behavioural simulator [49]. It is possible to create behavioural models that feature similar levels of accuracy to transistor level models. Unfortunately such models require long development times and are only valid over narrow physical and electrical operating points. There are two main

application areas where the model development times can be justified. The first application is in top down system design where system functionality is the main consideration. This only requires generic models that can be optimised for execution speed and/or memory requirements. The other application area is ASIC design using cell libraries. This makes use of ready made analogue modules designed by specialist IC designers. Exhaustive simulation and testing of these modules will have already been done using SPICE type circuit analysis tools and physical testing of fabricated devices respectively. An ASIC vendor will therefore have all the information required to produce suitable behavioural models of the components in a cell library.

Rumsey and Sackett developed a mixed-signal behavioural simulator called AMP based around Laplace transforms [50, 51]. The simulator uses "black box" models that are constructed from a set of parameterised building blocks. The models can exist at three different levels of abstraction. The simplest level uses functional blocks with ideal behaviour (e.g. an ideal op amp). The next level represents parameterised macro-models. A model of an op amp at this level would include standard data sheet parameters such as offset, common mode gain, differential gain, etc. The most complex level uses models with tuned behaviours. This means that the parameters associated with the model would change to reflect the actual physical and electrical environment being simulated (e.g. the offset voltage parameter of an op amp changing with the simulated temperature and power supply). The simulator includes a tool called NETTOLA that automatically generates Laplace transforms from a netlist description of a network containing resistors, capacitors and inductors. AMP was designed to interface to an in-house Texas Instruments event-driven simulator. A linker (AML) was also developed to interface the models to commercial digital simulators.

Gielen et al. developed an analogue behavioural simulator intended for use in simulation and synthesis of mixed-signal systems [52]. A simulator for a synthesis environment must use generic models. The models are required to fully describe the behaviour of each system block as a black box. No assumptions can be made about the internal architecture of the black box. This simulator therefore consists of a general simulation engine and a library of behavioural models. Each model represents a distinctive function such as

sampling, quantizing, filtering, phase-locking, etc. The models can be parameterised to reflect the effect of using particular technologies. Most of the parameters are statistical and require information about average values, variance and covariance to be supplied. The simulator and models are implemented using object-oriented techniques. The models are objects that are instantiated and interconnected for each particular application. The behaviour of each model is described in the form of input-output functions, differential or difference equations and transfer functions in the frequency domain. The models are invoked and the simulation controlled by a simple event driven process. The authors plan to ultimately integrate this with a commercial simulator such as Saber. The performance of the simulator is illustrated by a statistical minimum-rank model of a Nyquist-rate analogue to digital converter. This showed the effect of signal amplitude and channel mismatches on the noise performance of the converter.

2.6 Conclusions.

There has been much research and development activity in the area of mixed-signal simulators since the need for such tools became apparent. This activity has been driven by continuing developments in microelectronics technology that have enabled progressively larger and more complex mixed-signal systems to be implemented on a single integrated circuit. Developments in computer technology and programming languages have enabled highly complex simulation methodologies and tools to be developed that would not have been possible at the time when simulators such as SPICE were being developed (e.g. parallel processing, distributed systems and CAD Framework environments). Despite these developments, an integrated mixed-signal simulator that is well-suited to the design of large mixed-signal circuits has yet to be released.

Commercial CAD tool vendors have tended to create mixed-signal simulators by coupling an existing digital simulator to an existing analogue simulator using a variety of approaches. This approach has the advantage of being able to reuse existing component models but it does not produce the most efficient simulator. Extending a circuit-level simulator to include digital models has become popular in recent years but is only suitable for relatively small circuits (not large, predominately digital ASICs). Experimental

mixed-signal simulators have been developed using techniques originally devised for digital MOS circuits such as Waveform Relaxation. These have achieved promising results for a limited range of applications but cannot claim to be universal mixed-signal simulators.

Several researchers have investigated the use of piecewise linear (PWL) waveforms to represent signals. PWL waveforms can represent both analogue and digital signals and so could provide a uniform data format in an integrated mixed-signal simulator. This might overcome the problems of signal conversion and time synchronisation that exist in current mixed signal simulators. PWL waveforms also lead to very efficient numerical integration methods and so have the potential to reduce the time taken to solve the differential equations that describe most analogue circuits.

Behavioural-level simulators offer a way of investigating a large-scale design since they are concerned with functional blocks rather than individual transistors. Behavioural simulators have become common for digital system design using hardware description languages such as VHDL or Verilog. There are several analogue simulators that currently support behavioural-level modelling using proprietary HDLs. Analogue behavioural modelling methods that can match the accuracy of circuit-level simulation require further development. Analogue extensions to VHDL will lead to mixed-signal simulators that can work at a behavioural level for analogue and digital blocks using a common modelling language. However, for the foreseeable future these will still not be "integrated" mixed-signal simulators but based on existing coupled-simulator approaches.

3. Development of Modelling Techniques.

3.1 Introduction.

The efficiency of any simulator depends on the nature of the simulation models and the signals that are passed between them. Digital simulators are only concerned with discrete voltage levels (logic states). They can therefore represent signals much more efficiently than analogue simulators which require signals to be continuous. An ideal mixed signal simulator would represent analogue voltages and currents as efficiently as the logic states in a digital simulator. Piece-wise linear (PWL) waveforms provide an efficient method of representing analogue signals and can also be used to represent digital states. This chapter describes the development of modelling techniques for analogue, digital and mixed-signal components based on optimised PWL waveforms.

3.2 Representation of Signals.

3.2.1 PWL Representation of Digital Signals.

Piece-wise linear (PWL) waveforms represent changing signals as a series of connected linear segments. They are commonly used in SPICE-type simulators to represent digital signals as a series of pulses with finite rise and fall times. A typical PWL representation of a digital signal is shown in Figure 3-1. This waveform would be defined by the following set of points:

(0,0), (5,0), (6,5), (10,5), (11,0), (15,0), (16,5), (25,5), (26,0), (30,0).

The first ordinate in each pair defines a point in simulation time whilst the second defines the signal magnitude at that time. The time ordinate must increase between every point in the PWL waveform: since the waveform is represented by linear segments joining adjacent points, a decrease in the time would have no physical meaning. Two co-ordinates are associated with each signal transition, these specify the beginning and the end of the transition period. The PWL representation requires twice the amount of data to be stored compared to a conventional representation of a digital signal (a series of transition times and logic states). Consequently, PWL waveforms are not usually used in digital

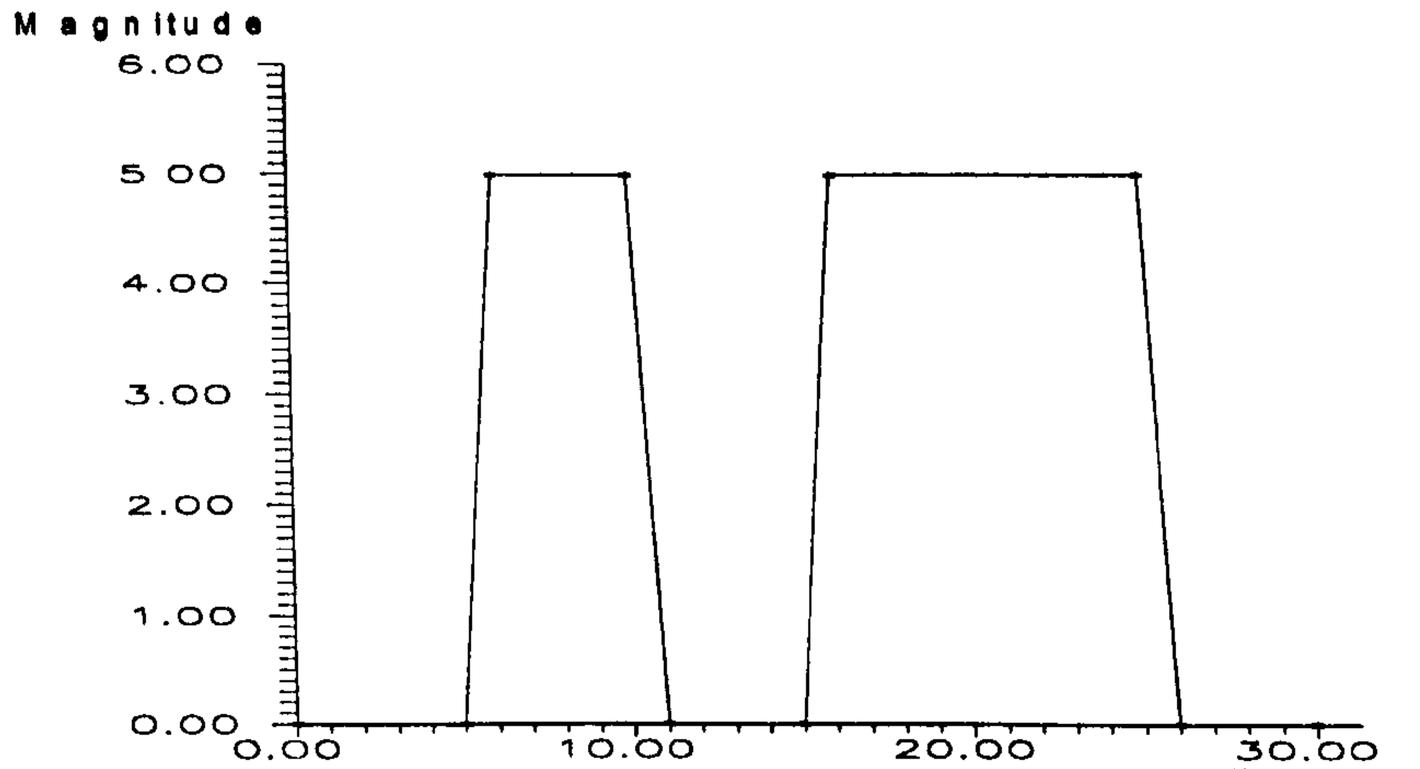


Figure 3-1. Piece-wise Linear (PWL) Representation of a Digital Signal.

simulators, even though they provide more information about the switching characteristics.

Digital signals are usually assumed to have a discontinuous nature, i.e. changes between logic states occur instantaneously. To represent this as a PWL waveform the requirement for time to increase between points must be relaxed to allow the time to also remain constant between a pair of points. Digital signals only exist in discrete states. The magnitude of a digital signal therefore remains constant between state transitions. A PWL representation that allows instantaneous changes of magnitude and where the magnitude remains constant between changes is known as a Piece-wise Constant (PWC) waveform. A PWC representation of the signal in Figure 3-1 is given in Figure 3-2.

The Piece-wise constant waveform in Figure 3-2 has redundant points: since signals are

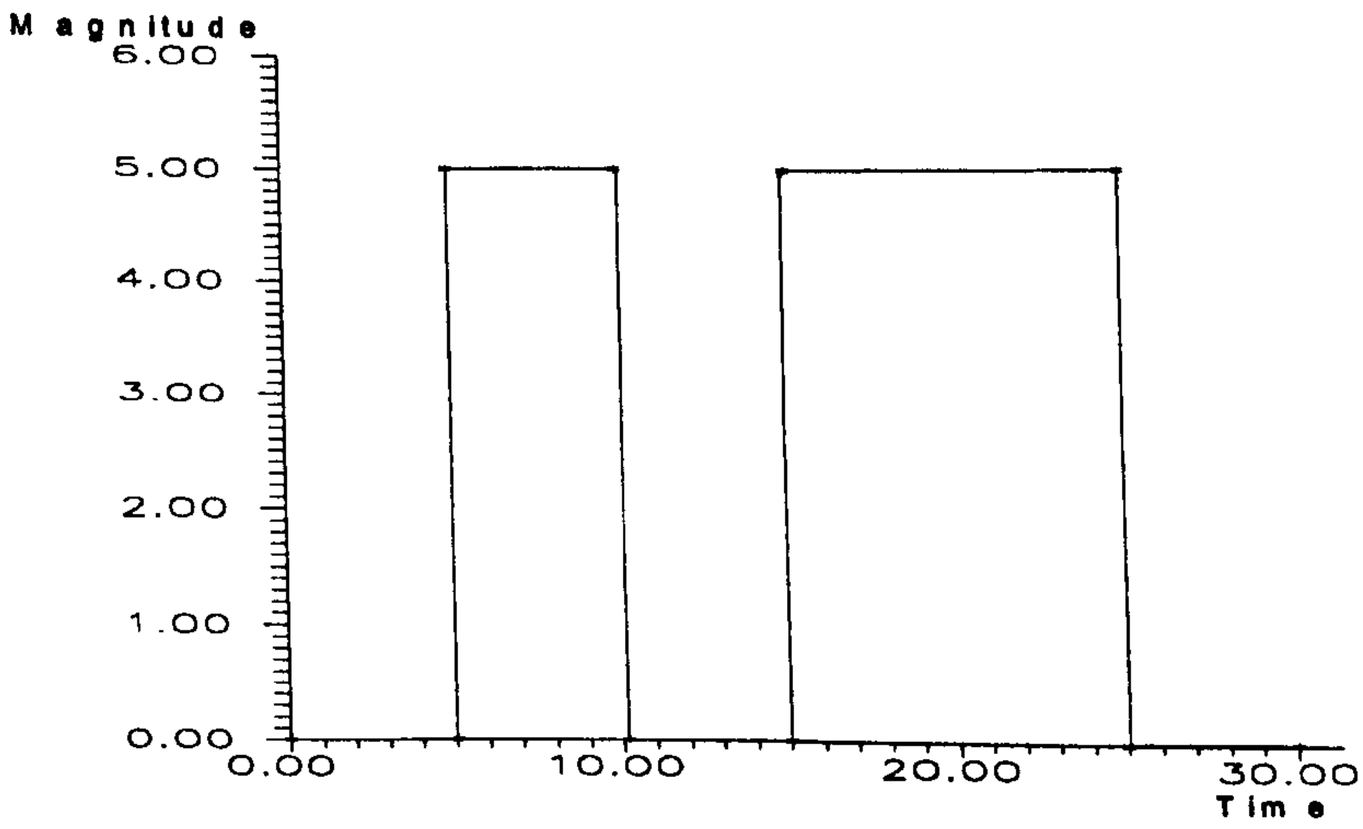


Figure 3-2. A Piece-wise Constant (PWC) Signal.

assumed to be constant within each segment, the co-ordinates defining the end of a segment can be derived from the magnitude at the start of the segment and the time at the start of the next segment. The amount of data required to represent a signal in a PWC format is therefore comparable to that required by conventional digital representations.

The effects of capacitance are significant in microelectronic systems. A piece-wise constant waveform will therefore provide a poor representation of voltage signals. However, since the effects of inductance are negligible for most semiconductor devices, PWC waveforms can provide an accurate representation of the currents that flow in a digital system [42].

Two methods of representing digital signals were investigated:

1. An efficient representation for use within completely digital blocks using a PWC representation. This is compatible with digital event-driven simulation methodologies.
2. An accurate representation for use where the analogue characteristics are important using a PWL representation.

A piece-wise constant representation cannot be used with analogue simulation algorithms that require signals to be continuous. However it is simple to convert a PWC waveform to a PWL waveform. The most efficient conversion method is to assign finite rise and fall times to each transition in the PWC waveform. A more accurate conversion technique is to consider the PWC waveform to represent the output current of a digital model. The corresponding PWL voltage waveform can then be obtained by considering the equivalent RC circuit of the output stage.

3.2.2 PWL Representation of Analogue Signals.

The selection of an optimum set of points to represent a constantly varying signal is a more complex operation than the assignment of points to represent a digital signal. A sinusoidal waveform is the most extreme example of a varying signal for representation in a PWL form since the rate at which it changes is never constant, i.e. it has **no** linear segments.

One method of placing the points representing a continuously varying waveform is to use a fixed time interval (time step) *delta* between each point. This is analogous to sampling the waveform in the time domain with a sampling frequency of 1/*delta*. The Sampling Theorem states that a signal of frequency f_h can be completely specified by a uniform sampling rate greater than $2f_h$. This requires the time step *delta* to be less than half of the period of the highest frequency component of any signals represented. However, accurate representation of a sine wave will require the time steps to be far smaller than those suggested by the Sampling Theorem since the points are connected by linear segments: a PWL representation of a sinusoidal waveform with only two points per cycle leads to a triangular waveform.

A PWL representation of part of a sinewave with 14 points per cycle is given in Figure 3-3. Comparison with a true sinewave shows that the PWL representation has introduced significant distortion into the waveform, most noticeably at the extremes where the variation of the sinewave is least linear. The deviation of a PWL representation from a pure sinewave decreases as the number of points per cycle is increased. The deviation becomes insignificant if more than 100 points per cycle are used.

The distortion resulting from the representation of a continuously varying signal by a

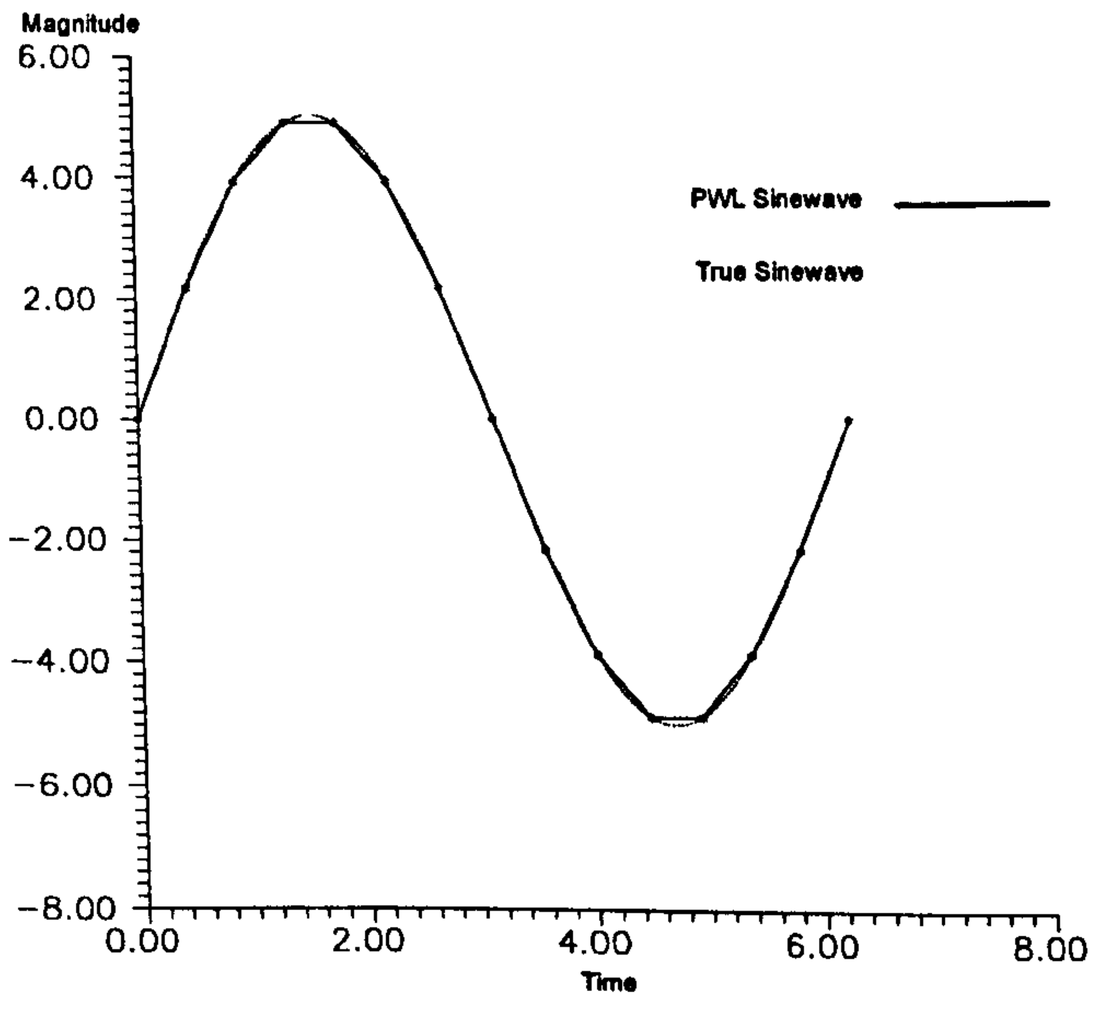


Figure 3-3. PWL Sinewave with Points at Fixed Time Steps.

finite number of linear segments can be studied using Fourier analysis. A series of PWL waveforms representing 10 cycles of a sinewave with various numbers of points per cycle (from 6 to 100) and uniform time steps between points was analysed using a 1024-point Fast Fourier Transform (FFT). The analysis found that the distortion mostly existed as a small number of harmonics. As the number of points per cycle was increased, the magnitude of these harmonics decreased whilst the order of the dominant harmonics increased. When 6 points per cycle were used, the significant harmonics were the 5th (4% of fundamental magnitude), the 7th (2%) and the 11th and 13th (approximately 1% each). When the number of points per cycle was increased to 25, the significant harmonics were the 24th and 26th (approximately 0.02% of the fundamental magnitude each).

The closeness of the PWL representation to a true sinewave is represented by the magnitude of the fundamental frequency produced by the FFT. The relationship between the number of points per cycle and the magnitude of the fundamental frequency is shown in Figure 3-4. The graph in Figure 3-4 gives an indication of how trade-offs could be made between the accuracy required and the efficiency of a PWL representation.

There is another form of distortion that can occur if there are not an integral number of time points representing each cycle. This distortion appears as amplitude modulation of

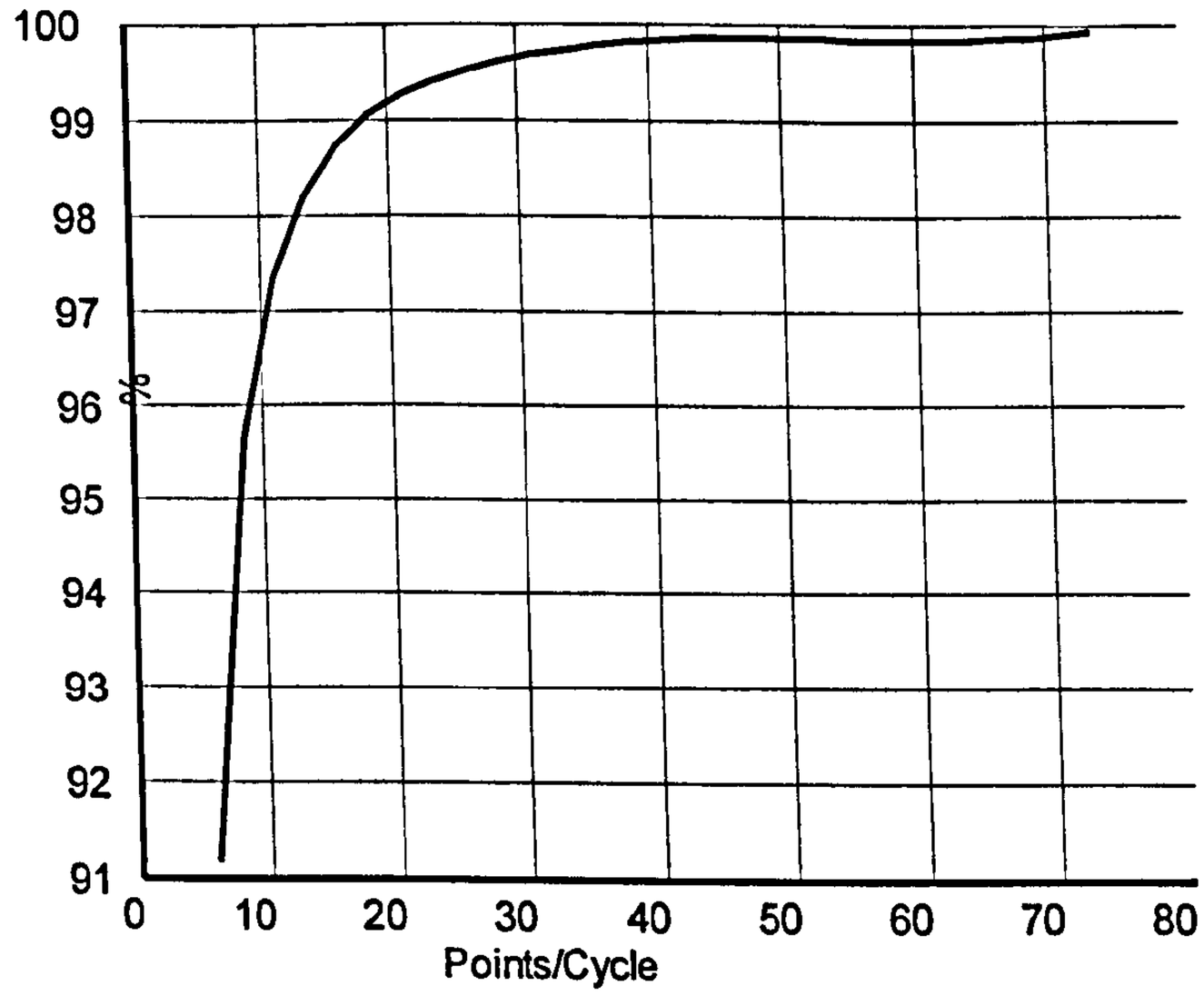


Figure 3-4. The Relationship between the Number of Points per Cycle and the Magnitude of the Fundamental Frequency of a Sinewave with Fixed Time Steps.

the waveform over a number of cycles and is shown in Figure 3-5. This effect is not revealed by Fourier analysis but only becomes significant when the number of points per cycle is small. It is caused by differences between the number and position of points within each cycle.

The discussion so far suggests that reasonable results could be obtained by using PWL waveforms in a simulator if the time steps were chosen to be about 25 times less than the period of the highest frequency sinusoidal signal. Unfortunately, it is often impossible to predict the component frequencies of all signals generated during a simulation. The time steps would therefore have to be made small enough to represent the highest frequency components that might be generated. This is likely to produce waveforms that have an unnecessarily large number of points and hence an inefficient simulator.

An efficient representation of a continuously changing signal must only place points where they are necessary to maintain a particular level of accuracy. This implies that the PWL waveform should have time steps of varying length (as was the case for the digital signal represented in Figure 3-1). A simulator that generates PWL waveforms must therefore provide a mechanism to determine when to insert a new point into a waveform.

A digital PWL waveform usually only has points at two finite levels (e.g. 0V and 5V) that correspond to the logic 0 and logic 1 states. If the number of levels is increased, the

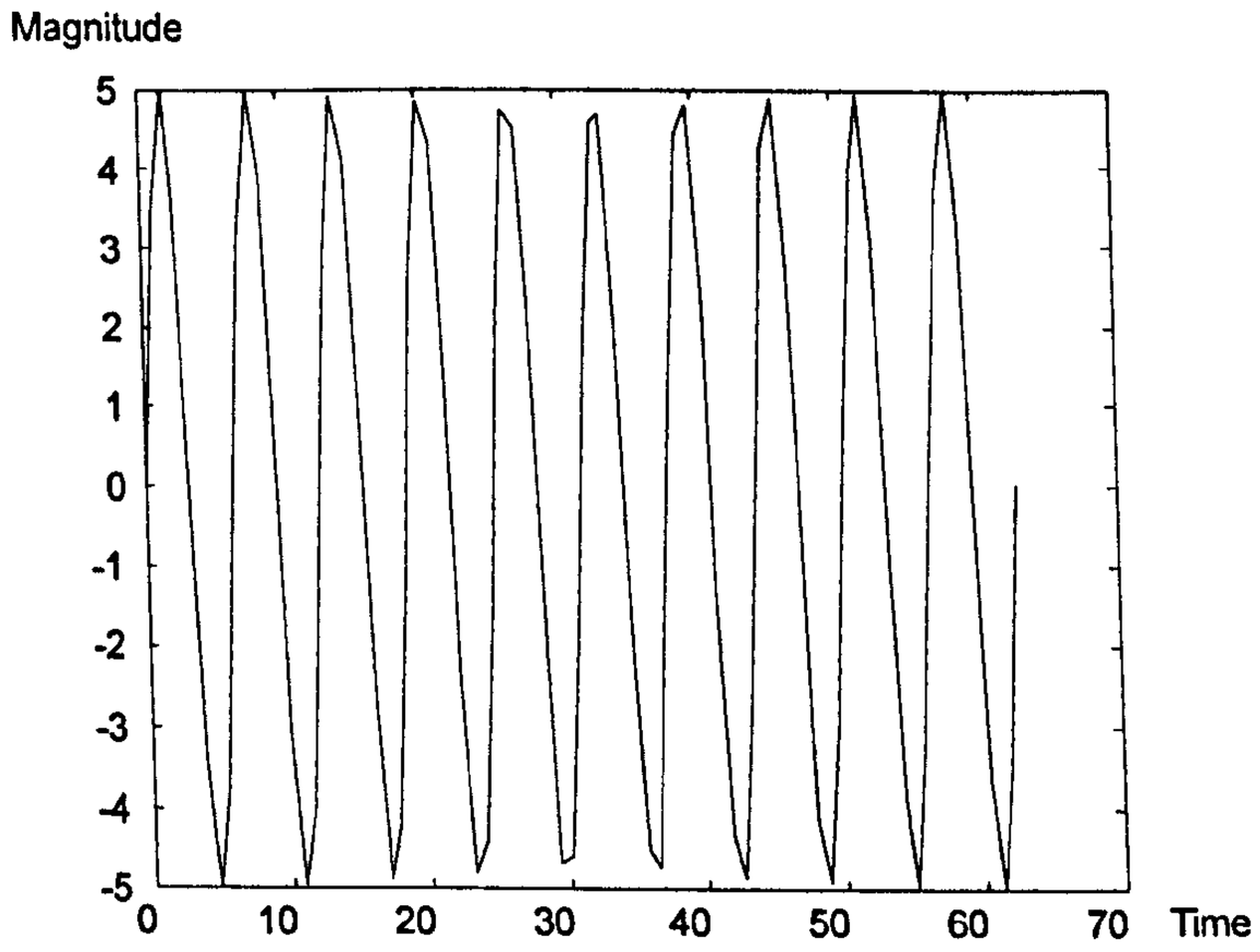


Figure 3-5. PWL Sinewave with 8.1 Points per Cycle.

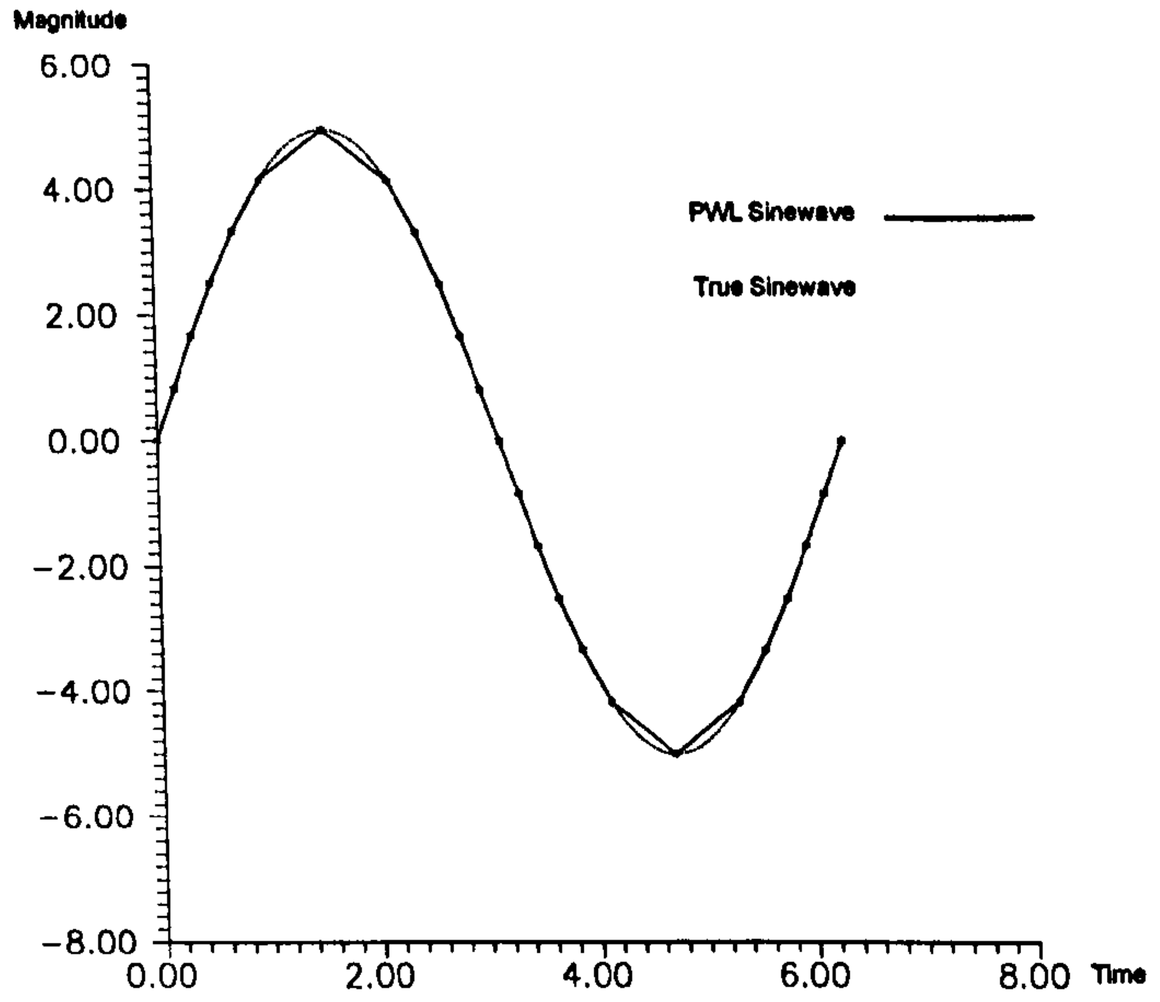


Figure 3-6. PWL Sinewave Using Fixed Magnitude Steps.

concept of placing all of the points at finite levels can be applied to any waveform. If the difference between levels is small, the PWL representation of a signal will be more accurate but require more points. A PWL representation of a sinewave using 14 discrete levels is shown in Figure 3-6. If this waveform is compared to that in Figure 3-3 (which has the same number of points) it can be seen that the representation using fixed magnitude levels is more accurate where the sinewave is changing rapidly but introduces more distortion at the upper and lower extremes. The errors are similar to the quantisation errors generated during analogue to digital conversion as a result of using a finite number of bits. If the dynamic range of signals is large, a large number of levels will be required to represent both large and small signals without loss of accuracy. Signal levels in an ASIC could vary from a few microvolts to tens of volts. If the magnitude steps were selected to be about 1μV for low-level accuracy, the waveform in Figure 3-6 would contain several million points, clearly undesirable. A non-linear range of magnitude steps (e.g. logarithmic) could improve the efficiency but might mask important low level signals superimposed on large signals, e.g. DC-biased amplifiers and systems using amplitude modulation (AM).

A method of allocating the points of the PWL waveform to provide the optimum trade-off between efficiency and accuracy without making assumptions about the nature of the signals requires variable steps to be used for both time and magnitude. The method developed in this thesis places the points in such a way as to maintain a particular level of

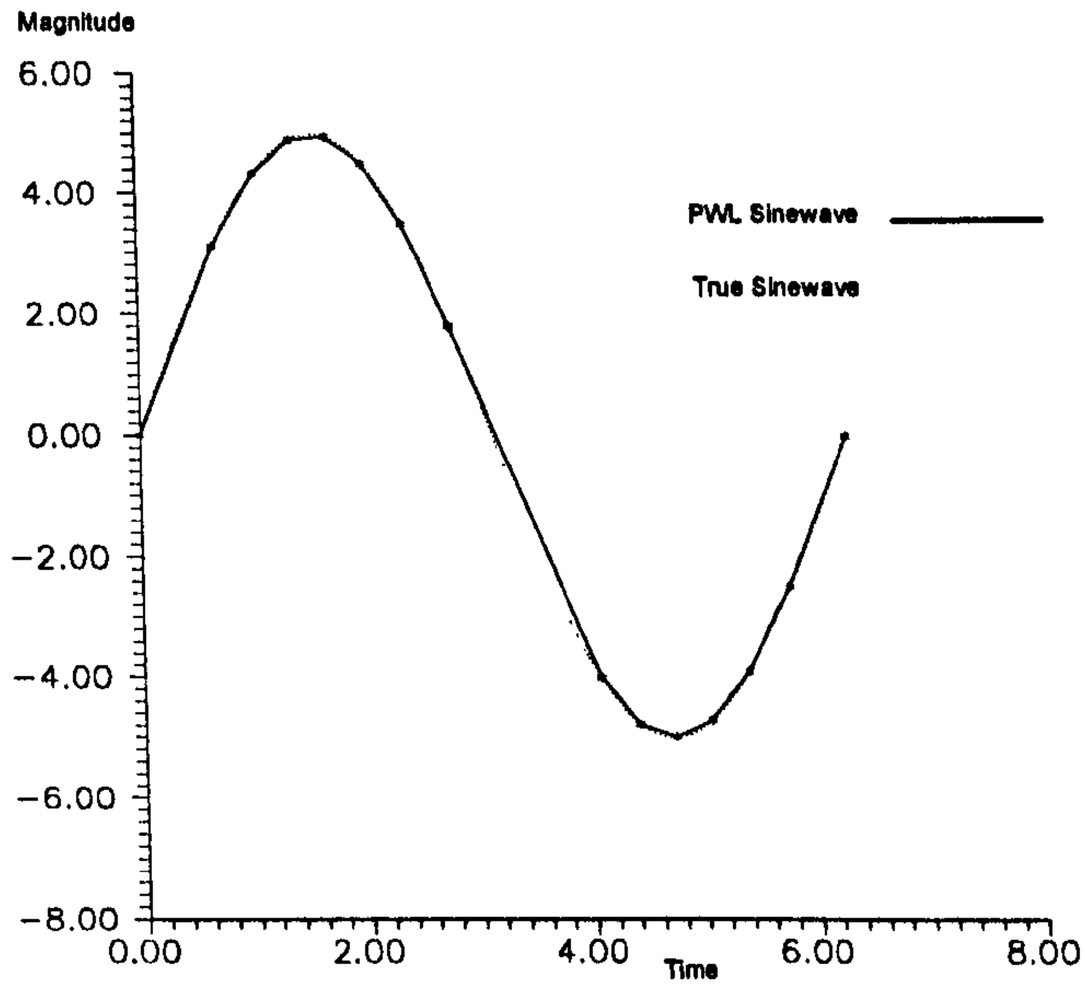


Figure 3-7. PWL Sinewave Using Variable Time and Magnitude Steps.

accuracy in the PWL representation. This method creates a new PWL segment whenever the **gradient** of the waveform changes significantly. The number of points is therefore proportional to the rate of change of the signal, i.e. the number of points is proportional to d^2V/dt^2 or d^2I/dt^2 as appropriate. This contrasts with the fixed magnitude step method where the number of points are proportional to dV/dt or dI/dt. This new method places more points at the extremes of a sinewave where the waveform is least linear and less in the centre regions. A PWL representation of a sinewave using variable time and magnitude steps is shown in Figure 3-7. This waveform contains the same number of points as those in Figure 3-3 and Figure 3-6 but provides a closer match to the true sinewave.

The distortion introduced when variable time and magnitude steps are used was analysed using the same 1024-point FFT as the constant time step waveforms. The relationship between the number of points used and the magnitude of the fundamental frequency is plotted in Figure 3-8. Comparison with Figure 3-4 shows that using variable time and magnitude steps produces a PWL representation that is comparable with, or more accurate than that produced using fixed time steps for any number of points per cycle. However, the representation using variable time and magnitude steps can maintain this accuracy across all signal frequencies, unlike the fixed time step representation.

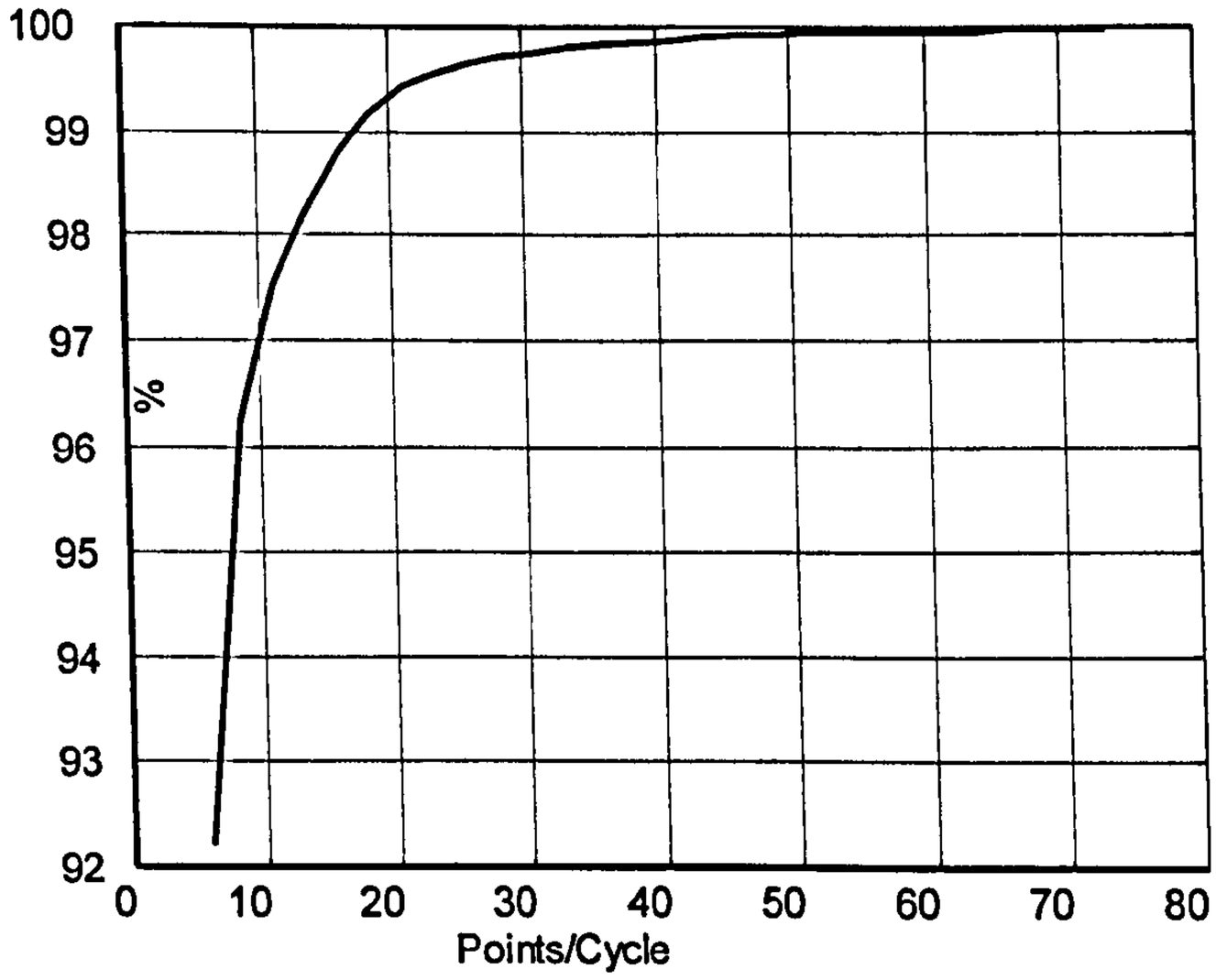


Figure 3-8. Relationship between Number of Points and Magnitude of Fundamental Frequency for a PWL Sinewave with Variable Time and Magnitude Steps.

3.2.3 Implementation of an Optimised PWL Representation.

Having determined that placing the PWL points where the signal gradient changes significantly using variable time and magnitude steps produces a representation that is both accurate and efficient, a suitable algorithm, capable of processing any signal must be developed. The principal issue is the criterion to use in determining when a new PWL segment becomes necessary.

The representation of signals that form the input to a simulation is critical: any distortion in these signals cannot be corrected and may cause unacceptable inaccuracies in the results. However, since these signals will be processed by simulation models, any unnecessary points will degrade the simulation efficiency. A method of generating optimised PWL input waveforms is therefore required. A two-phase approach was used:

1. A PWL waveform was generated using fixed time steps that were small enough to provide the required accuracy (e.g. using 1000 points per cycle for a sinewave).

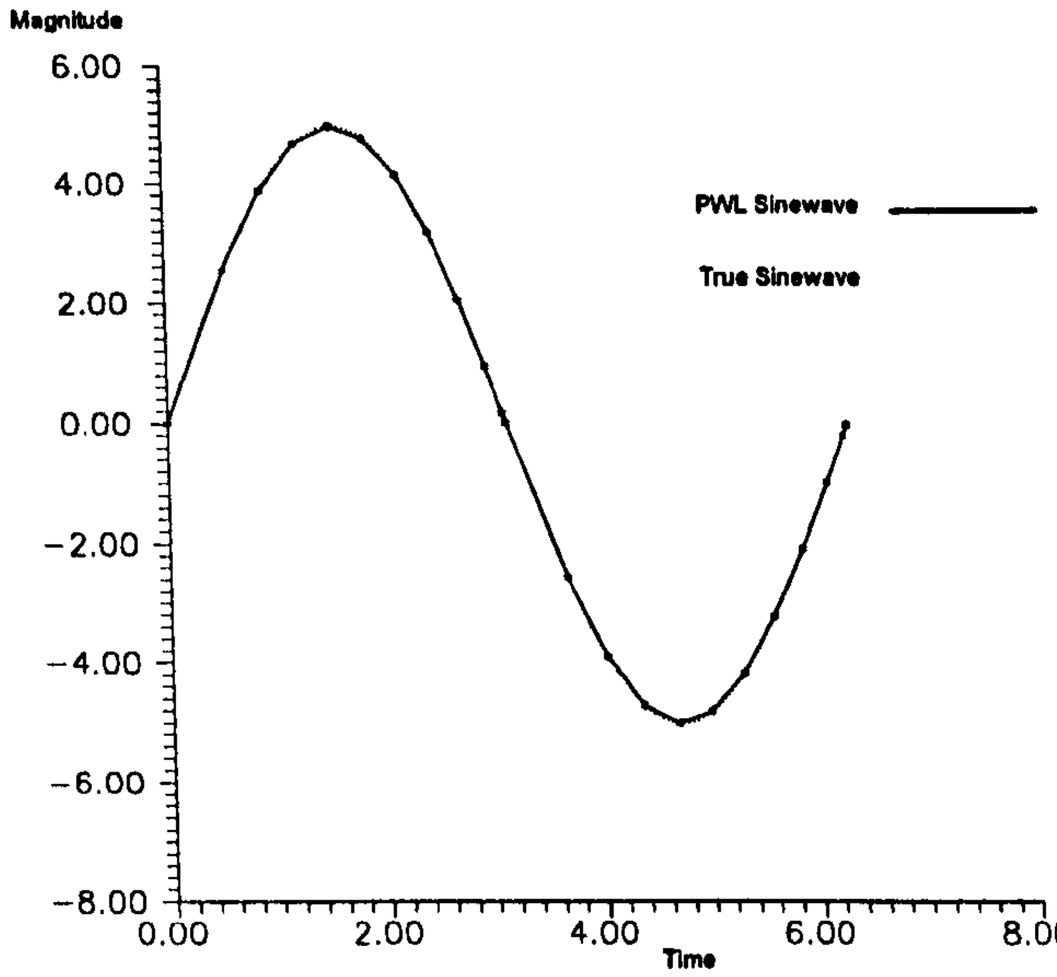


Figure 3-9. PWL Sinewave Using Relative Error Criterion.

2. The contribution of each point was assessed. If the removal of a point produced a magnitude error that was less than that allowed, the point was deemed to be redundant and removed.

The maximum allowable error was specified as a percentage of the peak magnitude. This produced acceptable results for sinusoidal signals. If a signal has a large dynamic range and its low level behaviour requires accurate definition, a second error criterion based on the error relative to the instantaneous magnitude could be used. Absolute and relative error criteria are often used in analogue simulators to overcome problems of large dynamic ranges. The effect of using a relative error criterion on a sinewave is shown in Figure 3-9. This waveform is a closer representation of a sinewave than Figure 3-7 (both were specified to have a maximum error of 5%) but has redundant points in the zero magnitude region. The mechanism for generating input signals supports both error criteria so that the most appropriate method can be chosen according to the nature of the signals and the simulation models used. The efficiency of each method is illustrated by the number of points generated for a single sinewave given in Table 3-1.

The PWL waveforms generated by simulation models should also be optimised. This is especially important if the characteristics of the output signals are very different from the input (the output waveform from a linear amplifier for example could probably have the

Maximum Allowable Error %	No. Points/Cycle using Absolute Error	No. Points/Cycle using Relative Error
0.2	77	101
0.1	50	65
1	35	47
2	24	35
5	14	23
10	10	17
15	8	15
20	7	13

Table 3-1. Points Required to Represent a Sinewave for Different Error Criteria.

same distribution of points as the input waveform, this is not the case for a comparator circuit). The optimisation method used in step 2) of the input waveform processing is also suitable for these output waveforms.

There are several possible algorithms that could be used to optimise the waveform to meet the given error criterion. Three possible algorithms are discussed below.

Algorithm 1.

The aim of the optimisation is to reduce the number of PWL segments used and hence the number of points that must be stored. This algorithm attempts to extend a PWL segment until such time as the error condition is exceeded, at which point a new segment is created. A PWL segment is defined by its starting point and its gradient. In an event-driven simulator, the creation of a segment would be classified as an event. Representing the PWL segment as an initial value plus a gradient enables subsequent models to evaluate the signal at any time, without having to wait until the end of the segment. This algorithm is described by the Pseudo code in Figure 3-10. Unfortunately, this algorithm becomes unstable when the allowable error is small: in these situations it produces small

PWL segments that oscillate around the original segments and so doesn't reduce the number of points appreciably.

Algorithm 2.

Algorithm 2 is similar to algorithm 1 except that when the error condition is exceeded, the gradient of the next segment is calculated from the values of the next two points in the input waveform. It produces a consistent reduction in the number of segments and remains stable when a small allowable error is specified. Unfortunately, this algorithm is not well suited to conventional event-driven simulators since the calculation of the gradient requires the signal magnitude at a future time point to be known.

```
WHILE (i_step < max_step)
{
        delta_t = in_time[i_step] - out_time[o_step];
        vpredict = out_data[o_step] + (dvdt * delta_t);
        vdiff = |in_data[i_step] - vpredict|;
        IF (vdiff > verror)
        {
                // start new PWL segment
                o_step++;
                out_data[o_step] = in_data[i_step];
                out_time[o_step] = in_data[i_step];
                // calculate new dvdt
                dvdt = (out_data[o_step] - out_data[o_step-1]) /
                        (out_time[o_step] - out_time[o_step-1]);
        }
        ELSE
        {
                // update current output state
                out_data[o_step] = vpredict;
                out_time[o_step] = in_time[i_step];
        }
        i_step++;
}
```

Figure 3-10. Pseudo Code for Optimisation Algorithm 1.

Algorithm 3.

This algorithm is similar to algorithms 1 and 2 in that it attempts to extend PWL segments using the segment start value and a gradient. Unlike the other algorithms, it stores the value of the input waveform at the previous time step. It uses this value to correct the output waveform when the error condition is exceeded. A Pseudo code description of this algorithm is given in Figure 3-11. Algorithm 3 is always stable and produces the closest match to the input waveform since it uses a backwards error correcting technique. This method was used to generate the PWL waveforms that were analysed in the previous section. Since the value of the current output point is corrected when a new segment is created, the gradient of the current segment is not fixed until this time. Consequently, there is always a delay of one time step before the output results can be processed by subsequent simulation models. This makes it unsuitable for conventional event-driven simulation techniques. However, since it provides the best accuracy and is always stable,

```
WHILE (i_step < max_step)
{
        vprev = in_data[i_step-1];
        delta_t = in_time[i_step] - out_time[o_step];
        vpredict = out_data[o_step] + (dvdt * delta_t);
        vdiff = |in_data[i_step] - vpredict|;
        IF (vdiff > verror)
        {
                // correct last estimated value
                out_data[o_step] = vprev;
                // start new PWL segment
                o_step++;
                out_data[o_step] = in_data[i_step];
                out_time[o_step] = in_data[i_step];
                // calculate new dvdt
                dvdt = (in_data[i_step] - vprev) /
                        (in_time[i_step] - in_time[i_step-1]);
         }
         ELSE
         {
                // update current output state
                out_data[o_step] = vpredict;
                out_time[o_step] = in_time[i_step];
         }
         i_step++;
}
```

Figure 3-11. Pseudo Code for Optimisation Algorithm 3.

it was decided to use this algorithm as the basis for the simulation methodology developed in this thesis. This would require the simulation models to be designed in such a way that a delay of one time step between an output signal changing and it being evaluated by following models would not cause significant inefficiencies.

3.3 Development of Building Blocks for Behavioural Models.

The modelling approach developed in this thesis is based on a small number of building blocks representing commonly occurring digital and analogue operations. Each operation is implemented by a function that is called from a library of generic functions. The generic functions are invoked with parameters to set the desired behaviour for each instance. The identification of the required operations and the creation of the corresponding building blocks are discussed in the following sections.

3.3.1 Digital Building Blocks.

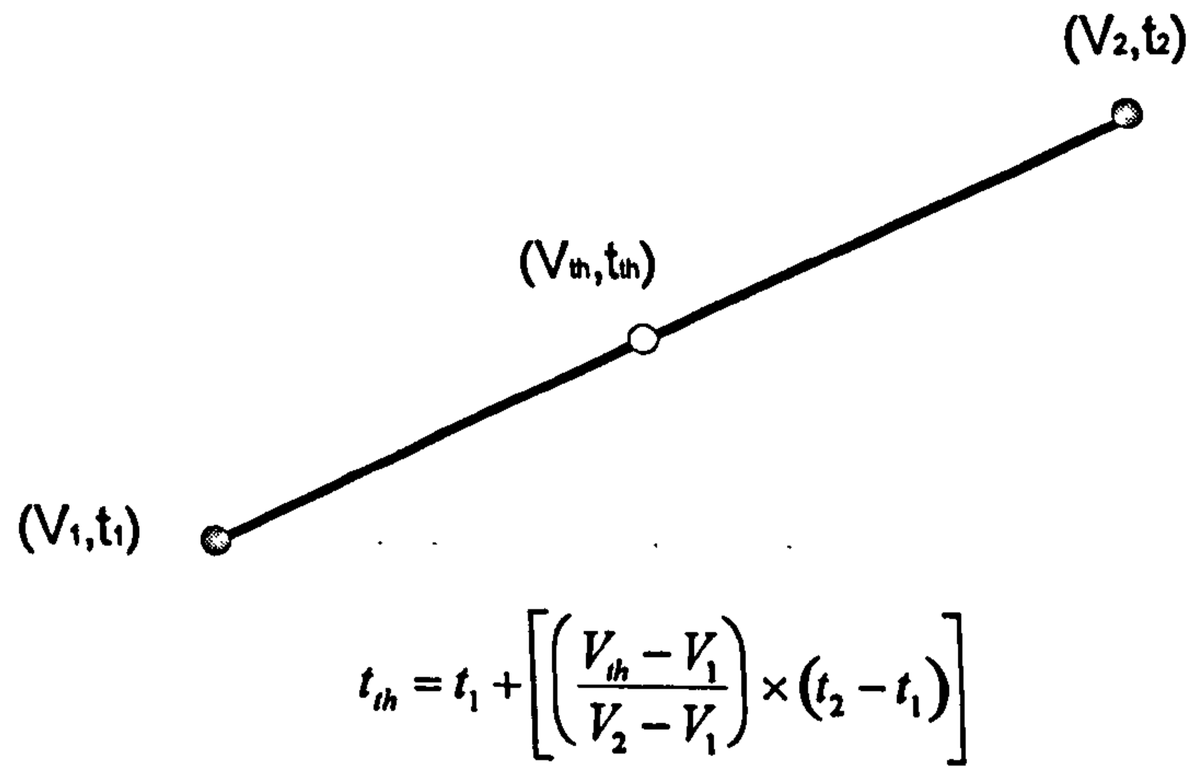
There are three logical operations that can be combined to implement any Boolean expression. These are the NOT, AND and OR functions. They form three of the building blocks used for digital models. A fourth building block that implements the XOR function was also created. Although it is not a primitive operation, the XOR building block simplifies the construction of models for many arithmetic circuits. The NOT function is the simplest since it has one input and one output. The other functions each have two inputs and one output.

If the models are used in a digital-only simulation they must process and produce PWC waveforms. Since PWC waveforms represent logic states the models can perform logical operations on the waveforms directly. This can be done continuously or using an event-driven approach. An event-driven approach is preferable since it involves less processing time. A Pseudo code description of an event-driven NOT operation for PWC waveforms is given in Figure 3-12. An event is a change of logic state and is detected by a function *event()*. The propagation delay of a physical logic gate is modelled by *delta* so that the output signal doesn't change at the same instant as the input changes. This model assumes that only the *logic 0* and *logic 1* states are possible.

```
WHILE (i_step < max_step)
{
        // test for event
        IF (event(in[i_step]))
        {
                o_step++;
                out[o_step].data = ! in[i_step].data;
                out[o_step].time = in[i_step].time + delta;
        }
        // jump to next input step
        i_step++;
}
```

Figure 3-12. Pseudo Code for PWC NOT Function.

The operation of models in an analogue environment is more complex since the signals are represented by PWL waveforms that correspond to continuously varying voltage waveforms rather than logic states. A model must therefore determine the logic states represented by its input waveforms before it can generate the appropriate output. Determination of the logic states is performed by comparing the magnitude of the input signal with one or more threshold values (analogue to digital (A/D) conversion). If the signal crosses a threshold, the time at which this occurs must be found so that an event can be scheduled for the digital model. The A/D conversion for PWL signals involves linear interpolation between two points when a threshold crossing is detected. This is far more efficient than the methods used in conventional mixed-signal simulators. The



$$t_{th} = t_1 + \left[\left(\frac{V_{th} - V_1}{V_2 - V_1}\right) \times (t_2 - t_1)\right]$$

Figure 3-13. Interpolation of PWL Waveform to Generate Digital Event.

interpolation method is illustrated in Figure 3-13.

The operation of the NOT function for PWL waveforms is similar to that for PWC waveforms except the logic state transition time is obtained from the interpolation. A Pseudo code description of the NOT function is given in Figure 3-14. This model creates two PWL points. The first point marks the end of the PWL segment defining the current logic state. This point is delayed by *delta* that represents the propagation delay through the logic gate after the input threshold voltage is crossed. The second point defines the PWL segment representing the transition to the next logic state. This point is delayed by *t_rise* or *t_fall* that model the rise and fall times of the output voltage. The *event()* function detects if the input waveform has crossed the threshold voltage *vth* since the last time step. The model is therefore only evaluated when an event has occurred.

The interpolation routine (inside the model) determines the exact time of the event. This approach is different from conventional event driven simulation where the event times (and hence the times when models are evaluated) are controlled by a global event schedule mechanism.

If *t_rise* and *t_fall* are zero, the model in Figure 3-14 will generate a PWC waveform with redundant points as shown in Figure 3-2. The interpolation routine will generate the correct event time with this type of waveform since the expression in Figure 3-13 reduces to *t1* when *t1* and *t2* are equal (i.e. when there is an instantaneous change of state). This model therefore generates the same results as the model in Figure 3-12 when both are driven by a PWC waveform with redundant points (except the PWL model retains the redundant points in the output waveform whereas the PWC model removes them). This consistency between different forms of the same function is essential in any mixed-signal simulator.

```
WHILE (i_step < max_step)
{
        // test for event
        IF (event(in[i_step]))
        {
                t_th = interpolate(in[i_step],in[i_step-1]);
                // mark end of current_state segment
                o_step++;
                out[o_step].data = out[o_step-1].data;
                out[o_step].time = t_th + delta;
                // create transition_to_next_state segment
                o_step++;
                IF in[i_step].data > vth
                {
                        out[o_step].data = Vlow;
                        out[o_step].time = t_th + t_fall;
                }
                ELSE
                {
                        out[o_step].data = Vhigh;
                        out[o_step].time = t_th + t_rise;
                }
        }
        // jump to next input step
        i_step++;
}
```

Figure 3-14. Pseudo Code for PWL NOT Function.

If the rise and fall times of the output voltage waveform cannot be predicted before the simulation is run (or if they change during the course of the simulation due to reconfiguration of the output circuit) a different form of model is required. The model in Figure 3-14 can easily be modified to generate a PWC current waveform instead of a PWL voltage waveform by setting *t_rise* and *t_fall* to zero and by assigning values other than *Vlow* and *VHigh* to *out[o_step].data* (e.g. -1mA and +1mA). The PWC current waveform would then provide the input to the following RC stages (see later sections) that would generate the correct PWL voltage waveforms.

The model in Figure 3-14 assumes that only a single voltage threshold is used and that only two output states are possible. The threshold voltage used by the interpolation

routine could be set according to the current state of the model. This would produce two threshold values and so could model the input hysteresis exhibited by some circuits.

PWC models of the AND, OR and XOR functions are constructed using the same techniques as for the PWC NOT model. Since these models generate an output from a pair of input waveforms the *event()* function must be sensitive to two independently changing PWC input signals. Models with more than two inputs are constructed from cascaded 2-input models.

The AND, OR and XOR models that process PWL waveforms are more complex than the PWC models and the PWL NOT model since they must be able to determine the states of two independently varying PWL waveforms that are not defined using common time steps. Two such waveforms are shown in Figure 3-15. Waveform A has points at t_0, t_1, t_2, t_3, t_4 and t_6 whilst waveform B has points at t_0, t_1, t_5 and t_6. The *event()* function will detect that event E1 has occurred when time step t_2 is reached. However, the state of waveform B cannot be determined until time step t_5 is reached. This also applies to event E3 that will be detected when time step t_4 is reached. These events must be stored in an 'Event List' until such time as they can be processed. Event E2 will be detected when time point t_5 is reached. This must be inserted into the event list before event E3 so that

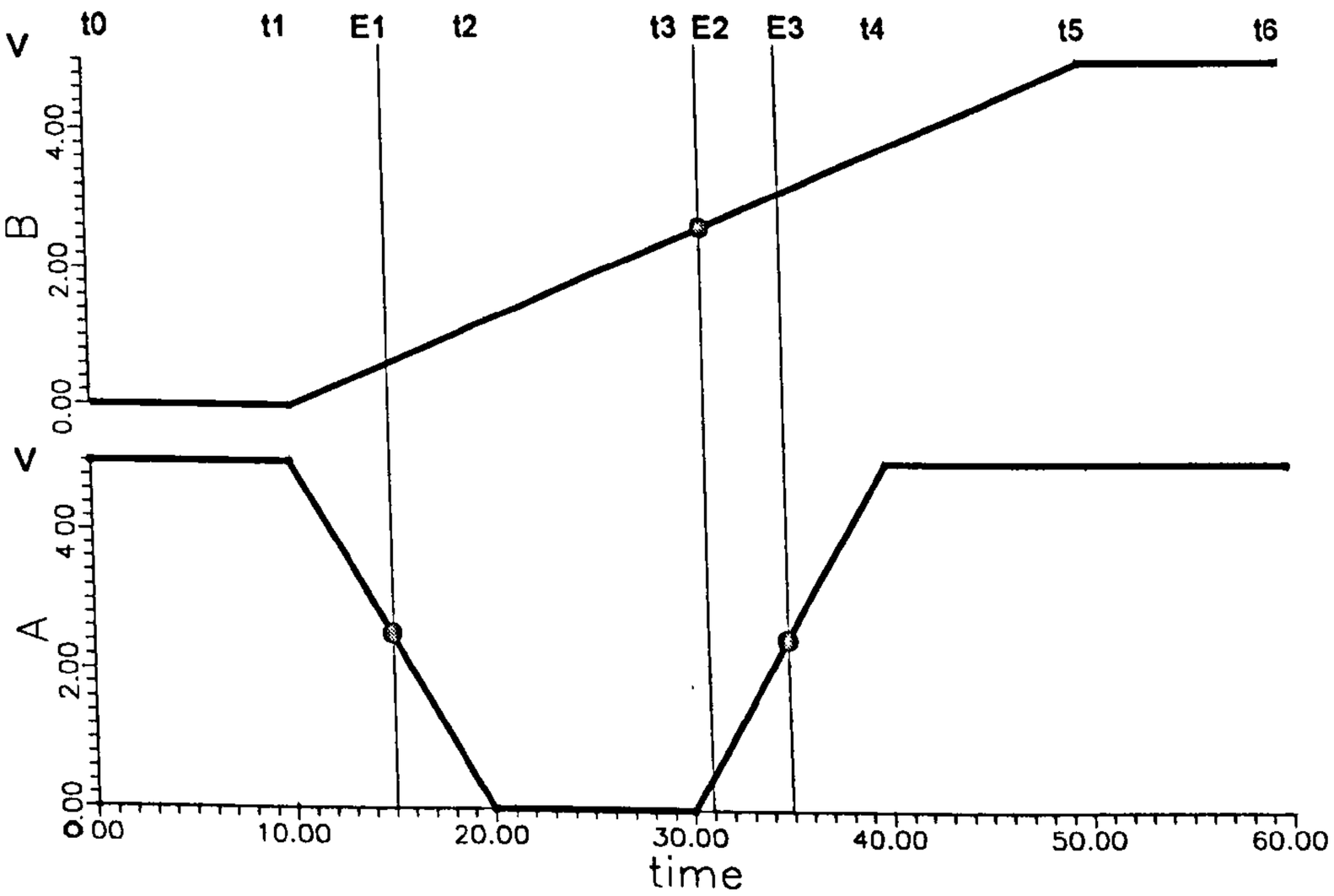


Figure 3-15. Independently Changing Digital PWL Signals.

the events are processed in the correct sequence. Since the interpolation function (that generates the events) is part of the model, each model is required to maintain its own event list: a queue of pending events. The algorithm for setting up the event list is shown in Figure 3-16 where *a* and *b* refer to the two input waveforms. Finding the next *a_event* or *b_event* involves stepping through the input waveform points until a threshold crossing is detected (using the same *event()* function as before). The time of the event is then determined by the interpolation routine.

Events E2 and E3 in Figure 3-15 occur in close proximity. If different propagation delays are associated with the A and B inputs, it is possible that event E2 could generate an output state that is over-ridden by that generated by event E3 and so would not be detected at the output of a physical circuit. Any glitches or invalid outputs should therefore be detected and removed before the output waveform is passed to other model

```
Find next a_event;
Find next b_event;
IF (b_event before a_event)
{
        WHILE (b_event before a_event)
        {
                Write b_event to event_list;
                Find next b_event;
        }
        Write a_event to event_list;
        WHILE (b_event before a[step].time)
        {
                Write b_event to event_list;
                Find next b_event;
        }
        Set event_pointer to 1st b_event in event_list;
}
ELSE
{
        WHILE (a_event before b_event)
        etc...
        .
}
```

Figure 3-16. Algorithm to Set Up Event Queue.

inputs. The simulation methodology developed in this thesis allows models to free run for a fixed simulation period, subject to the availability of valid outputs from preceding models. The correction of output waveforms is therefore performed by each simulation model before control is passed to subsequent models. The control and co-ordination of this process and the implications for feedback systems are discussed in Chapter 4.

Sample simulation results of the XOR PWL model are shown in Figure 3-17. This shows the XOR model generating correct results for a variety of input signal transitions including fast rise and fall times, slow rise and fall times, short-lived glitches and coincident events on both waveforms. The input signals deliberately contain unnecessary points that are successfully filtered-out by the model. Three delay parameters are used in this particular model to represent the propagation delay for a logic 0 to logic 1 transition (2.4ns); the propagation delay for a logic 1 to logic 0 transition (1.6ns); and the rise and fall time (2.0ns). Since the logic 1 to logic 0 transition is faster than the logic 0 to logic 1 transition, the near-incident input transitions at approximately 70ns cause an invalid output (the 0→1 output transition would be scheduled to occur after the 1→0 transition producing a negative width glitch). This error condition is detected by the model and

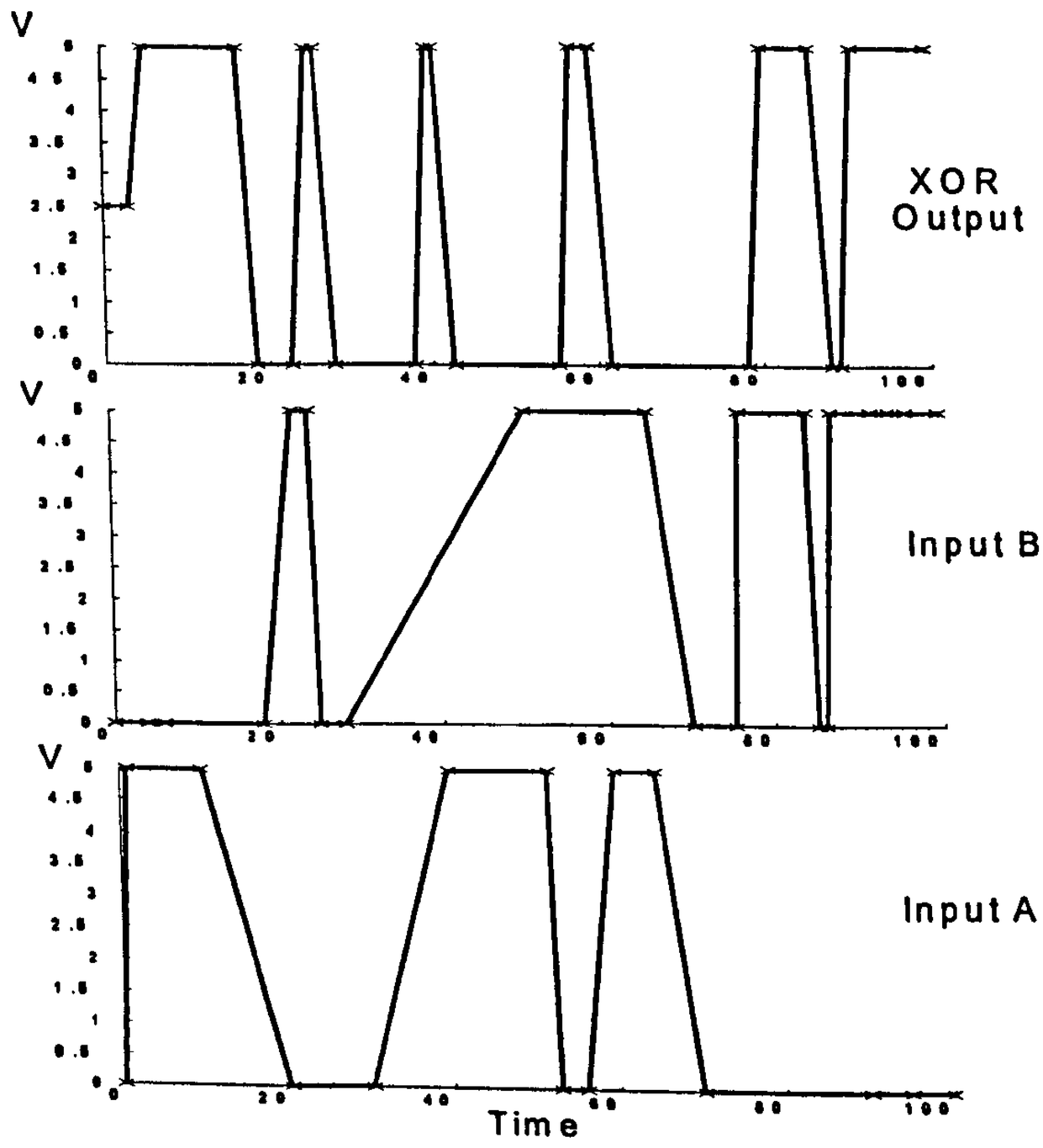


Figure 3-17. Simulation Results for PWL XOR Model.

automatically corrected before control of the output waveform is relinquished. The initial value of the output waveform is 2.5V. This corresponds to the X-state of conventional digital simulators. Like conventional simulators, this value is not set to a value corresponding to a logic state until the first event occurs.

3.3.2 Analogue Building Blocks.

It is possible to simulate any analogue function in the time domain using a combination of four basic functions: addition, multiplication, integration and differentiation. These operations can be implemented very efficiently for PWL waveforms. Building blocks were created that implement each of these functions. They were designed such that they can be easily combined to construct models of physical components.

3.3.2.1 Adder Model.

Two forms of adder are required. The first adds a scalar quantity to a PWL waveform. This adds the scalar quantity to the magnitude of every PWL point. It is a very simple function since each point in the PWL waveform can be processed independently: the relative positions of the points and the gradient of each segment are not affected.

The second form adds two PWL waveforms. This is more complex than the first form since direct addition of two PWL waveforms is not possible unless all of the points occur at the same times in both waveforms. Interpolation of both waveforms is required to align the points at common times, the magnitudes can then be added together. The positioning of the points in each waveform will have been optimised when the waveforms were generated to provide a given level of accuracy. The interpolation process is therefore only required to determine the magnitude of each waveform at the time steps corresponding to points in the other waveform. The resultant waveform is likely to contain more points than either of the input waveforms: if signal A contained n_a points and signal B contained n_b points, signal (A + B) could contain up to $(n_a + n_b)$ points. It is therefore beneficial to filter the output waveform to remove unnecessary points. The adder model is likely to be used with a wide range of signal magnitudes. This suggests that a relative error criterion is most suitable for the output filter. Simulation results for the adder model are given in

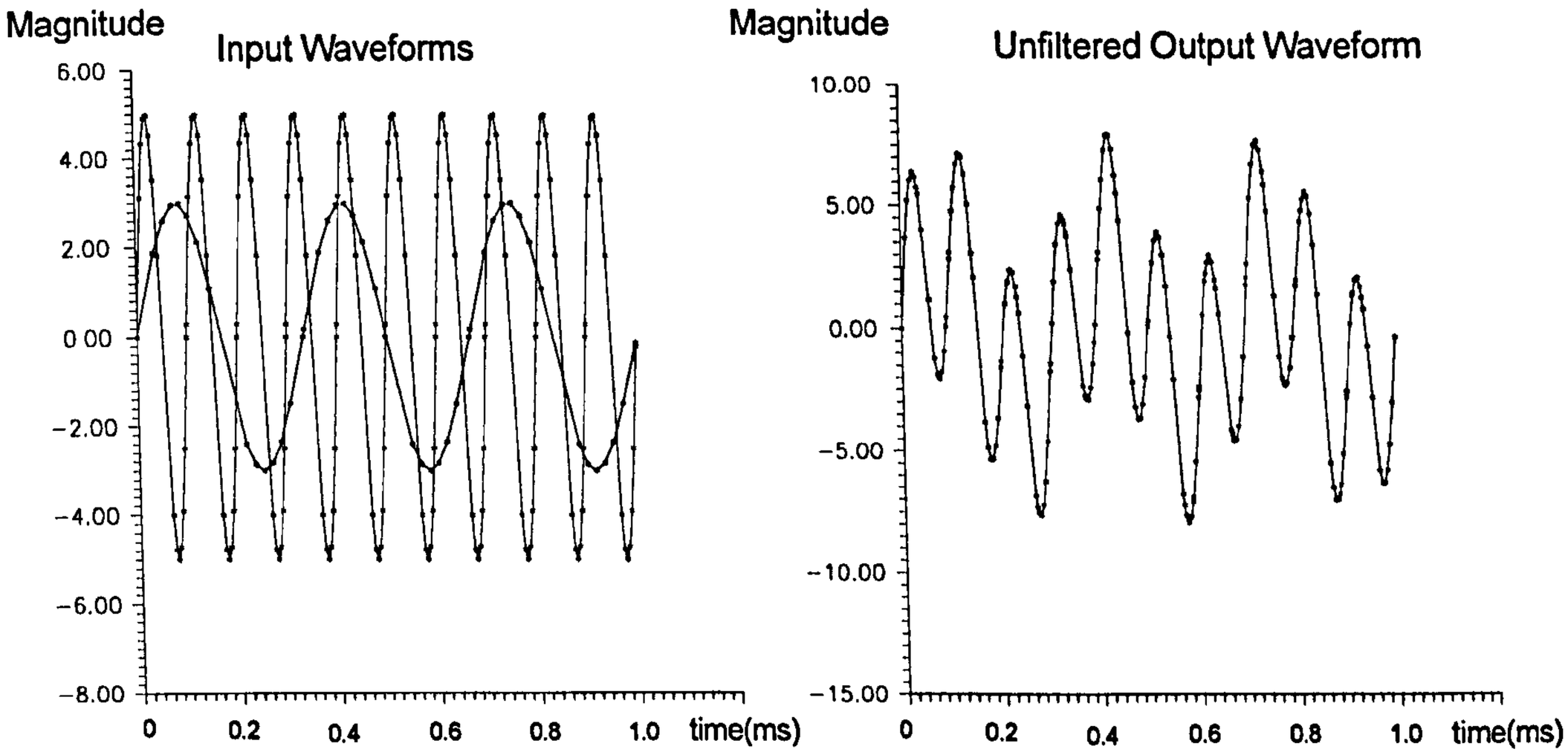


Figure 3-18. Simulation Results for PWL Adder Model.

Figure 3-18. These show the increase in points in the output waveform. The input waveforms were generated using a 5% absolute error criterion. Applying a 5% relative error filter to this output waveform removes 25% of the points without introducing any significant deviations.

3.3.2.2 Multiplier Model.

The structure and operation of the multiplier model is very similar to the adder model. It also exists in two forms: one for multiplying a PWL waveform by a scalar quantity (amplification) and the other for multiplying two PWL waveforms together (modulation). As with the scalar adder, scalar multiplication only requires alteration of the magnitude of the waveform so each point can be processed independently. No new points are generated and the points remain at the same time steps. However, the relative positions of the points will change as the gradient of each PWL segment is effectively multiplied by the scalar quantity.

The second form of the multiplier has to interpolate both PWL input waveforms to generate a set of points at common time steps before the multiplication can be performed. The interpolation is performed using the same method as the adder model. The number of points in the output waveform can therefore also be as large as the total number of points

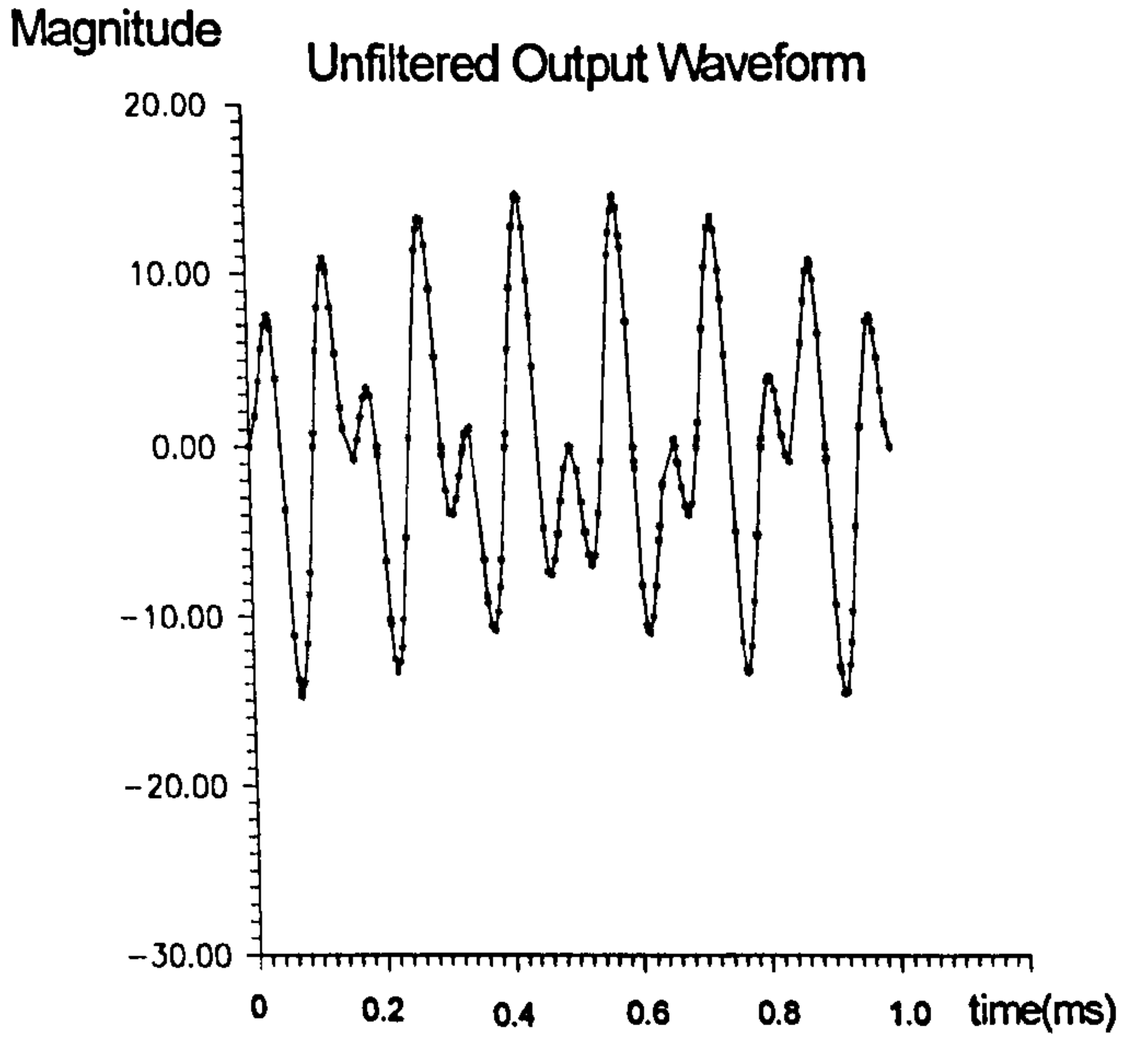


Figure 3-19. Simulation Results for Multiplier Model.

in both input waveforms. Consequently, filtering of redundant points from the output waveform is also performed. Simulation results for the multiplier model using the same input waveforms as Figure 3-18 are shown in Figure 3-19. Applying a 5% relative error filter to this waveform reduces the number of points by 19%. The reduction in the number of points by the filter is smaller than that produced for the adder model. This is expected as the multiplication process generates an output waveform with frequency components not present in the input waveforms (the waveform therefore requires more points than the adder waveform for the same degree of accuracy).

3.3.2.3 Integrator Model.

The integrator model is based on the behaviour of the ideal passive RC integrator network shown in Figure 3-20. It generates a PWL output waveform from a single PWL input waveform. The value of the RC time constant is chosen to implement the required behaviour. It is used to build models of components that have frequency dependent characteristics.

The current that flows into the input terminals is given by:

$$i(t) = C\frac{dV_{out}}{dt} = \frac{V_{in} - V_{out}}{R} \qquad \textbf{Equation 3-1}$$

This has a solution of the form:

$$V_{out} = V_{in} + Ae^{\frac{-t}{RC}} \qquad \text{where } A \text{ is determined by the initial conditions.}$$

If $V_{out} = 0$ at time $t = 0$, $A = -V_{in}$. V_{out} is then given by:

$$V_{out} = V_{in}\left(1 - e^{\frac{-t}{RC}}\right) \qquad \textbf{Equation 3-2}$$

When t is much greater than RC, the effect of the $e^{-t/RC}$ term becomes negligible and so $Vout = Vin$.

If V_{in} is continuously varying, the previous values of V_{in} must be considered as well as the present value. The solution for V_{out} is then given by:

$$V_{out}(t) = \frac{1}{RC}\int_{-\infty}^{\tau} V_{in}(t)\cdot e^{\frac{-(t-\tau)}{RC}}\cdot d\tau \qquad \textbf{Equation 3-3}$$

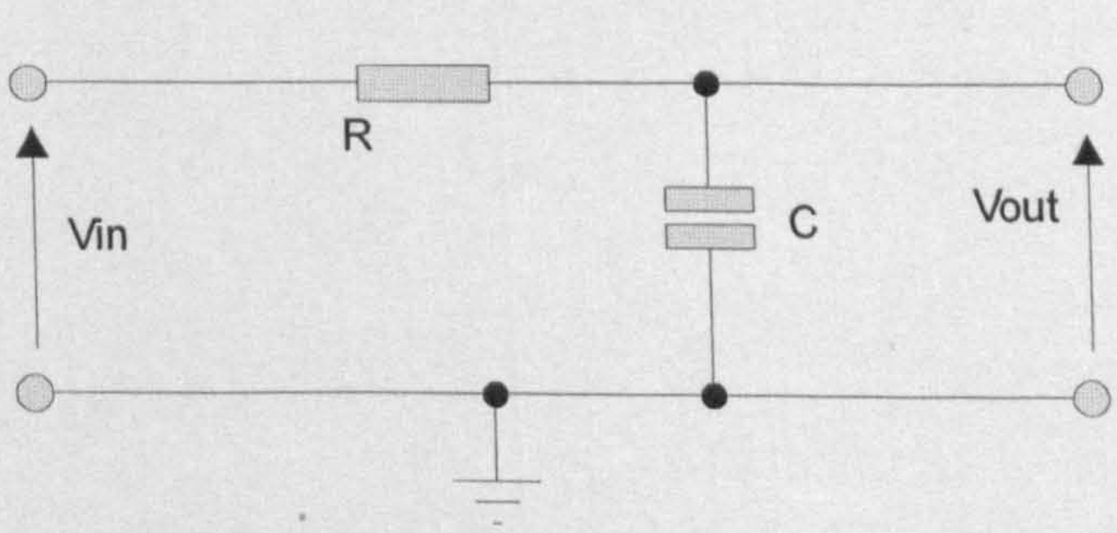


Figure 3-20. Ideal RC Integrator Model.

where τ is the current time step. The network is therefore acting as an integrator with the output voltage corresponding to the integral of the input voltage. Conventional analogue simulators solve this equation using numerical techniques. A more efficient method is required for this simulator.

An expression for the rate of change of V_{out} can be obtained from Equation 3-1:

$$\frac{dV_{out}}{dt} = \frac{1}{RC}\left(V_{in} - V_{out}\right) \qquad \textbf{Equation 3-4}$$

If the time between waveform points is much smaller than *RC*, the magnitude of V_{out} will not change significantly from one time step to the next. Equation 3-4 can therefore be approximated by:

$$\frac{dV_{out}(t)}{dt} = \frac{1}{RC}\left(V_{in}(t) - V_{out}(t-1)\right) \qquad \textbf{Equation 3-5}$$

The value of V_{out} can then be derived:

$$V_{out}(t) = V_{out}(t-1) + \left[\frac{dV_{out}(t)}{dt} \times \left(t - t_{(t-1)}\right)\right] \qquad \textbf{Equation 3-6}$$

This is equivalent to assuming that V_{in} behaves in a piece-wise constant manner. A step change occurs at each new time point creating a potential difference between V_{in} and V_{out}. The magnitude of this potential difference determines the rate at which V_{out} attempts to change to V_{in}. The rate of change predicted by this method will be over optimistic because the change in V_{in} is assumed to occur instantaneously at the start of each time step rather than gradually throughout the time step. If the length of the time step is much less than the *RC* time constant (as assumed above) this further assumption will not cause significant errors. Equation 3-5 and Equation 3-6 form the basis of the integrator model.

Since the rate of change of the output signal is derived from the value at the previous time step, it was discovered that overshoot errors could be generated at time steps where the value of dV_{out}/dt changes polarity. This condition is simple to detect. The integrator model therefore monitors the value of dV_{out}/dt and limits the value of V_{out} to the present value of

V_{in} (its theoretical maximum) at time steps where a change of polarity is detected. This was found to eliminate the overshoot errors.

Equation 3-5 and Equation 3-6 assume that the time between every time point in the output waveform is much less than *RC*. If this is not so the model becomes invalid and errors are produced. The integrator model detects this condition and automatically inserts extra points at a suitable time intervals into the output waveform. Placing the extra points at time intervals of *RC/5* was found to maintain a suitable level of accuracy. If an input signal remains constant, this correction method could insert a large number of unnecessary points as the output converges to a steady state. The number of points added to such waveforms is kept to a reasonable level by monitoring the difference between V_{in} and V_{out}. Once V_{out} has become sufficiently close to V_{in}, it is set equal to V_{in} and no further points are inserted.

The integrator model can dramatically increase the number of points in the output waveform for slowly varying signals. Filtering of redundant output points is therefore performed by this model. Sample simulation results for a 10kHz sinewave input signal for a variety of RC time constants are shown in Figure 3-21. The output results have been filtered using a 5% relative error criterion. This reduced the number of points in the output waveform by 92% when the time constant was ten times less than the wave period.

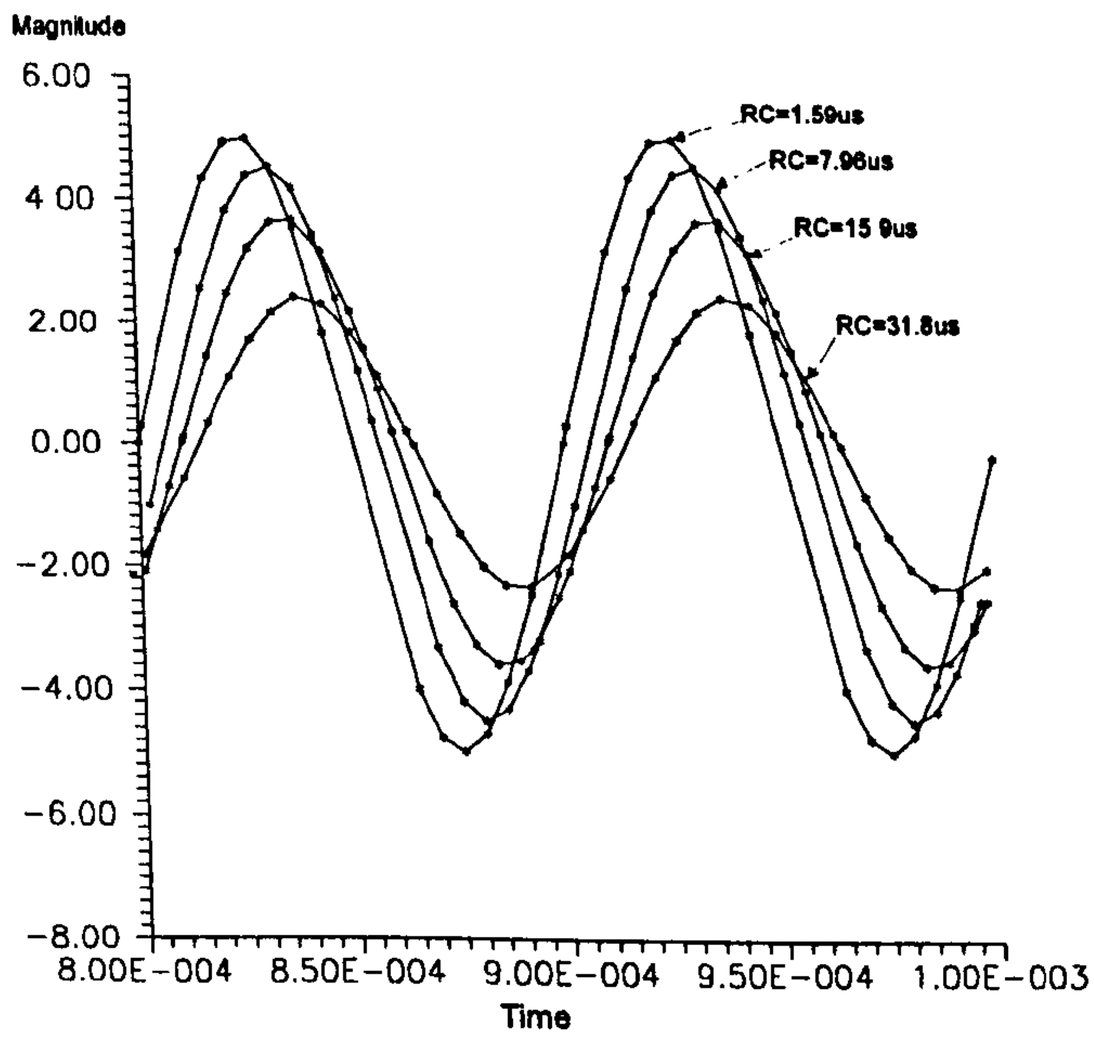


Figure 3-21. Simulation Results for Integrator.

3.3.2.4 Differentiator Model.

The model of the differentiator is based on an ideal passive RC network in a similar way to the integrator. It also generates a PWL output waveform from a single PWL input signal. The RC network is shown in Figure 3-22.

The input current is given by:

$$I = C\frac{d(V_{in} - V_{out})}{dt} = \frac{V_{out}}{R} \qquad \textbf{Equation 3-7}$$

If I is eliminated, Equation 3-7 can be re-written as:

$$\frac{dV_{in}(t)}{dt} - \frac{dV_{out}(t)}{dt} = \frac{V_{out}(t)}{RC} \qquad \textbf{Equation 3-8}$$

If the time steps in the output waveform are much less than RC, Equation 3-8 can be approximated by:

$$\frac{dV_{out}(t)}{dt} = \frac{dV_{in}(t)}{dt} - \frac{V_{out}(t-1)}{RC} \qquad \textbf{Equation 3-9}$$

The output waveform can then be derived:

$$V_{out}(t) = V_{out}(t-1) + \left[\frac{dV_{out}(t)}{dt} \times \left(t - t_{(t-1)}\right)\right] \qquad \textbf{Equation 3-10}$$

This model predicts that the rate of change of V_{out} will follow the rate of change of V_{in} subject to a decay that is proportional to the previous magnitude of V_{out} and RC. When Vin is changing rapidly, the term in RC in Equation 3-9 becomes insignificant and V_{out}

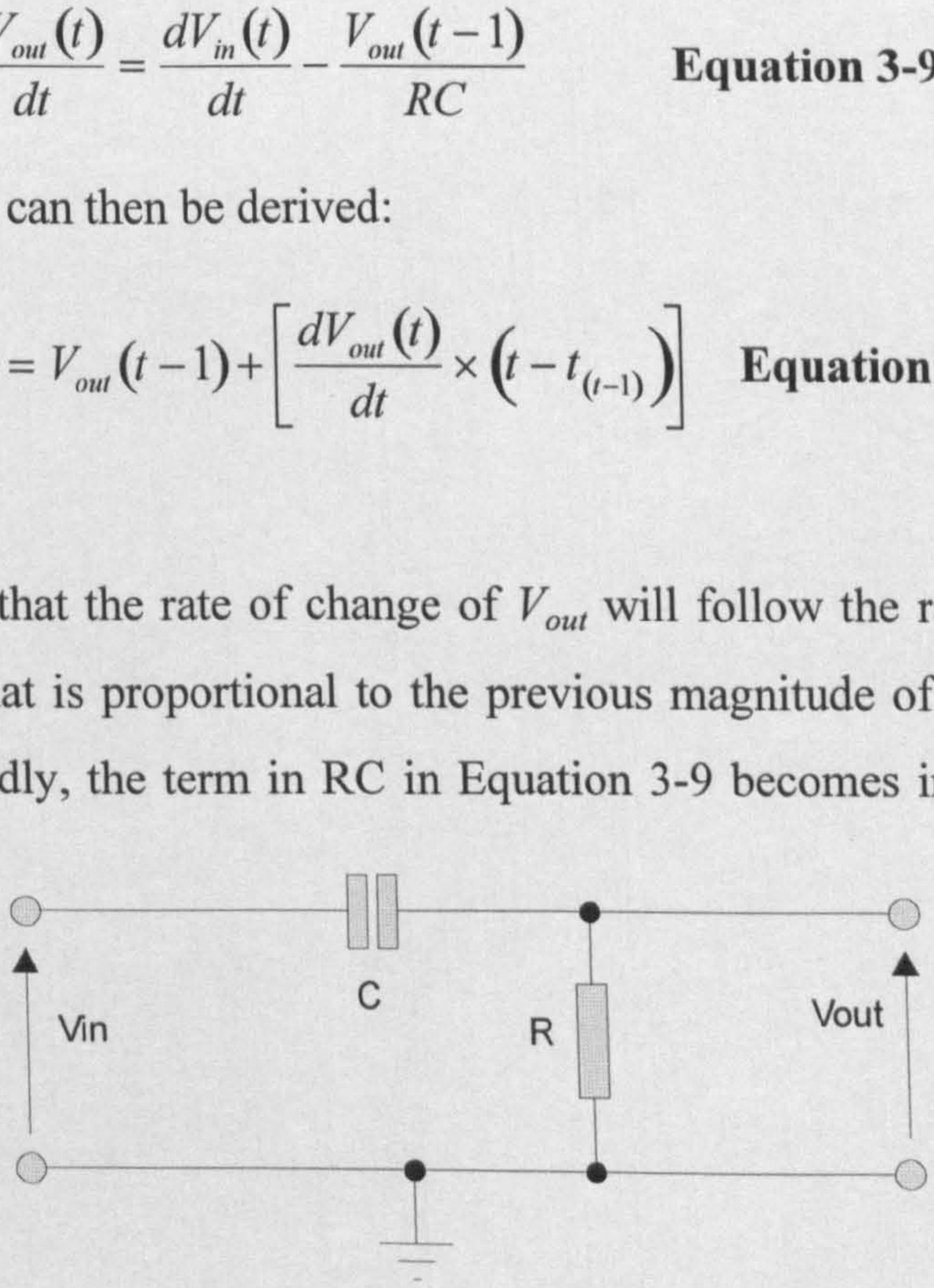


Figure 3-22. Ideal RC Differentiator Model.

will change in the same manner as V_{in}. V_{out} will not necessarily equal V_{in} in these circumstances since Equation 3-10 shows that the value of V_{out} at any time step is dependent on its value at the previous time step. When the rate of change of V_{in} is less than the time constant then V_{out} will decay until it reaches zero.

If the time steps in the input waveform are not much less than *RC* Equation 3-9 becomes invalid. This condition is detected by the model and extra time steps inserted into the output waveform (e.g. at intervals of *RC/10*) until the next point in the input waveform is reached or the magnitude converges to a steady state (zero). As with the integrator model, filtering of redundant points from the output waveform is required.

3.3.3 Mixed Signal Building Blocks.

One additional building block is required to construct mixed signal models. This implements a non-linear function that forces the output waveform to a particular level (e.g. +5 volts) according to the magnitude of its input signals (i.e. it is not a simple gain or addition function). It is used to implement models such as digitally-controlled analogue switches and sample-and-hold devices. This block is also useful to enable non-linear effects such as power-supply signal clamping to be implemented in analogue models.

3.4 Conclusions.

The representation of digital and analogue signals by piece-wise linear and piece-wise constant waveforms has been investigated and shown to provide good accuracy when sufficient points are used. Conventional approaches using fixed time steps are inefficient for simulations where the frequency range of signals is wide or unknown. A method of placing the points using variable time and data intervals controlled by an error monitoring process has been developed. It provides a more efficient representation that doesn't depend on signal frequency. This new method maintains or surpasses the level of accuracy that could be achieved using fixed time steps and the same number of points.

A small set of model building blocks has been developed to work with the new PWL signal representation. These can be combined to construct behavioural models of any analogue, digital or mixed-signal component. The models each contain a mechanism to maintain a local event-queue of pending events. This is intended to improve the efficiency of the simulation that would otherwise be impaired by the need to wait until the end of PWL segments before event-times can be determined.

Certain analogue models can generate output waveforms that include more points than are required to maintain a particular level of accuracy. These models include a filter to remove such redundant points before the waveforms are passed to subsequent models.

Integration and differentiation are frequently used operations in analogue simulation. It has been shown how these operations could be implemented by models based on the behaviour of ideal RC networks provided the time between points is relatively small. The time between points is monitored and new points automatically inserted when required. This approach is much simpler than the iterative numerical methods used in conventional analogue simulators since the variation of a signal between time points is always linear. This should enable the integration and differentiation operations to run more efficiently than conventional techniques.

4. Models and Experiments.

4.1 Introduction.

During the development of the simulation methodology it became apparent that an object-oriented approaach was well-suited to a mixed-signal simulator. Models of physical components that processed PWL signals were created at various levels of abstraction to test the validity of the modelling approach proposed in Chapter 3. These models were evaluated using POISE - an object-oriented simulation environment developed as part of this project and described in Appendix A. Each model's behaviour was compared against the observed behaviour of physical components and that predicted by commercial simulators. Comparisons were also made between the execution times for the PWL models and those of conventional simulators.

The creation of a range of models revealed areas where the original PWL approach was not suitable. New techniques were developed to overcome these limitations.

The development and validation of the object-oriented models and modelling techniques are described in this chapter.

4.2 Class Hieracrchy to Implement the Object Oriented Simulation Methodology.

The C++ language was chosen to implement the object-oriented simulator. C++ is a superset of C, simplifying the task of recoding functions created during the early stages of the project. The analysis and design of the classes was carried out with the aid of the Select CASE tool (based on the methods developed by Rumbaugh [53]).

A true object oriented environment consists entirely of interconnected classes with all classes derived from a single abstract base class. This enables weak type checking to be used by the compiler so providing maximum flexibility for the programmer. The C++ language supports OO design but isn't a pure OO language as it uses strong type checking

BOURNEMOUTH UNIVERSITY LIBRARY 50

and doesn't provide a root object class. The C++ approach removes some of the flexibility from the programmer (e.g. a floating point number cannot be directly assigned to an integer) but provides a high execution speed. The degree to which a program adopts a true object oriented approach can be viewed as a trade-off between flexibility (decisions not made until run-time) and speed (decisions made by the compiler). Since the execution speed of a mixed-signal simulator is a major consideration, it was decided not to adopt a true OO approach. This meant that the class hierarchy had to be sufficiently rich to maintain the advantages of a true OO system whilst giving operating speeds close to those of traditional, structured approaches. This led to the creation of two distinct classes of objects: classes to represent the simulation models and classes to represent the signals that are passed between models. These can be viewed as the equivalents of components and wires in a physical system.

4.2.1 Signal Classes.

The root of the class hierarchy representing signals is the *TSignal* class shown in Figure 4-1. The rectangle representing each class is split into three sections: the first section contains the class name; the second section contains the class attributes (i.e. data items); the third section contains the class operators (i.e. methods or functions). The *TSignal* class represents a single PWL or PWC point. It contains two attributes, *DataP* and *TimeP*,

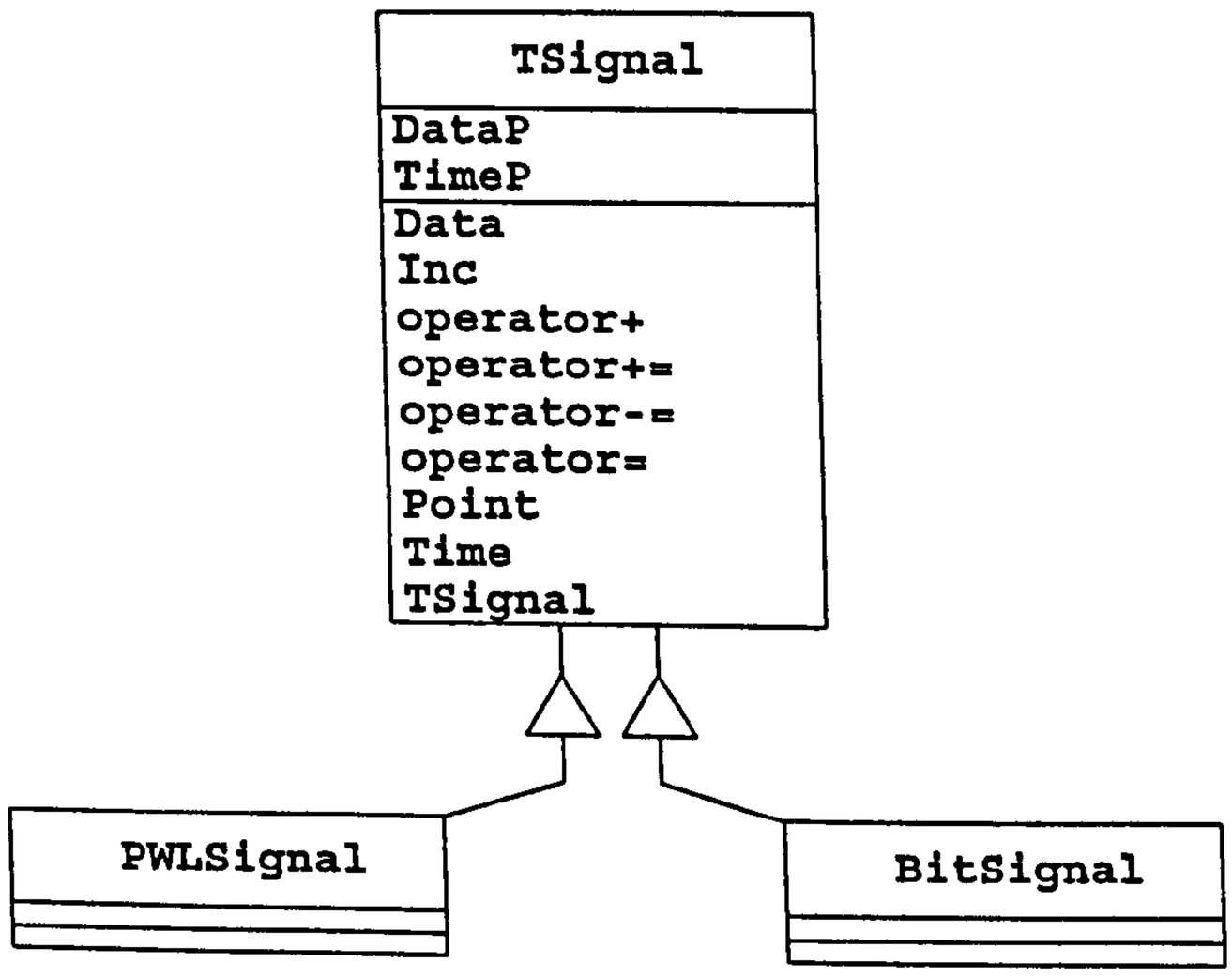


Figure 4-1. Root Classes for Representing Signals.

corresponding to the magnitude and time of the point respectively, together with operators required to perform commonly required processing of the point. The attributes are private to the *TSignal* class. This means that they can only be accessed via *TSignal's* public operators and makes it simple to change the internal representation used. The initial values of *DataP* and *TimeP* are set when the *TSignal* constructor function is called. The other operators enable the values of *DataP* and *TimeP* to be retrieved and updated. The *Point* operator is used to write new values to *DataP* and *TimeP* while the *Inc* operator is used to increment their existing values. The *Data* and *Time* operators are used to access or change the values of *DataP* and *TimeP* respectively. They make use of the C++ overloading facility to implement different functions that exhibit intuitive behaviours. Thus the statements:

```
a_data_item = a_signal.Data();
a_time = a_signal.Time();
```

retrieve the values of *DataP* and *TimeP* from the object *a_signal* while:

```
a_signal.Data(a_data_item);
a_signal.Time(a_time);
```

write new values to *DataP* and *TimeP* to the object *a_signal*.

The other operators overload the standard C++ operators "+", "+=", "-=" and "=" so that *TSignals* (i.e. PWL and PWC points) can be directly added, subtracted and copied. This simplifies the coding of functions that process *TSignals* (e.g. interpolation between points).

The *TSignal* class is not directly instantiated to form objects. Instead it is used as a C++ **template** for objects belonging to the *PWLSignal* or *BitSignal* classes. Figure 4-1 shows that the *PWLSignal* and *BitSignal* classes add no new attributes or operations to the *TSignal* class. They are only required in order to differentiate between objects representing PWL and PWC signals. The definitions of some attributes and operations vary between the *PWLSignal* and *BitSignal* classes. In a pure OO environment, this would cause poor cohesion in the class hierarchy. However, the **template** feature of C++ provides a way around this. All declarations in the *TSignal* class that depend on the signal

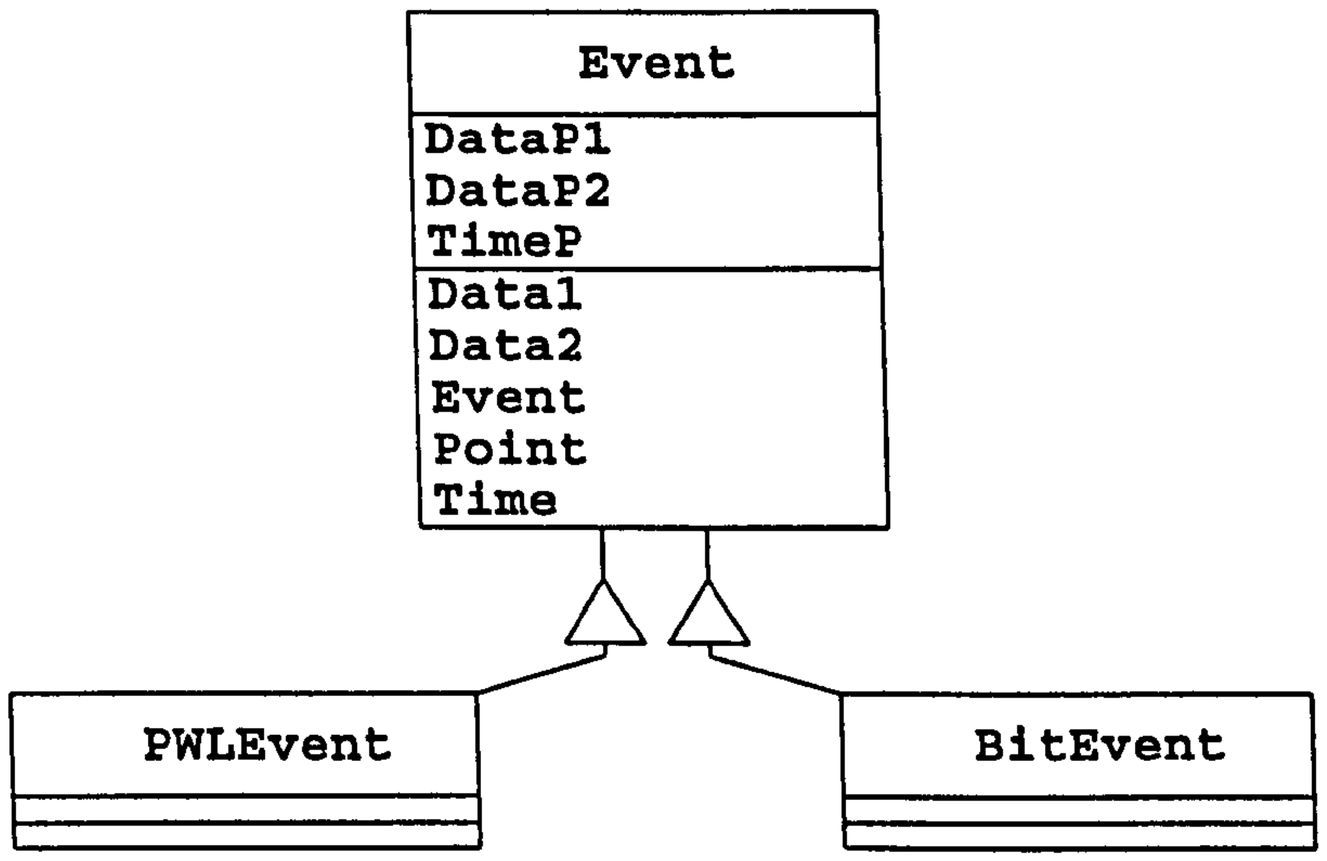


Figure 4-2. Root Classes for Representing Simulation Events.

type are defined using a template parameter *<class T>*. Objects of *PWLSignal* and *BitSignal* classes can then be created using the *TSignal* constructor function with *T* set to *PWLType* or *BitType* respectively.

Simulation models that process two independently changing input signals are required to form events containing the values of each input at significant time points as described in Chapter 3. An *Event* class to store such an event was therefore created. This is illustrated in Figure 4-2. The *Event* class is similar to the *TSignal* class except that it contains two data values for each time point. It is also used as a template to form objects representing events involving two *PWLSignal* inputs or two *BitSignal* inputs.

The *PWLSignal* and *BitSignal* classes provide the storage for a pair of signal co-ordinates (time, magnitude). These co-ordinates have to be stored in a form that can be accessed sequentially to be of use in a simulator. Classes were therefore designed to create and manipulate sequences of co-ordinates existing in *PWLSignal* or *BitSignal* classes. These classes provide the interface between component model classes and the signal points. Rather than design classes to directly implement sequences of *PWLSignals* or *BitSignals*, it was decided to create a class hierarchy where these classes were derived from a common base class. If the component models are written in such a way that their input and output signal sequences belong to the base class, they can also process signal sequences belonging to any of the derived classes.

The base class used for storing sequences of signals (*ADataStore*) is shown in Figure 4-3 together with the derived *TDataStore* class. The *ADataStore* class is abstract, i.e. it cannot be used to directly create any objects. It defines some of the attributes and operations that are common to all derived classes and provides outlines for some of the others. A unique name is associated with each signal in POISE (see Appendix A, section 6.3.2), this is the equivalent of the node name/number used in SPICE type simulators. The *Identify* operator is used to return this name. The datastore classes are designed to be read and written to independently. The *ADataStore* class therefore has attributes *ReadCount* and *WriteCount* that store the address of the last location read from or written to respectively. The *Next* operator increments the *ReadCount* attribute while *ReRead* and *ReWrite* reset *ReadCount* and *WriteCount* to the start of the signal sequence respectively. The *DSize* attribute stores the number of points in the signal sequence and is accessed with the *Size* operator. It is used by POISE to ensure sufficient memory is allocated for related signal sequences. The *EndData* and *AtEnd* operators in *ADataStore* are virtual functions. This means that they are declared but not defined within the *ADataStore* class. All derived classes must therefore include a definition of these operators (or an error is reported by the compiler). These operators mark and detect the end of a signal sequence respectively. They cannot be defined in the base class since their implementation depends on the signal type.

The nature of the proposed simulator (see Appendix A, Figure 6-5) allows the processing of a model to be suspended and control passed to subsequent models before the end of the input waveforms has been reached. This situation must be flagged so that subsequent models do not attempt to process signals occurring after this simulation time point. The *SuspendData* operator marks the end point of the output waveform when a model is suspended. The *AtSuspend* operator is used by subsequent models to detect the corresponding flag in their input waveforms. Both of these operators are also virtual functions since their definition depends on the signal type.

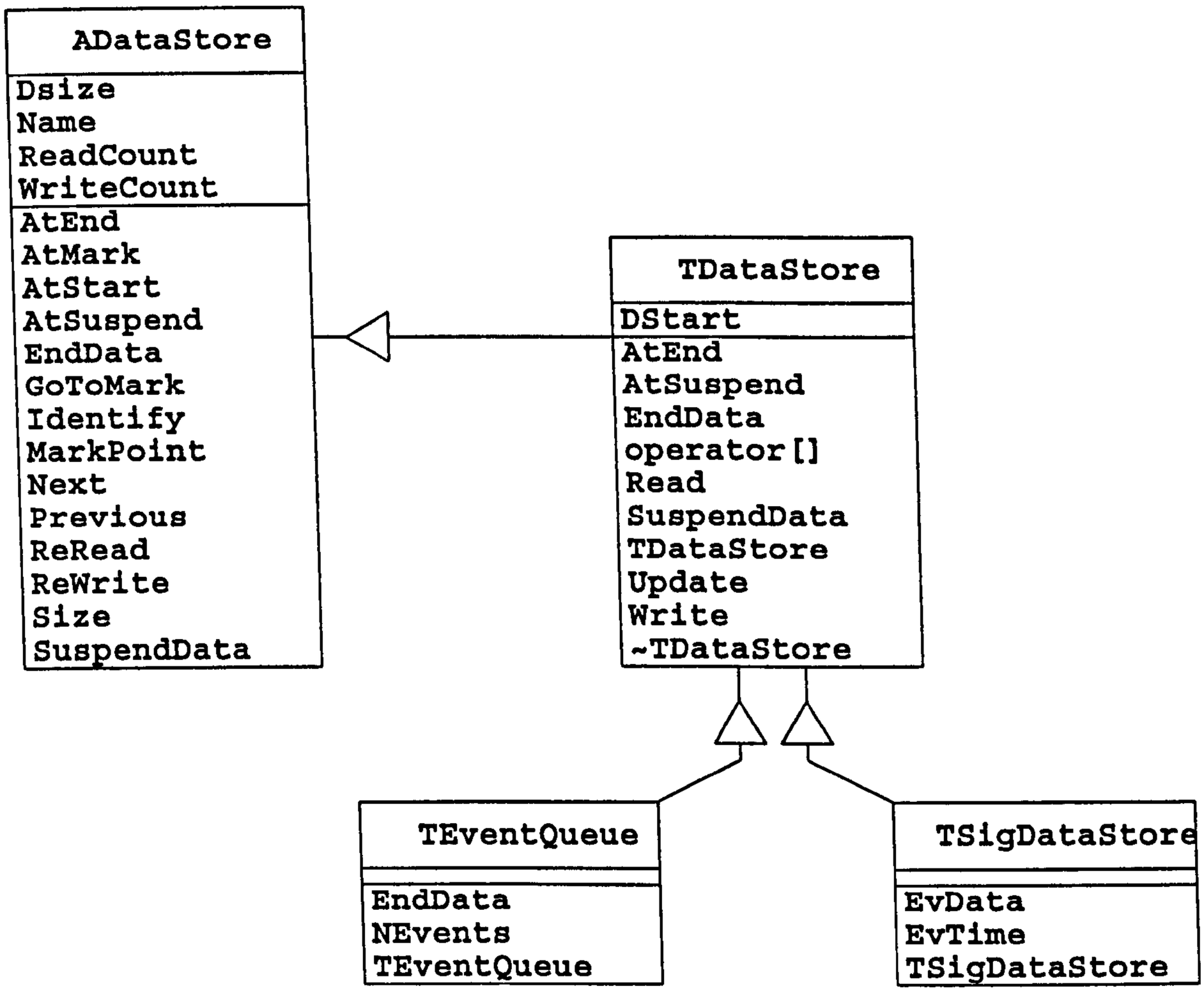


Figure 4-3. ADataStore and TDataStore Classes.

A waveform might be read by a number of models during the course of a simulation (e.g. if an output has a fan-out greater than one). A model must therefore be able to locate the correct points of its input waveforms when it resumes processing after suspension. This is achieved with the *MarkPoint*, *AtMark* and *GoToMark* operators.

The *TDataStore* class is a specialisation of the *ADataStore* class and includes the attributes and operators that depend on the signal class type. It has constructor and destructor functions to enable dynamic creation and destruction of objects based on the *TDataStore* class (*TDataStore* is used as a template for classes containing specific signal types like the *TSignal* class). The constructor function is overloaded so it can create an empty datastore of a given size or can build a datastore from an input file given a filename. The *Read*, *Update* and *Write* operators return the address of the appropriate point and are used in conjunction with the operators from the *TSignal* class to manipulate

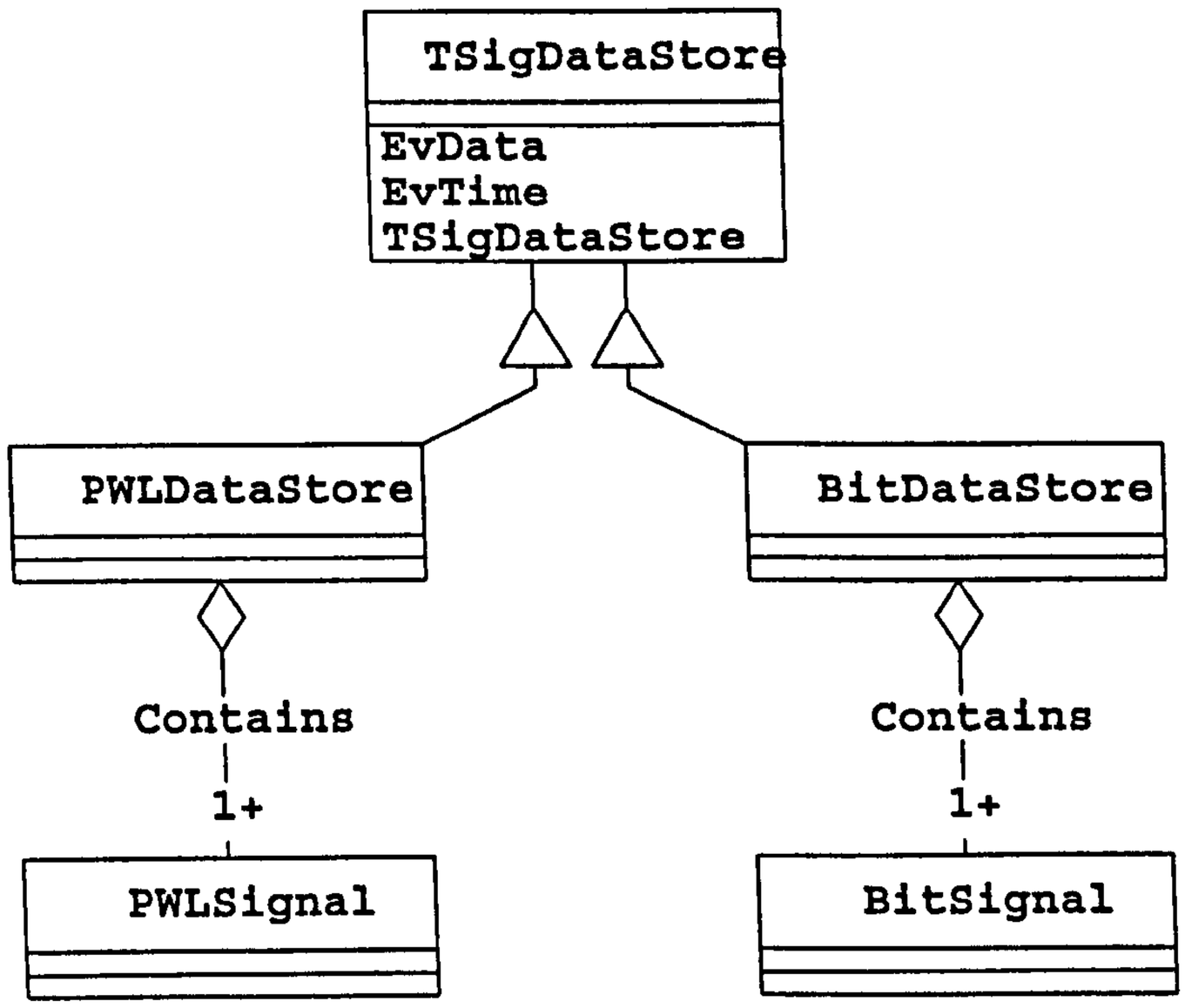


Figure 4-4. DataStore Classes for Specific Signal Types.

the points. An overloaded *Write* operator writes the contents of a signal sequence to an output file. The *DStart* attribute and *[]* operator are used to enable the signal sequence to be processed in the same manner as the standard C array type.

Figure 4-3 shows two classes derived from the *TDataStore* class: *TSigDataStore* and *TEventQueue*. These are used for storing sequences of signals and simulation events respectively. The *TSigDataStore* class includes two extra operators (apart from its constructor). The *EvTime* operator interpolates the waveform and returns the time when a particular magnitude is reached. The *EvData* operator performs a similar interpolation but returns the magnitude at a particular point in time. The *TEventQueue* class also includes two extra operators. The *EndData* operator overloads that inherited from the *TDataStore* class. The *NEvents* operator returns the number of simulation events that are contained in a *TEventQueue* object.

The use of the *TSigDataStore* class as a template to create classes related to a specific type of signal is shown in Figure 4-4. This shows that the derived *PWLDataStore* class contains one or more *PWLSignal* objects while the derived *BitDataStore* class contains one or more *BitSignal* objects.

4.2.2 Simulation Model Classes.

The hierarchy of classes representing simulation models has an abstract base class that defines the attributes and services that are common to all models. This is the *TGenComponent* class shown in Figure 4-5. It has two attributes: *Name* and *Status*. The *Name* attribute is used to store an instance name for each model object. It is set by the *TGenComponent* constructor and is retrieved with the *Identity* operation. The *Status* attribute is a flag that is used to indicate the state of a model. It is initialised to the *START_MODEL* state once a model object has been created to indicate that the model is ready for use. When a model detects the end of one of its input waveforms, *Status* is set to the *END_MODEL* state to prevent further processing. The *GetStatus* operation returns the state of *Status*.

The TGenComponent class contains several other operations: *Reset*, *Run*, *SaveState*, *SetState* and *SuspendTime*. These are all virtual functions. Specialised classes based on *TGenComponent* must provide definitions for these operations in order to be instantiated as objects. This ensures consistency of the interface provided by derived classes regardless of the signal types used.

The *TGenComp1* and *TGenComp2* classes also shown in Figure 4-5 provide the base classes for component models that process one and two input signals respectively. The *InVectorPTR1* and *ResultPTR* attributes of *TGenComp1* store pointers to the datastore objects for the input and result signals respectively. The *TGenComp1* class provides definitions for the *SuspendTime* and *Reset* operations: *SuspendTime* returns the time of the last point written to a model's output waveform when execution is suspended and control passed to subsequent models. It is used by the subsequent models to ensure they don't attempt to process further points once this time is reached. The *Reset* operation is provided so that the simulator can ensure that all waveforms are reset to time zero prior to a new simulation commencing. The *SaveState* and *SetState* operations save and restore respectively the location of the last point in the input waveform to be processed before the operation of a model is suspended (using *Marker1*).

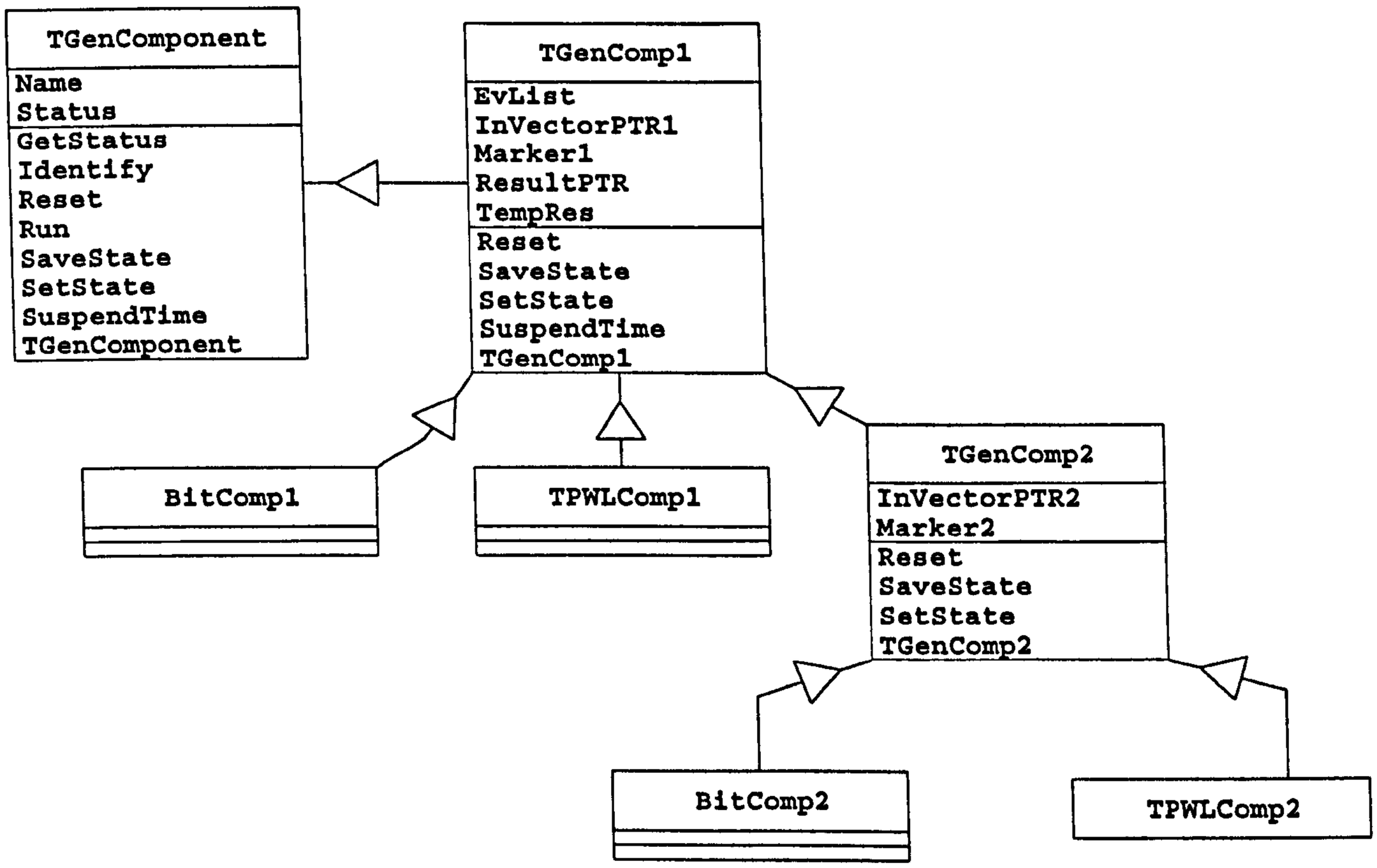


Figure 4-5. Base Classes for Simulation Models.

The *TGenComp2* class is derived from *TGenComp1* and so inherits all of its properties. *TGenComp2* has additional attributes to store the address of the second input signal datastore (*InVectorPTR2*) and the location of the last point in this waveform to be processed when a model is suspended (*Marker2*). It provides *Reset*, *SaveState* and *SetState* operations that hide these operations in its parent class. These new operations are required to process the datastore corresponding to *InVectorPTR2* as well as *InVectorPTR1* (inherited from *TGenComp1)*.

The *TGenComp1* and *TGenComp2* classes are C++ templates that are used to build the signal-dependent classes *PWLComp1*, *BitComp1*, *PWLComp2* and *BitComp2*. These form the base classes for model building blocks and component models that require interfaces to specific types of signals. None of the classes in Figure 4-5 provide a definition for the *Run* operation. They are all therefore abstract classes. Concrete classes (i.e. capable of forming objects) derived from the *PWLComp1* class that represent analogue model building blocks are shown in Figure 4-6. These correspond to the building blocks discussed in Chapter 3. Each class has an attribute that controls its behaviour (set by its

constructor) and a *Run* operation that performs the required processing. The structure of the building blocks with two input signals and for digital models is similar. Although the *Add* and *Multiply* classes are shown in Figure 4-6, these were only implemented for classes with two input signals since a facility for the addition or multiplication of a PWL waveform by a scalar quantity is provided within the *DataStore* class.

The structure of classes to implement component models is illustrated by the class hierarchy in Figure 4-7. This shows three different types of model classes that can be derived from the *PWLComp1* class represented by *DCShift*, *FeedbackAmp* and *Resistor*.

The *DCShift* class is an example of the simplest type of model. It adds a constant offset to the magnitude of the input signal. It contains two operations: a constructor that initialises its single attribute (*Offset*) and *Run* that processes the input waveform and generates the output waveform.

The *FeedbackAmp* class defines the *Run* operation and so is not abstract. However, it is used to build models of various types of feedback amplifier and so is not instantiated as an object itself. A feedback amplifier has components connected to its input terminal and in the feedback path. The *FeedbackAmp* class contains two attributes, *InputModelPTR* and *FbackModelPTR* that point to the model objects representing these components. These pointers are used by the *Run* operation to invoke the *Run* operations in each of the constituent models. Objects created from classes derived from *FeedbackAmp* are

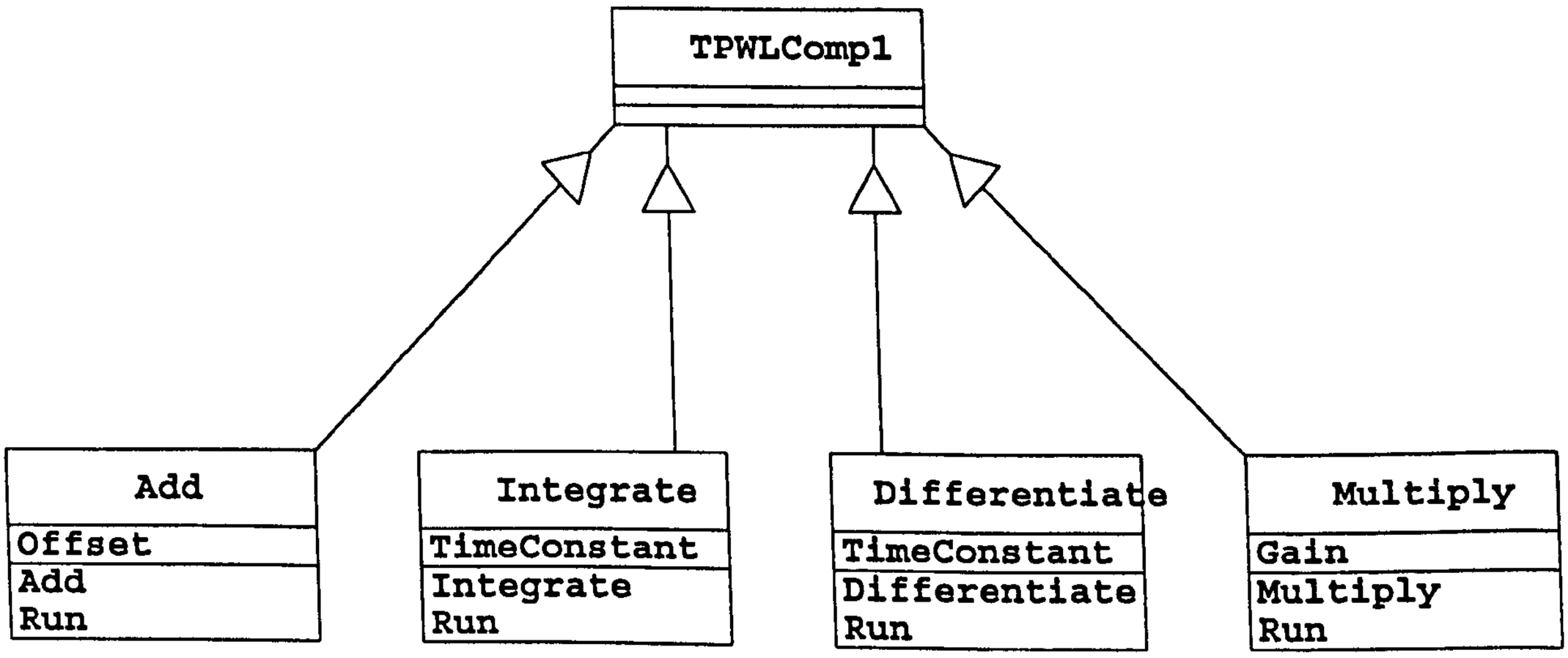


Figure 4-6. Classes for Analogue Building Blocks.

responsible for the initialisation and operation of their constituent model objects. The *Reset* operation is therefore redefined in the *FeedbackAmp* class to invoke the *Reset* operations of the constituent classes.

Figure 4-7 shows two examples of classes derived from the *FeedbackAmp* class: *Filter* that is used to build active filters and *ResAmp* that implements a resistive negative feedback amplifier. The *ResAmp* class doesn't have any additional operators from its parent except for its constructor function. It creates objects representing its input resistance and feedback resistance using the *ResVI* and *ResIV* classes respectively. Since these classes are both derived from the *TGenComponent* class, the pointers inherited from the *FeedbackAmp* class can be used without redefinition.

The *Resistor* class in Figure 4-7 is abstract since it doesn't define the *Run* operation. It is used to form two different types of resistor model where one terminal is connected to a reference voltage source:

1. A resistor that generates a current output signal from a second voltage source applied

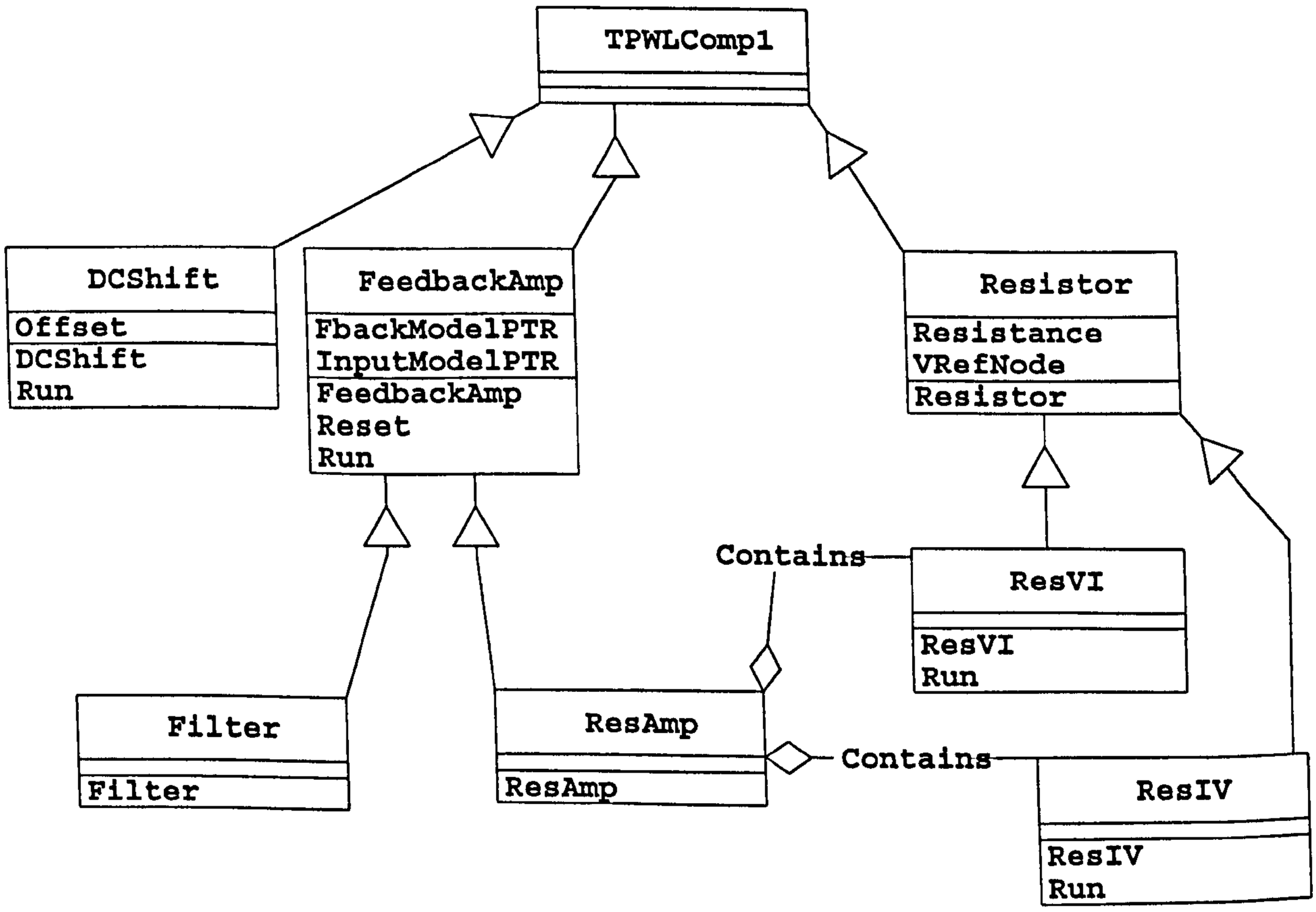


Figure 4-7. Example of Class Hierarchy for Analogue Component Models.

to the other terminal (*ResVI*);

2. A resistor that generates a voltage output signal from a current source attached to the reference terminal (*ResIV*).

It has attributes to represent the resistance and the magnitude of the reference voltage. The *ResVI* and *ResIV* child classes define the required behaviour in the *Run* operation and so can be used to form objects. The use of these classes is described in section 4.4.4.

The classes shown in Figure 4-7 represent the upper levels of the analogue component model class hierarchy. More specialised classes are derived from these to build models with more complex behaviour that provide a closer approximation to physical components. Since all component classes are derived from the *TGenComponent* class, they all have a consistent interface, simplifying the task of replacing one model with another at a different level of abstraction.

The structure of the simplest class for digital models is shown in Figure 4-8. The *TPWLDComp1* class forms an abstract base class for all single-input digital models with PWL input waveforms. The *Delay*, *TFall* and *TRise* attributes control the dynamic behaviour of the model whilst the *VThresh*, *VCC* and *VDD* attributes determine the DC

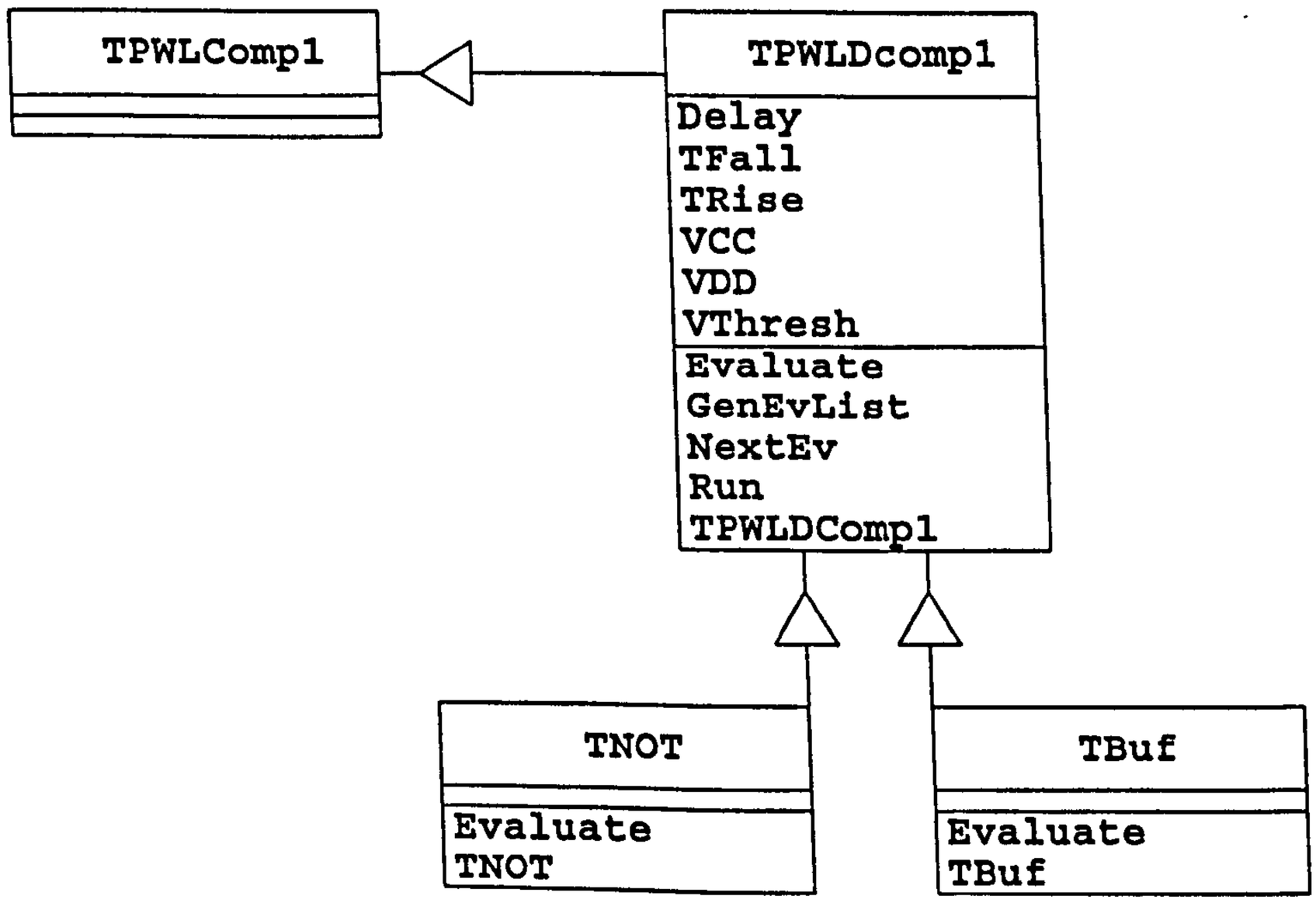


Figure 4-8. Classes for Single-Input Digital Models.

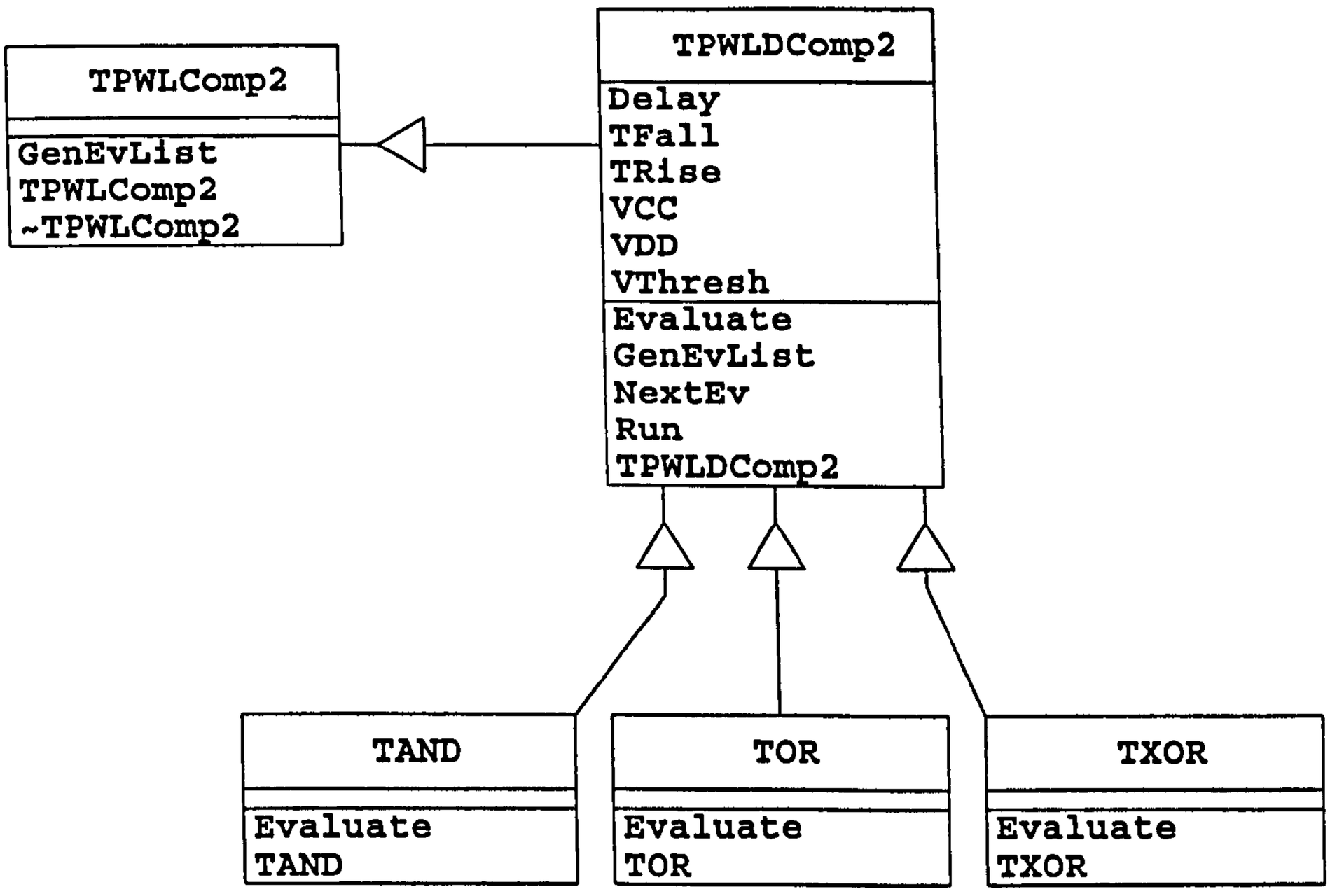


Figure 4-9. Classes for 2-Input Digital Models

characteristics. The *Run* operation invokes the *GenEvList*, *NextEv* and *Evaluate* operations for any class derived from *TPWLDComp1*. The *Evaluate* operation is a pure virtual function and so is implemented by the *Evaluate* operation in each child class. The operation in the child class (*TNOT* or *TBuf*) generates the appropriate output waveform from the event-list. The event-list is generated by the *GenEvList* operation: this is suitable for all single-input digital gate models.

The classes for representing two-input digital gate models are shown in Figure 4-9. The *TPWLDComp2* class has attributes and operations with the same name as the *TPWLDComp1* class. Whilst the meaning of these attributes and operations is the same, their natures are different. The operations in particular are more complex that for the *TPWLDComp1* class since they must process two independently changing input waveforms. Figure 4-9 shows three classes that are derived from the *TPWLDComp2* class. These each include an *Evaluate* operation that generates the appropriate output waveform whilst their parent class generates the event list, output levels and output timing (as with the single-input digital gate models.

4.3 Models of Digital Circuits.

4.3.1 Structure of Simple Models.

Models of logic gates that process and produce PWL signals were created using the digital building blocks described in Chapter 3 (AND,OR,NOT and XOR). These models use the class structures for single-input and two-input digital models shown in Figure 4-8 and Figure 4-9 respectively. Each model inherits all the operations and attributes of its parent class: *TPWLDComp1* or *TPWLDComp2* for models with one or two PWL inputs respectively. The parent classes were defined and refined as the initial digital models were developed and tested. The reuse of common parent classes reduced the development of further models to a trivial task since the only differences between models are in their *Evaluate* operations. Consequently, models were also created for other one- and two-input combinational logic gates (non-inverting buffer, NAND, NOR and XNOR).These additional models provide extra building blocks to simplify the construction of more complex models.

Each model inherits *Reset*, *SuspendTime*, *SetState* and *SaveState* operations from its *TGenComp1* or *TGenComp2* parent class. These operations provide the only mechanism for controlling the execution of the models by a simulator program. The *Run* operation is inherited from a model's *TPWLDComp1* or *TPWLDComp2* parent class. The *GenEvList*, *NextEv* and *Evaluate* operations are only invoked by the *Run* operation and are not visible to the simulator. The *Run* operation must therefore be invoked to simulate a model. As the *Run* operation is inherited from an abstract parent class, it follows that all models provide a consistent interface to the simulator. This approach simplifies the investigation of various simulation strategies and modelling techniques: changes can be made to the operation of the simulator and the models independently so long as the behaviour of the interface remains consistent.

The *Run* operation processes the input waveforms for a set time (*FreeRunTime*) and produces a corresponding output waveform. The steps performed by the *Run* operation are shown in Figure 4-10. The event list (*EvList*) is generated by the *GenEvList* operation from the input waveforms as described in Chapter 3. This uses the *NextEv* operation to

find the next event on each input waveform. The *GenEvList* operation detects the *EndData* and *SuspendData* flags in the input waveforms and sets the model's *Status* flag if either is found. It also sets the *Status* flag if the *FreeRunTime* is exceeded. The *Status* flag is monitored by the *Run* operation which will only process the input waveforms while *Status* is set to *RUN_MODEL*. The results of the *Evaluate* operation are written to a temporary data store (*TempRes*) contained within the model. *TempRes* has one point corresponding to each event generated from the inputs. The contents of *TempRes* are compared to the output waveform so that new points are only written to the output when necessary and momentary invalid results (glitches) can be removed.

Every output transition requires two points to be written to produce finite rise and fall times. The propagation delay from the input to output is added by the *Evaluate* function. The time of the start of each output transition can therefore be obtained from *TempRes*. The rise or fall times are then added to *TempRes* and the point defining the end of the transition written.

The *Evaluate* operations are simple and only require a few lines of C++ code. This can be illustrated by the *Evaluate* operation for an exclusive-OR gate model shown in Figure 4-11. The first two lines of code set a flag (*dig1* or *dig2*) to a 1 or 0 according to whether the value of the current event in the event queue is greater than a threshold value (*VThresh*). The next line of code determines whether the output should be set to VCC or VDD. The output value is then written to the internal output buffer (*TempRes*) after the propagation delay.

The *Run* and *Evaluate* operations for digital models with one input are similar to those for two input models. The *GenEvList* operation is much simpler since it only has to find an event in one signal. The *EvList* and *TempRes* attributes are therefore only required to store a single point and so can make more efficient use of memory as their exact size is known at compile time.

```
int TPWLDComp2::Run(TimeType FreeRunTime)
{
if (Input1 or Input2 at start) write initial value to Results;

ResumeTime = last time point written to Results;
if (Status == SUSPEND_MODEL) Status = RUN_MODEL;

while (Status == RUN_MODEL)
{
        // loop until FreeRunTime or end of data

        Generate EvList;
        Reset TempRes;
        for (all events found) Evaluate(EvList,TempRes);
        Mark end of TempRes;
        for (all points in TempRes)
                {
                if (TempRes different from Results)
                        {
                        if (glitch detected) update previous point to remove glitch
                        else
                                {
                                write previous value to Results;
                                add delay for TRise or TFall to TempRes;
                                Results = TempRes;
                                }
                        }
                }
switch (Status)
        {
        case END_MODEL:
                write last point to end of Results;
                Results =  EndData flag;
                break;
         case SUSPEND_MODEL:
                Results = SuspendData flag;
                break;
                default:
                        continue;
                };
        }
return Status;
}
```

Figure 4-10. Run Operation for Digital Models with 2-inputs.

```
void TXOR::Evaluate(PWLEvStore* EvQ, PTADataStore TempRes)
{
        int dig1 = (EvQ->Read().Data1() > VThresh) ? 1:0;
        int dig2 = (EvQ->Read().Data2() > VThresh) ? 1:0;
        PWLType out = ((dig1 && !dig2)||(!dig1 && dig2)) ? VCC:VDD;
        TempRes->Write().Point(EvQ->Read().Time()+ Delay,out);
}
```

Figure 4-11. C++ Code for Exclusive-OR Model.

4.3.2 Validation of Simple Models.

Models were written for the simple logic gates discussed in the previous section. These were initially tested with short PWL waveforms so that their function could be verified and any design or coding errors detected and corrected. Most errors were related to the processing of input waveforms and the identification of events for the models with two inputs. The input waveforms are not required to be synchronised, do not usually contain the same number of points and do not have to end at the same time. Events will not usually occur on both waveforms at coincident times but if they do, the correct states should be written to the event queue. These characteristics require the *GenEvList* operation to be fairly complex (about 40 C++ statements). Fortunately, this operation is inherited by all two-input digital models from the *TPWLDComp2* class so modifications were only required in the parent class and not individual models.

The proposed simulation methodology permits each model in turn to process its input waveforms for a given time interval (see Appendix A, section 6.3.3). It is possible that during a particular time interval, a number of events could be detected in one input waveform of a two-input model whilst none are detected in the other. It is not possible to evaluate the model until such time as an event is detected on the second input waveform and its state can be determined. If no events are detected in the second waveform before the end of the simulation, the output waveform would be incomplete. This is undesirable. It is also unnecessary if the lack of further events in an input waveform can be assumed to indicate that the waveform maintains its previous state until the end of the simulation. This assumption makes it possible to determine the effects of events on the other waveform and so generate the appropriate output waveform up to the end of the

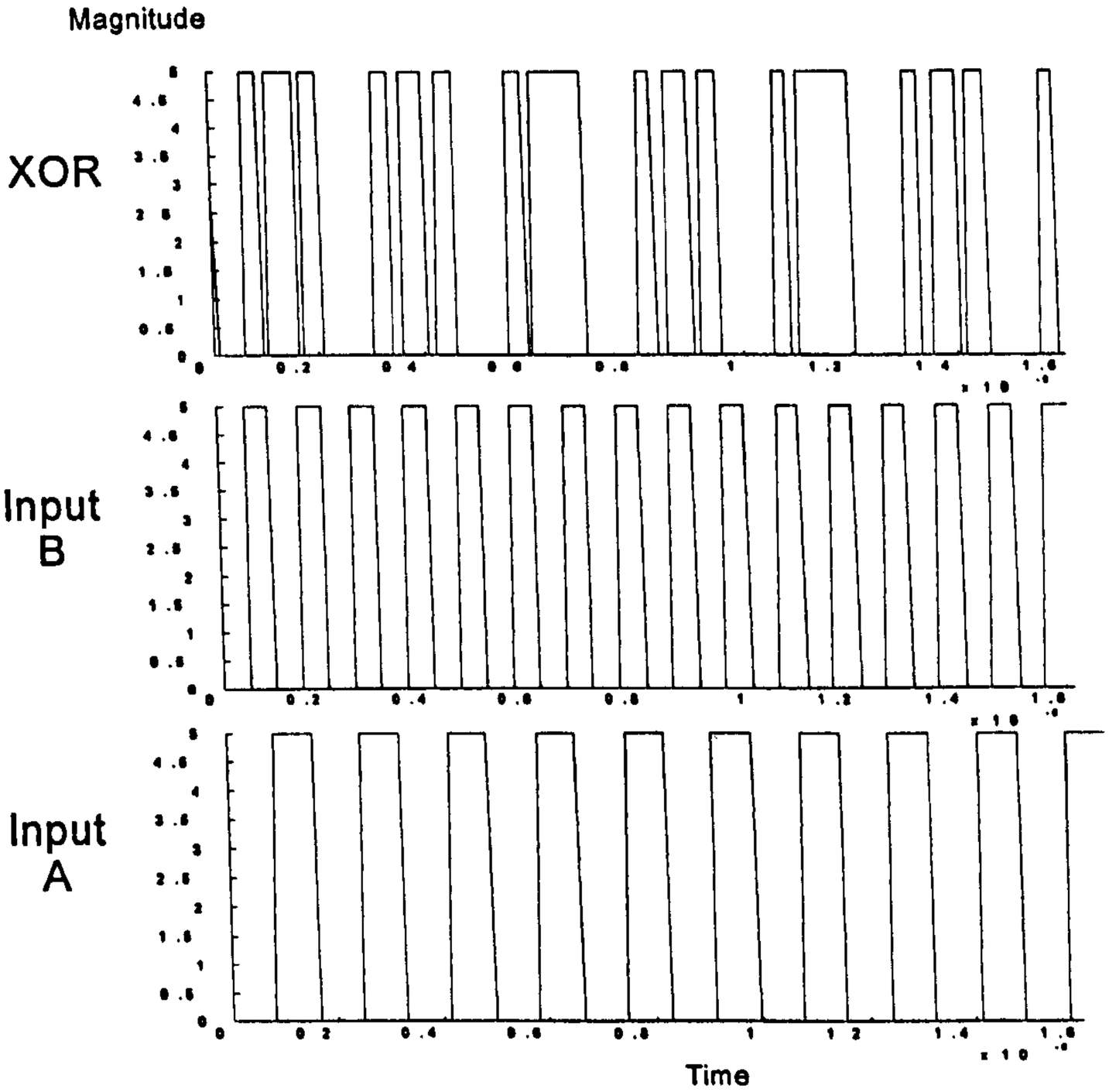


Figure 4-12. Simulation Results for PWL XOR Model.

simulation period. The *GenEvList* operation was modified to make use of this assumption to generate the appropriate event queue when the *EndData* flag was detected in either input waveform.

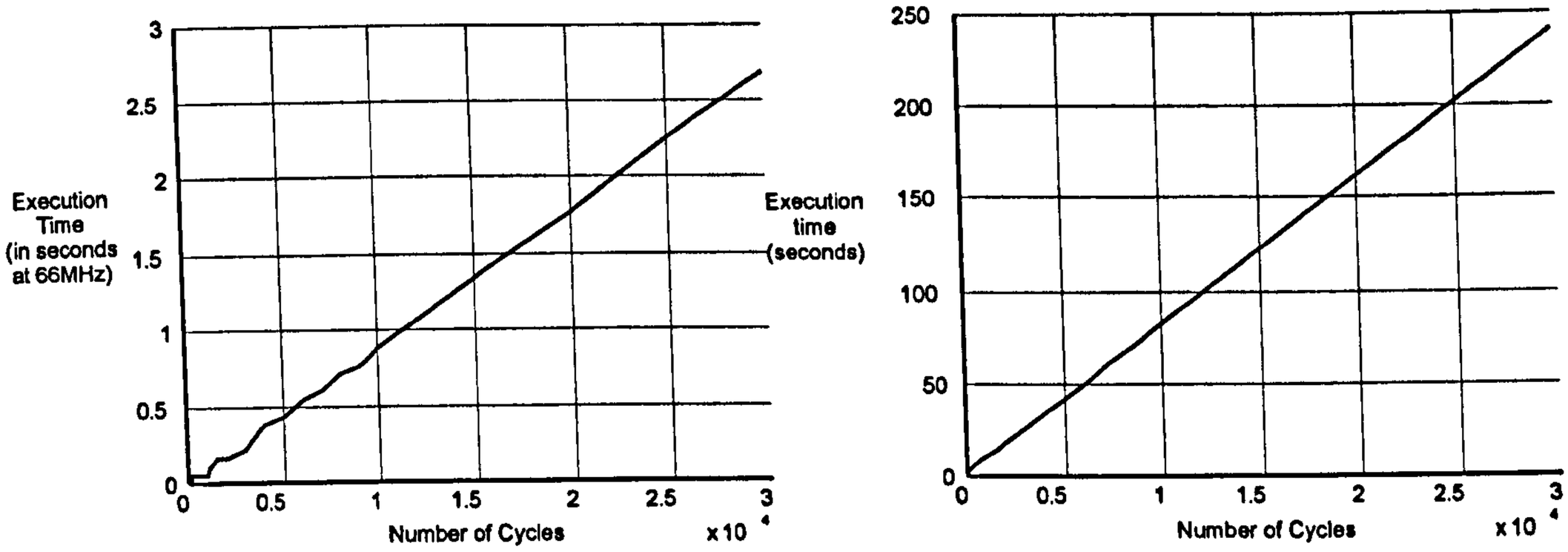
The output waveforms from the modified models produce the expected results. Sample results for an XOR model are given in Figure 4-12. This model had its propagation delay set to 5ns with output rise and fall times of 5ns. The input waveforms are 6MHz and 10MHz square waves, also with rise and fall times of 5ns. The output waveform would be expected to consist of groups of narrow pulses corresponding to the times when the states of the input signals were different. However, Figure 4-12 shows occasions where two pulses have merged to form a single wider pulse. This is an intentional feature built into the models to reflect observed physical behaviour: very short pulses are unlikely to be passed through the output stage of a logic gate. For this model, the minimum output pulse width was taken to be 5ns. When pulses of shorter duration are detected, their points are removed as the output waveform is written.

4.3.3 Comparison of Simple Model Performance.

The performance of the digital models described in the previous sections was investigated by a series of tests using the Windows-based demonstration simulator (POISE) described in Appendix A. This enabled direct comparisons to be made with a widely-used commercial mixed-signal simulator (PSPICE) that ran in the same environment (Windows version 3.11 on a 486 DX4-100 or DX2-66 personal computer with 16 Mbytes of RAM). The potential performance of PSPICE with simple models is significantly degraded by writing the state of all internal nodes to an output file at every print step (the default behaviour). To make a fair comparison of both simulators' performance, the generation of output files was suppressed to prevent the (relatively slow) disk access times from dominating the PSPICE simulation execution time. Only digital models were used for these tests. Voltage levels were therefore translated into logic states by PSPICE without consideration of signal rise and fall times and other loading effects.

The first performance test investigated how the execution time of a simulation was related to the number of cycles in the input waveform. A periodic square-wave signal was applied to both inputs of a two-input NAND gate. The time taken for the simulation to complete was recorded for input signals of varying lengths. The test results are shown in Figure 4-13. The relationship between the execution time and the number of cycles is approximately linear for both simulators. The deviations from a linear characteristic in Figure 4-13 a) reflect the difficulties encountered in obtaining accurate execution times for simulations that took much less than a second to complete. This test showed that POISE completed each simulation approximately 90 times faster than PSPICE.

Comparisons were also made with the Viewlogic version 5.1 digital simulator (ViewSim) and the SMASH mixed-signal simulator. Both of these simulators also ran in the same environment. The performance of ViewSim depended greatly on whether it was required to produce output waveforms. With no output waveforms produced, the execution times were too small to be recorded accurately (a time-logging facility was not available in this version of the simulator so timing was done by hand using a stop watch). However, when the output from ViewSim was sent to a waveform file or monitor, its performance was similar to PSPICE. The performance of SMASH was similar to that of POISE provided



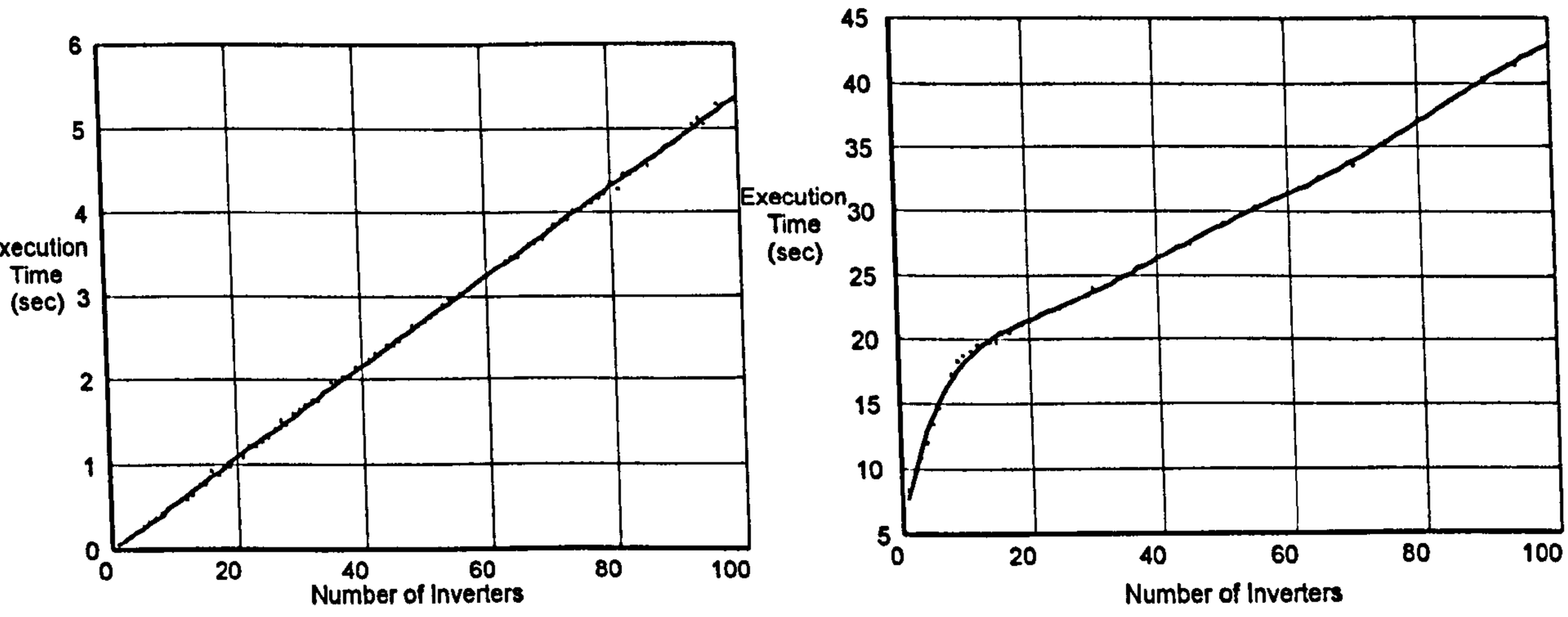
a) PWL Simulator (POISE) b) PSPICE

Figure 4-13. Simulation Execution Times for 2-Input NAND Gate.

no output signals are generated. If output waveforms are enabled, the performance becomes similar to that of Viewlogic and PSPICE.

Tests were also made to investigate how the simulation time was related to the number of logic gates in a circuit for POISE and PSPICE. A periodic square-wave signal was applied to circuits consisting of up to one hundred inverters connected together in a single chain and the simulation time for 1000 waveform cycles recorded. The results of these tests are plotted in Figure 4-14. The execution time for the PWL simulation varies almost linearly with the number of inverters (approximately 53ms per gate per 1000 cycles). Since each gate model is evaluated sequentially, a linear relationship was expected.

The execution time for PSPICE increases sharply as the number of gates is increased



a) PWL Simulator (POISE) b) PSPICE

Figure 4-14. Simulation Times for Inverter Chains of Various Lengths.

when the total number of gates is small (less than 15). However, as the total number of gates in the circuit is increased beyond this point the rate of increase of the execution time is reduced. When there are more than 20 gates in total, the execution time increases by approximately 260ms per gate per 1000 cycles. This is about 5 times higher than the PWL simulation. It confirms that the PWL simulation approach is more efficient than PSPICE when evaluating behavioural models of digital circuits. However, the almost linear relationship between the execution time and number of inverters for PSPICE disagrees with published results that suggest this relationship should be approximately exponential [54]. This difference might be due to the untypical nature of these test circuits since each node only connects a single model output to a single model input.

4.3.4 Construction and Validation of Complex Models.

Models of complex digital components can be constructed by connecting the simple models together. To illustrate this, a structural model of an exclusive-OR gate was created from simple models as shown in Figure 4-15 and the simulation results and execution times compared to those for the simple behavioural model.

The development of this model revealed the need for the *SaveState* and *SetState* operations described in section 4.2.2. The simulation methodology activates each model in turn for an allocated simulation interval. Each model must therefore be able to resume processing its input waveforms at the correct point. The structure of the signals are hidden from the models to maintain the principle of encapsulation. The location of the last point to be processed is stored within the signal object. If a signal is only processed by one

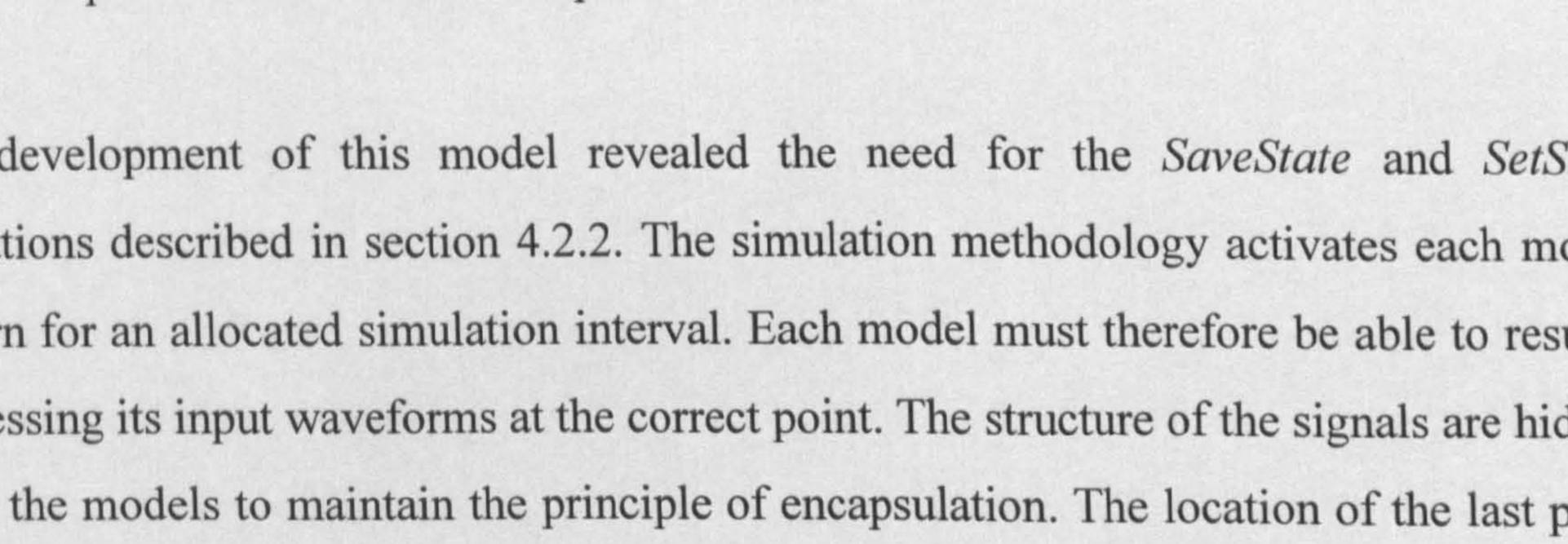


Figure 4-15. Structural Model of an XOR Gate.

model input, the correct point will always be retrieved. However, signal A is processed by both the NAND1 and AND2 models at different points in the simulation. Consequently, if the AND2 model was activated after the NAND1 model, it would incorrectly resume processing the A signal at the last point to be processed by NAND1. The *SaveState* operation stores the location of the last point processed in a *Marker* attribute within the model when a model is suspended. The *SetState* operation restores this location when a model is reactivated. This does not break the principle of encapsulation since the *Marker* attribute can only be used by the *SetState* operation and does not provide the models with any information about the structure of signal objects.

The simulation methodology initially used in POISE allowed each model to run for a set time (*FreeRunTime*). When a model's input waveforms reached this time, the value of *FreeRunTime* was updated to reflect the last point written to the output waveform. This ensured that subsequent models didn't attempt to read non-existent points beyond this time. When all models had been evaluated, the value of *FreeRunTime* was incremented and the simulation repeated. This continued until the end of the input waveforms was detected. The structural XOR gate model revealed problems with this approach. These problems were caused by the order in which models of parallel circuit elements were evaluated. If the AND1 model was evaluated before the AND2 model, it would always prevent the AND2 model from running beyond the time of the last point written to the NAB waveform. This would prevent the AND2 model from evaluating any events in its input signals occurring after this time. In this case, the simulation would never complete. These problems were solved by defining a *SuspendData* flag to mark the last point written to an output waveform when the model generating it is suspended. This flag is detected when models process their input waveforms, removing the requirement for models to alter the value of *FreeRunTime* used by subsequent models.

These alterations enabled POISE to process the structural XOR gate model correctly. The results generated by this model were inspected and found to agree with those of the simple behavioural model. There were small differences in the timing of the output waveforms since each component of the structural model had finite propagation delays, rise and fall times. This is expected and reflects the more detailed behaviour that is

generally produced by structural models. The simulation performance for the structural model was compared with a similar model in PSPICE using clock waveforms of two different frequencies for the input signals. With a 10MHz and 6MHz input clock and a simulation time of 33.2μs (200 cycles of the 6MHz waveform) the PWL simulation took 0.109 seconds whilst PSPICE required 49.87 seconds (with 100MHz CPU). The PWL simulation was therefore approximately 450 times faster than PSPICE. The execution time varied linearly with the simulation time (number of cycles) for both simulators. The execution time for POISE corresponded to the expected value assuming that it was linearly related to the number of events and the number of models. The execution time for PSPICE was much higher than that found when simulating inverter chains. This suggests that the exponential increase in the execution time of SPICE-type simulators with circuit size reported in the literature does occur when the circuits are more complex than simple inverter chains.

4.3.5 Considerations for Circuits with Feedback.

The simulation methodology adopted in the demonstration system allows each model to run independently for the duration of a specified simulation time interval, or until it encounters the end of one of its input waveforms. This approach removes the need for a global event queue and aims to increase the simulation efficiency by reducing the number of times that each model is invoked. This approach assumes that the models representing a system can be ordered in such a way that if they are evaluated in a particular sequence, every model will have its input signals defined for most (if not all) of each simulation interval. Unfortunately, if there is feedback between models, the feedback signal can significantly reduce the number of events in input waveforms that can be processed by models in the forward path in a single iteration, regardless of the length of the simulation interval. In the worst case, where the feedback signal is sensitive to any change in the inputs, the models could only process a single input event on each iteration. Each model would then have to be invoked for every event in its input waveforms (as with conventional approaches).

The ring oscillator circuit shown in Figure 4-16 was used to investigate the effect of feedback on the simulation performance. The *TNOT* model derived from the *TPWLDComp1* class is not suitable for this circuit: the waveform for *Node1* initially contains no events, preventing *Inv1* from generating any points in its output waveform. A new class (*TPWLDComp1F*) was created for digital models with a single input used inside feedback loops. This class was derived from the *TPWLDComp1* class and only redefines the *GenEvList* operation. The revised *GenEvList* operation writes the current input waveform point to the model's event list if it does not detect any new events (changes of logic state). The model will insert a corresponding point in its output waveform and the simulation can proceed. This process is less efficient than the event-driven approach but does not result in any more points in the output waveform since redundant points are still removed when each new point is written. The *GenEvList* operation also monitors the times of the input waveform points and sets the model status to *END_MODEL* when a specified simulation end time is reached (the model status is normally set to this state when the *END_FLAG* is detected in the input waveforms).

A *TNOTF* model class was derived from the *TPWLDComp1F* class and used for each inverter in the ring oscillator circuit. This class was identical to the *TNOT* class but with a different parent class. Simulation over 1000 cycles produced the expected waveforms and took 1.21 seconds. If an execution time of 53ms per inverter model is assumed, the overhead incurred by only processing single events on each iteration is approximately 170μs per event. This value was confirmed by the simulation of a ring oscillator containing five inverters over 1000 cycles that took 1.98 seconds. Digital simulation of a three-gate ring oscillator in PSPICE took 46 seconds for a 1000 cycles. The PWL approach is approximately 40 times faster.

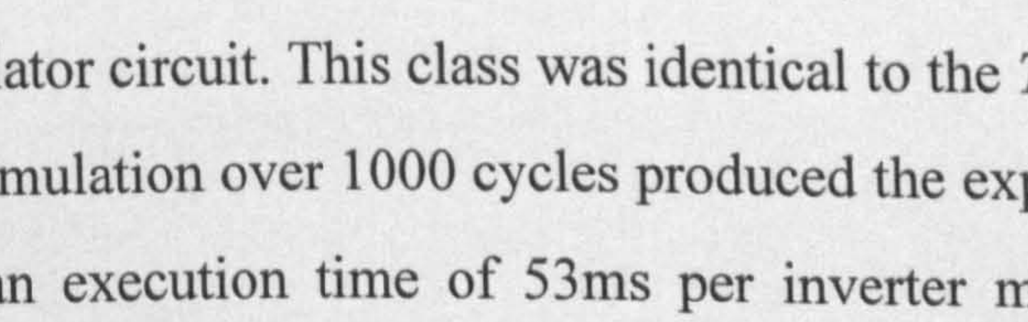


Figure 4-16. Ring Oscillator Circuit.

Simulation of circuits with feedback requires correct initialisation of all signals. POISE does not set the initial state of an output waveform until the model producing it is first evaluated. The correct initialisation of the signals in mixed analogue-digital circuits is a major problem for mixed-signal simulators. The selection of the most appropriate initialisation mechanism requires further investigation (see Appendix A, section 6.4).

The impact of feedback signals on the simulation can be reduced if a circuit is partitioned into blocks whose outputs only depend on the correct propagation of block input signals. This approach would require the development of a suitable partitioning algorithm, similar to those found in simulators based on relaxation methods [24]. Circuit partitioning should also be considered by any future investigation.

4.4 Models of Analogue Circuits.

4.4.1 Structure of Simple Models.

Models of simple analogue components were created using the one- and two-input building blocks described in Chapters 3 (addition, multiplication, integration and differentiation). These models are built from objects defined as part of an object-oriented model class hierarchy in the same way as the digital component models. They make extensive use of the object-oriented inheritance mechanism to simplify their construction and to ensure consistent interfaces between models.

The simplest models are those based on addition or multiplication of a waveform by a scalar quantity. A *DCShift* model was created to perform a signal level shifting function and so represent signal biasing networks. This model adds a scalar quantity to the magnitude of each point in its input waveform. Its output waveform therefore contains the same number of points as the input waveform with the points in both waveforms occurring at coincident time steps. The steps required for the operation of this model are trivial since every point in the input waveform can be processed independently. The scalar quantity (*Offset*) is set when a *DCShift* model object is instantiated.

A *Scale* model was created to represent simple signal gain or attenuation stages (i.e. neglecting frequency-dependency, non-linearity and other deviations from the ideal behaviour). The characteristics of this model are very similar to those of the *DCShift* model: the main difference is that the magnitudes of the input waveform points are multiplied by a scalar quantity rather than incremented by it. The scalar quantity (*Gain*) is set when a *Scale* model object is instantiated.

The integration and differentiation building blocks were used to create models of ideal first order low-pass and high-pass filters respectively. These models also process a single input waveform using a scalar quantity (*TimeConstant*). However, their operation is more complex than the *DCShift* and *Scale* models described above. There are two main factors for this increase in complexity:

1. The magnitude of each point in the output waveform does not depend only on the magnitude of the corresponding input waveform point. The magnitude and time step of the preceding point in the output waveform must also be considered.
2. The approximation of the ideal integrator and differentiator transfer functions assume that the time steps in the input waveform are much smaller than the time constant. If this is not the case, extra time steps must be inserted into the input waveform before it is processed.

These factors prevent each point from being processed independently. The models must monitor the time steps in the input waveform and generate additional points using interpolation if the time steps are not much smaller than the time constant. These extra points must be stored in a local event queue if they are not processed immediately after their generation. The number of extra points inserted should not be fixed as the time taken for the output to converge to a static state depends on the rate of change of the input signal: generating sufficient extra points to cater for all possible input signals would lead to the processing of unnecessary points in many cases. Every extra point inserted into the input waveform will result in an additional point being generated in the output waveform. These additional points in the output waveform are required to maintain the accuracy of the integration and differentiation processes. However, they are likely to cause the output

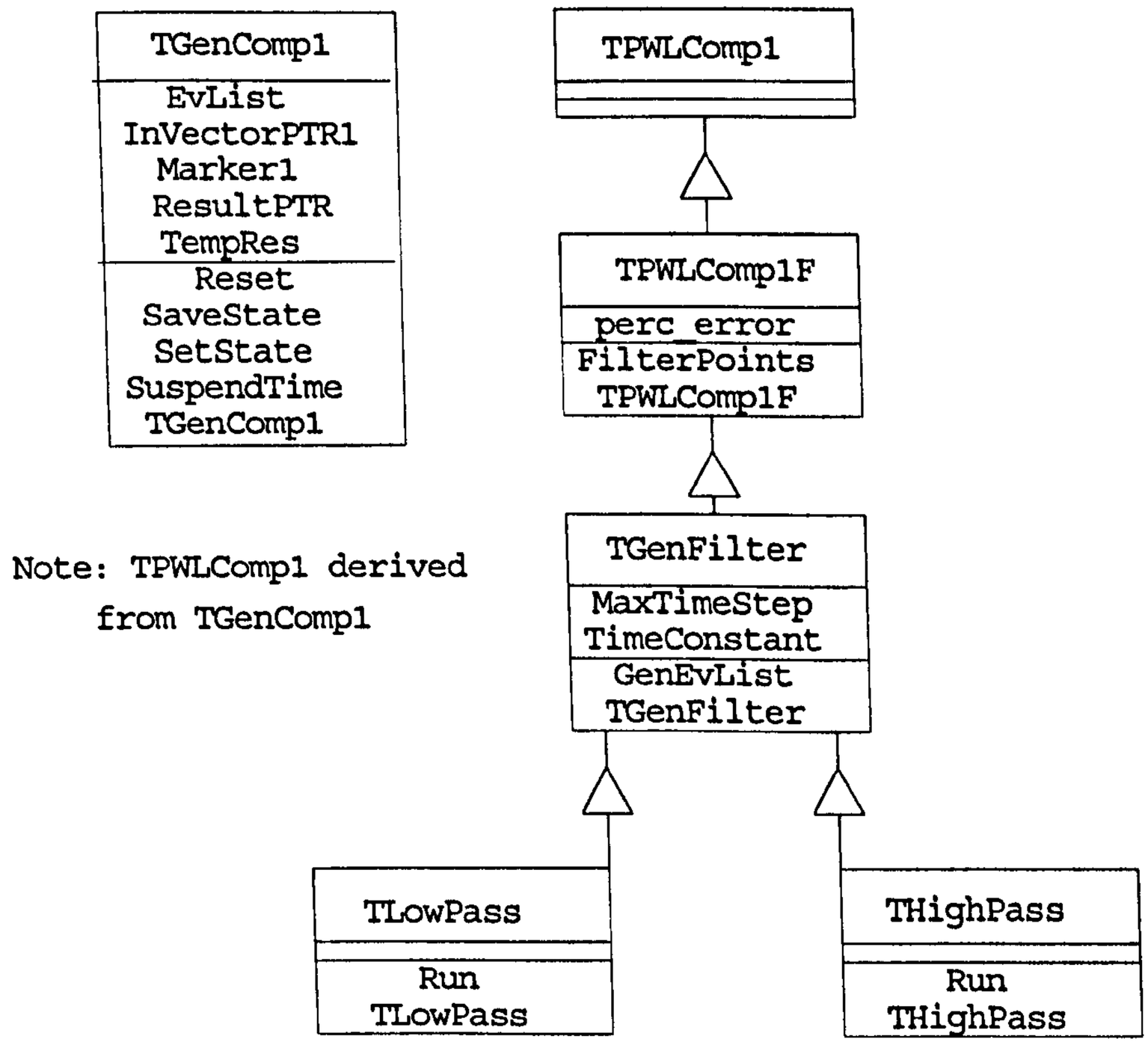


Figure 4-17. Class Hierarchy for Simple Filters.

signal to contain many more points than are necessary to represent its waveform with appropriate degree of accuracy (the representation of the output waveform for this type of model cannot be any more accurate than its input waveform, regardless of the number of points used). The unnecessary points would reduce the efficiency of models that subsequently process the waveform as well as increasing the amount of memory required. To overcome these limitations, the models write their results to a temporary buffer. This buffer is filtered to remove redundant points before it is copied to the output waveform. The structure of the simple low-pass and high-pass filter models is shown by the class hierarchy in Figure 4-17. The *TLowPass* and *THighPass* classes are both derived from the *TGenFilter* class. This defines the *TimeConstant* attribute, the corresponding maximum allowable input time step (*MaxTimeStep*) and the *GenEvList* operation. The *TGenFilter* class is derived from another abstract class: *TPWLComp1F*. This defines the *FilterPoints* operation and so can be used as a base class for any analogue model with a single input waveform that requires filtering of its output waveform.

```
int TLowPass::Run(TimeType FreeRunTime)
{
ResumeTime = last time point written to Results;
if (Status == SUSPEND_MODEL) Status = RUN_MODEL;

while (Status == RUN_MODEL)
        {
        // loop until FreeRunTime or end of data
        if (Input != previous Result)
                {
                GenEvList;    //generates EvList, inserting extra points if req'd.
                Reset TempRes;
                for (all points in EvList)
                        {
                        calculate dvdt and delta;
                        write new point to TempRes;
                        detect and correct overshoot;
                        if ((|TempRes| - |Input|) < |MAX_ERROR|) GoToEnd EvList;
                        }
                Mark end of TempRes;
                FilterPoints;  // writes required points from TempRes to Result
                }
  else
                Result = Input;
 Get next Input;
 if (((Input - ResumTime) > FreeRunTime) || (Input == SuspendData flag))
                {
                Step back to previous Input;
                Results = SuspendData flag;
                Status = SUSPEND_MODEL;
                }
  else if (Input == EndData flag)
                {
                Results = EndData flag;
                Status = END_MODEL;
                }
        }
return Status;
}
```

Figure 4-18. Run Operation for Simple Low Pass Filter Model.

The steps performed in the low pass filter model are illustrated by the pseudo code of its *Run* operation in Figure 4-18. The *GenEvList* operation compares the size of each time step in the input waveform against *MaxTimeStep* (set to *TimeConstant* ÷ 5 for the integrator and *TimeConstant* ÷ 10 for the differentiator during object instantiation). If the time step is less than *MaxTimeStep* then *GenEvList* writes the input point to *EvList* and returns (i.e. *EvList* will only contain a single point). However, if the time step is greater than *MaxTimeStep*, extra points will be generated and inserted into *EvList*. These extra points are inserted at intervals of *MaxTimeStep* and have magnitudes corresponding to the interpolated input signal at these times. Extra steps are inserted until *FreeRunTime* is exceeded or *EvList* is full. If the input signal is changing slowly relative to the time constant its (optimised) points are likely to be spaced at intervals far greater than *MaxTimeStep*. In these cases, the output value will converge to a known state (equal to the input signal magnitude for a low pass filter and zero for a high pass filter) after a period of several time constants so *EvList* can safely be made a finite size. When *EvList* is evaluated, the convergence to the known state is checked to avoid processing of unnecessary points. Generating the extra steps is a simple operation so filling *EvList* with points that are subsequently not used does not make a large impact on the simulation efficiency.

The algorithm used for the integration function is susceptible to overshoot whenever the gradient changes sign. The low pass model therefore checks for possible overshoot (i.e. if the magnitude of the output is larger than the input) and restricts the output value in these cases. This does not occur with the differentiation function.

The *FilterPoints* operation copies *TempRes* to the output waveform, removing any redundant points. It uses error criteria based on the relative signal magnitude so produces reasonable results for most waveforms.

The models of simple analogue components with two inputs are similar to the low pass and high pass filters in so far as they use an input event queue and write results to a temporary buffer that is filtered to remove redundant points before the output waveform is

written. Only two models are required: a summing junction and a multiplier. These are based on the Add and Multiply building blocks respectively.

The class hierarchy of the two-input analogue models is similar to the simple two-input digital models and is shown in Figure 4-19. The *TPWLAComp2* class defines most of the behaviour of both *TAMult* and *TASum* model classes. The *NextEv*, *GenEvList* and *Run* operations are similar to those found in the *TPWLDComp2* class but less complex since waveforms are not required to be converted into digital states. The *Evaluate* operation is invoked during the *Run* operation. It generates and writes points to the *TempRes* buffer from the values in the event queue. The *Evaluate* operation is defined in the *TAMult* and *TASum* models. It is simple since it is only required to multiply or add the magnitudes of two points together. The contents of the *TempRes* buffer are filtered and written to the output waveform by the *FilterPoints* operation. This is the same operation as used with the low-pass and high-pass filter models.

4.4.2 Validation of Simple Models.

The *DCShift* and *Scale* models were tested and found to produce the expected results. However, since these do not correspond to actual components, no comparisons were made

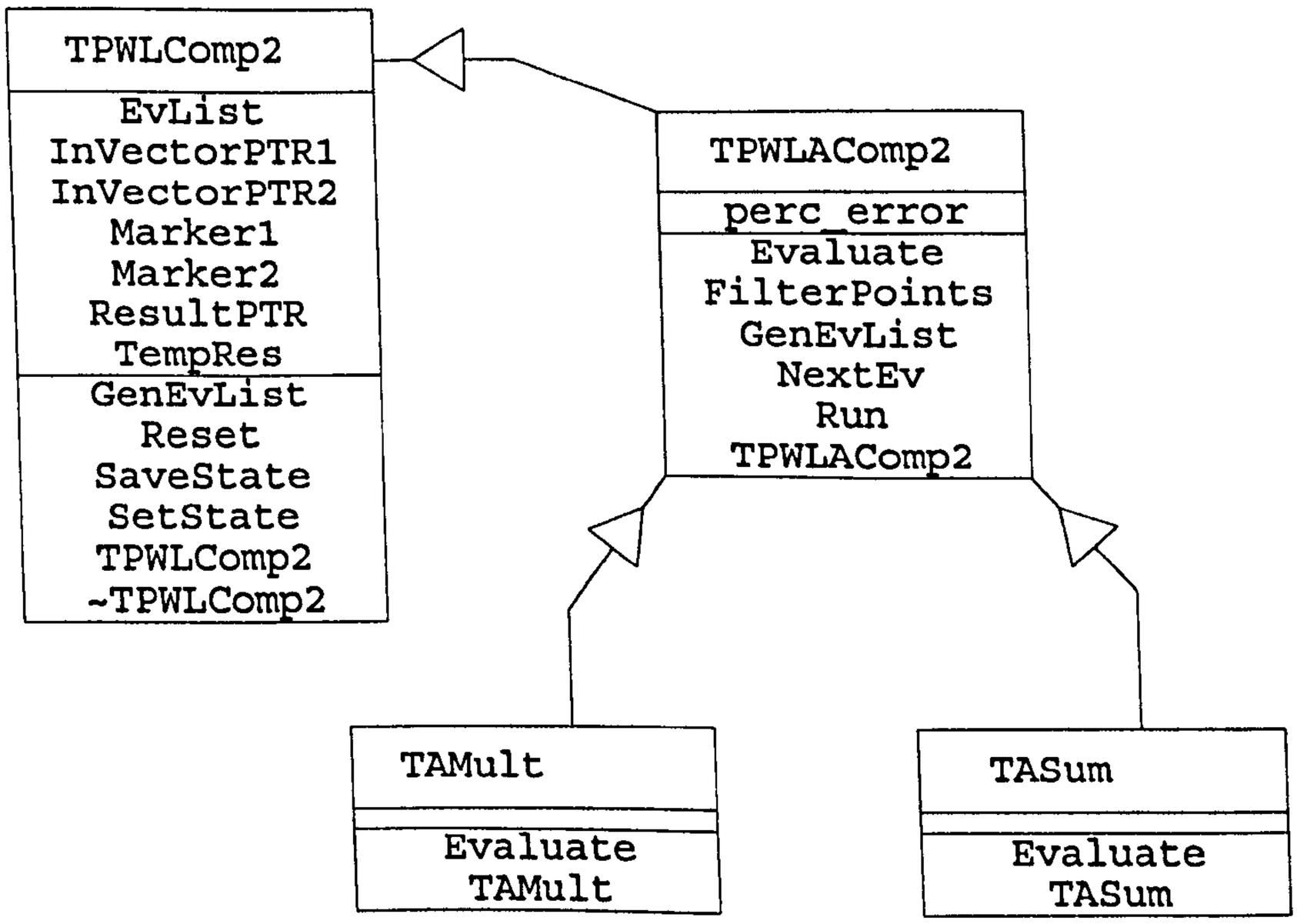


Figure 4-19. Class Hierarchy for 2-Input Analogue Models.

with other simulators at this stage.

The first order low pass model was tested with a variety of input waveforms and the results compared to the simulation of an unloaded passive RC filter in PSPICE. The waveforms generated for simulations with a sinusoidal input and filters with four different time constants are shown in Figure 4-20. This shows close agreement between the two simulators. The waveforms generated by the PWL models have a slightly higher peak magnitude. This is because the PWL approximation to the integrator assumes that the input signal changes to its next level at the start of each time step rather than continuously throughout it. The rate of change of the output signal predicted by the PWL model is therefore slightly over-optimistic. Setting the maximum input time step to a period of one fifth of the time constant (as described in the previous section) kept the errors small.

The PWL model and PSPICE both assume an initial output level of 0 volts. The simulation may therefore need to be performed for some time before the output signal converges to its steady state value (i.e. the value assuming the input has been applied for an infinite length of time prior to the start of the simulation). The PWL simulator and PSPICE both take the same number of cycles to converge to the steady state.

The response of the low pass filter models to a train of pulses is given in Figure 4-21. This also shows close agreement between the PWL model and PSPICE. The over-optimistic rate of change of the PWL model can be seen more clearly with the pulse waveform. It causes the output to reach the steady state slightly earlier.

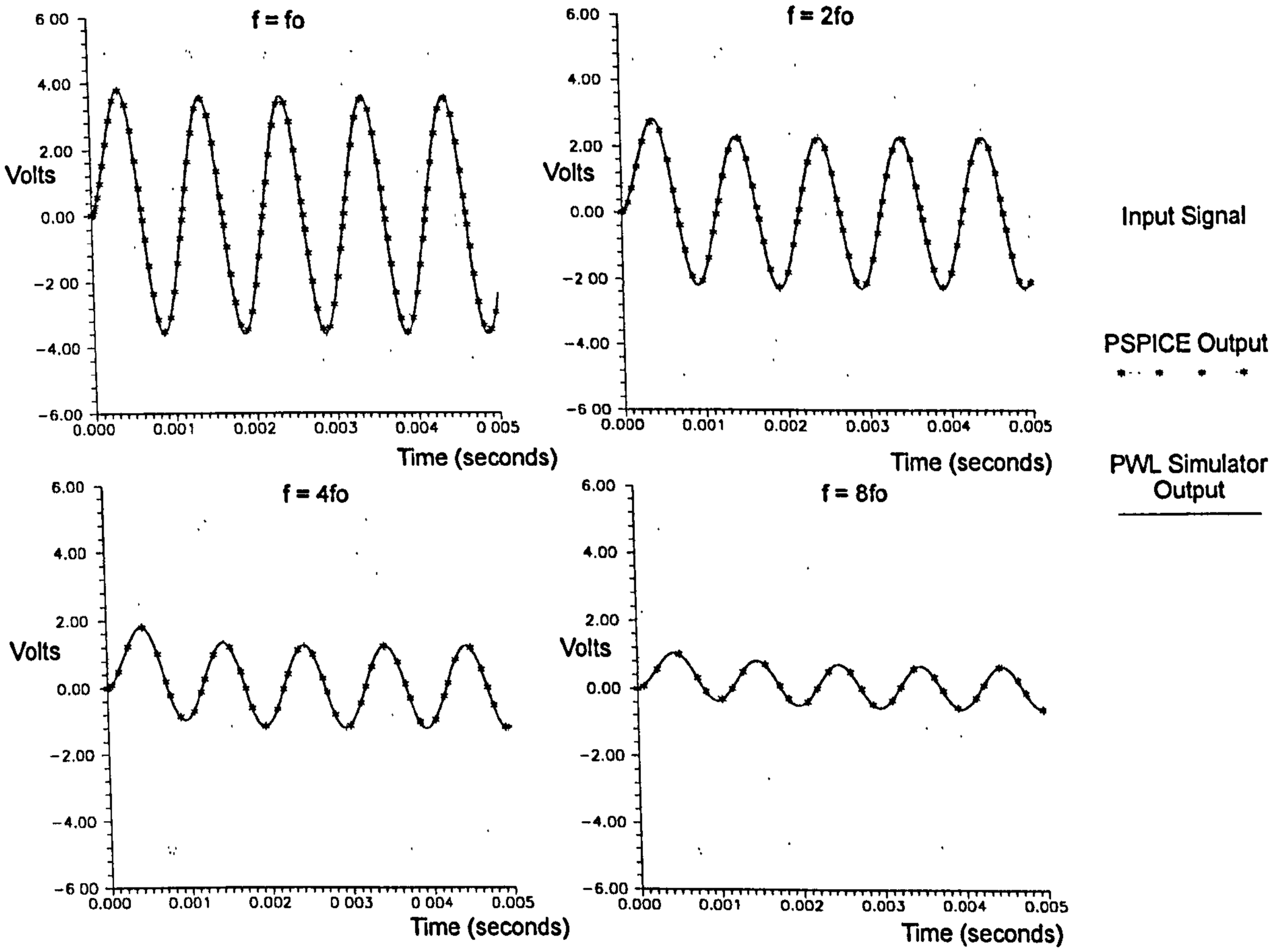


Figure 4-20. Results of Low Pass Filter Simulation in PSPICE and PWL Simulator.

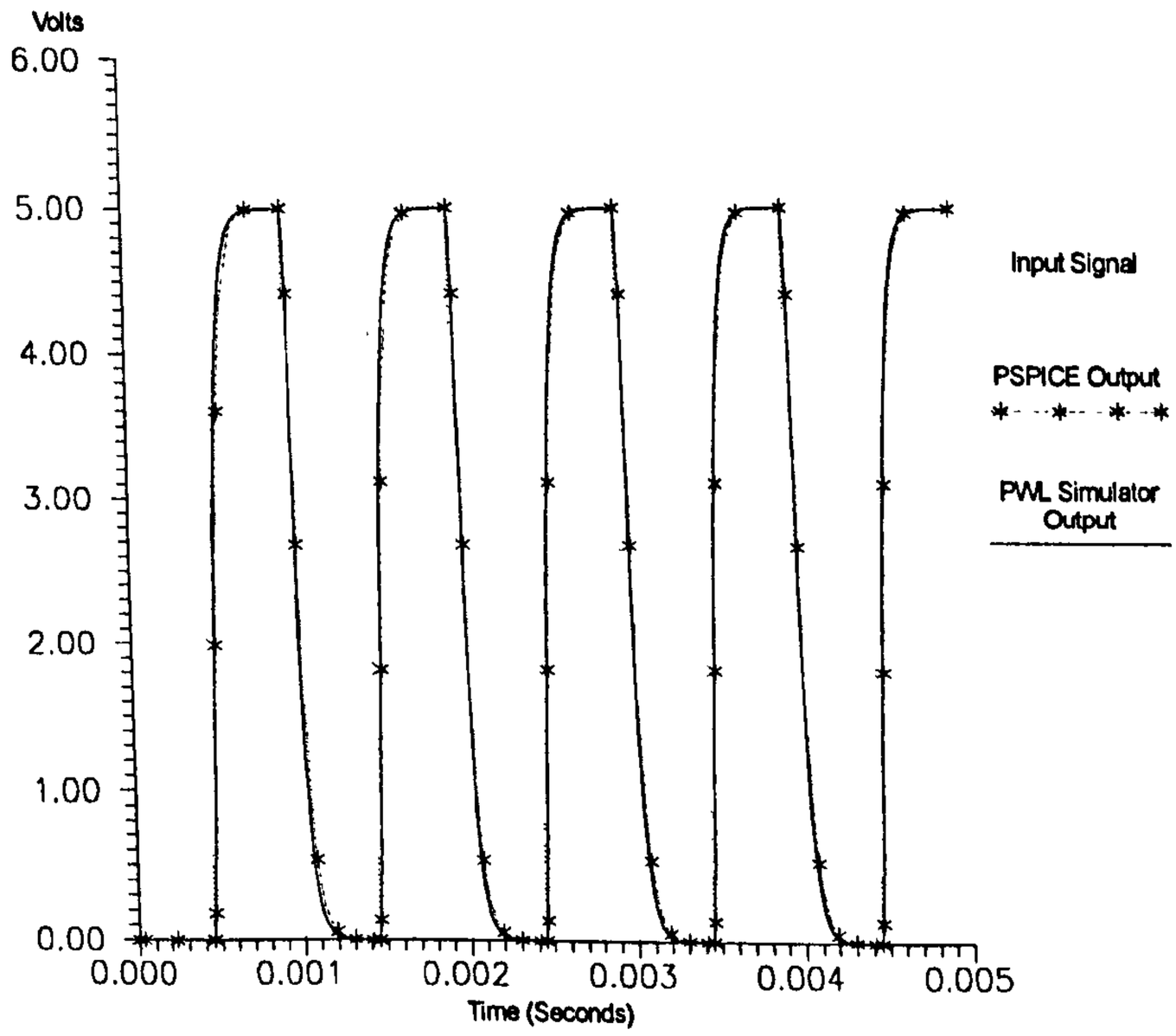


Figure 4-21. Response of Low Pass Filter to Digital Input Waveform for PSPICE and PWL Simulator.

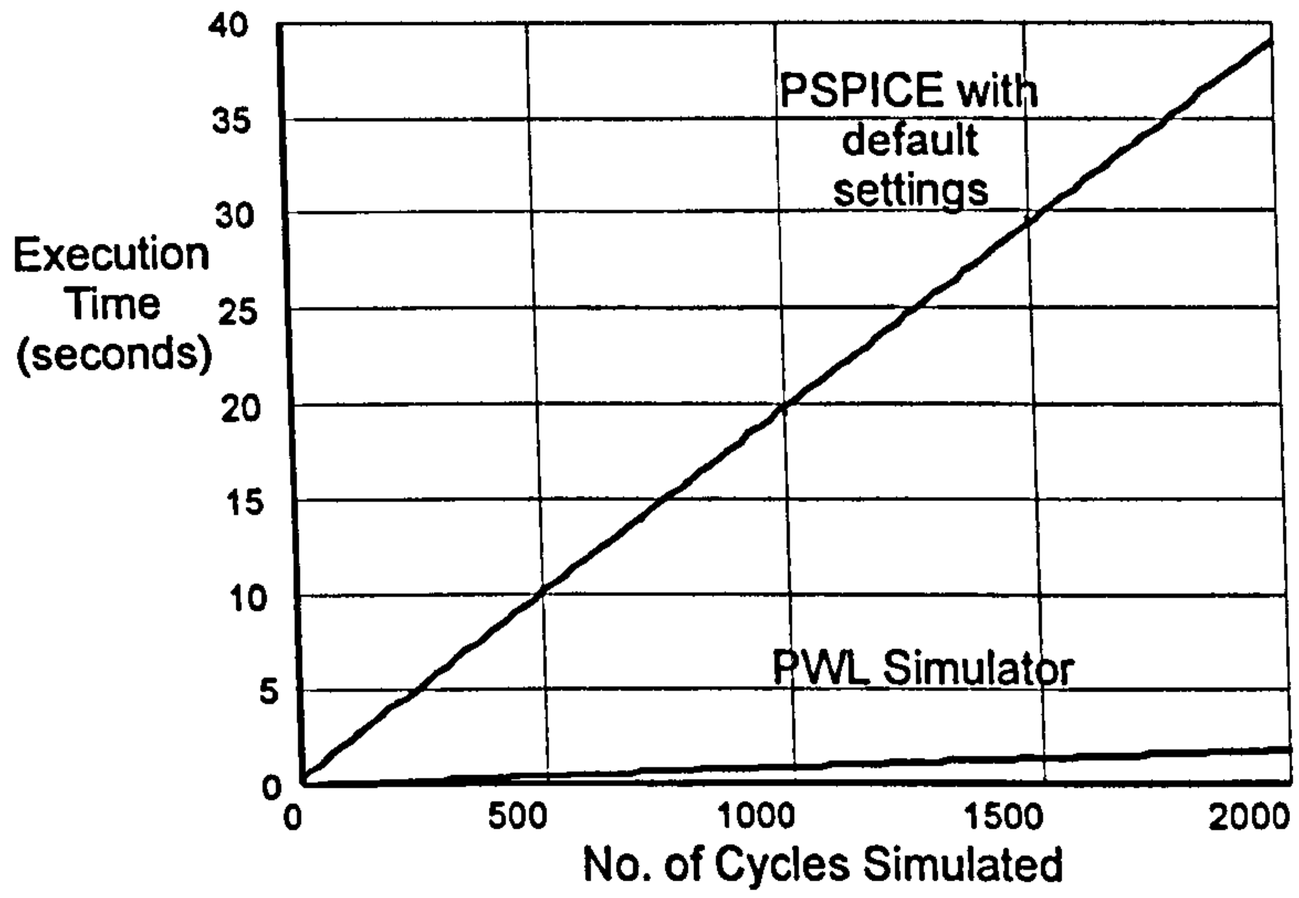


Figure 4-22. Lowpass Model Performance.

To test the performance of POISE against PSPICE, each model was driven by a sinewave and simulated for a large number of cycles. POISE used input waveforms with a relative error of less than 0.1%. The output waveform filter was also set to 0.1%. PSPICE used the default settings with the simulator output waveform file disabled. The simulation execution times were recorded and are plotted in Figure 4-22. This shows that the execution time increases approximately linearly with the number of cycles. The execution time of PSPICE increases by approximately 20ms per waveform cycle while for the PWL simulator the increase is less than 1ms per cycle. Repeating the PSPICE simulation with the output enabled revealed gross distortion of the sinusoid. This is a result of the time step control mechanism in PSPICE which sets the minimum time step to a fraction of the total simulation time. The waveform distortion was reduced by reducing the RELTOL parameter from its default value of 0.001 to 0.00001. A portion of the PSPICE output waveform with the reduced distortion is given in Figure 4-23. A PWL output waveform with a maximum relative error of 1% is also shown for comparison. The PSPICE simulation (with the output file disabled) took 275 seconds for 2000 cycles. This is approximately 150 times longer than a PWL simulation with 0.1% maximum relative errors.

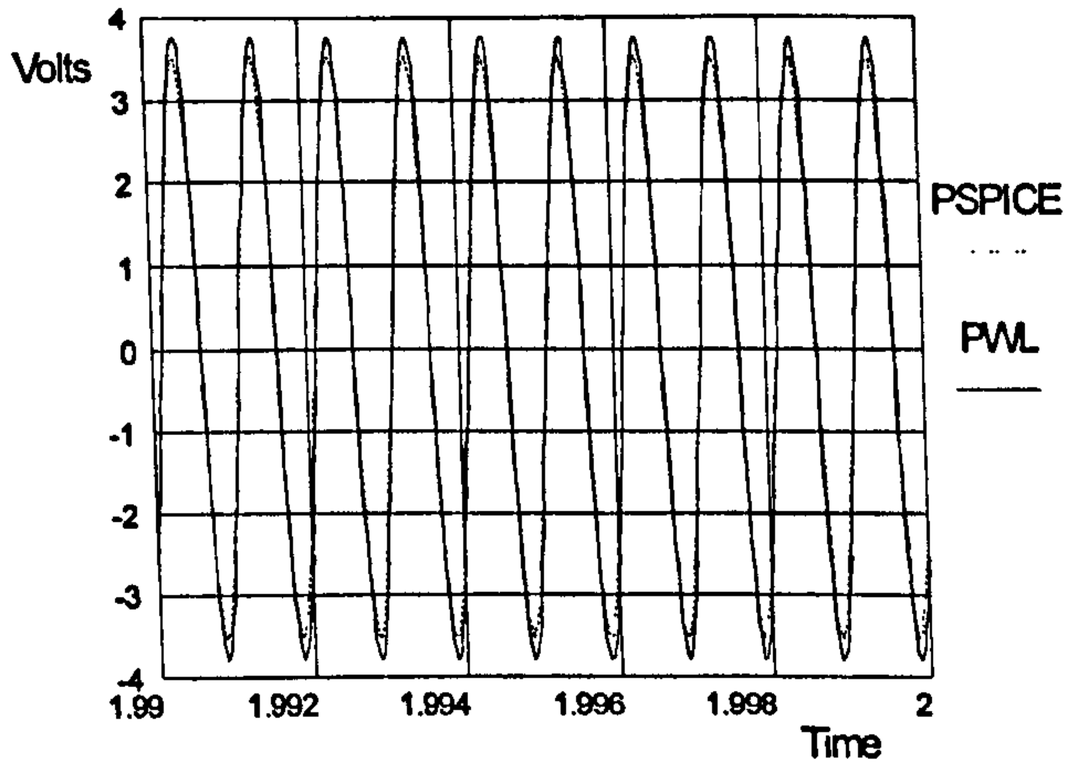


Figure 4-23. Waveform Distortion.

The high pass filter model was tested against an unloaded passive RC filter in PSPICE using a variety of input waveforms. Results for a sinusoidal input waveform are given in Figure 4-24 and for a series of pulses in Figure 4-25. These show close agreement between the PWL simulator (POISE) and PSPICE. The PWL high pass filter predicts a rate of change that is slightly higher than it should be (for the same reason as the low pass filter) and so also over-estimates the peak magnitudes.

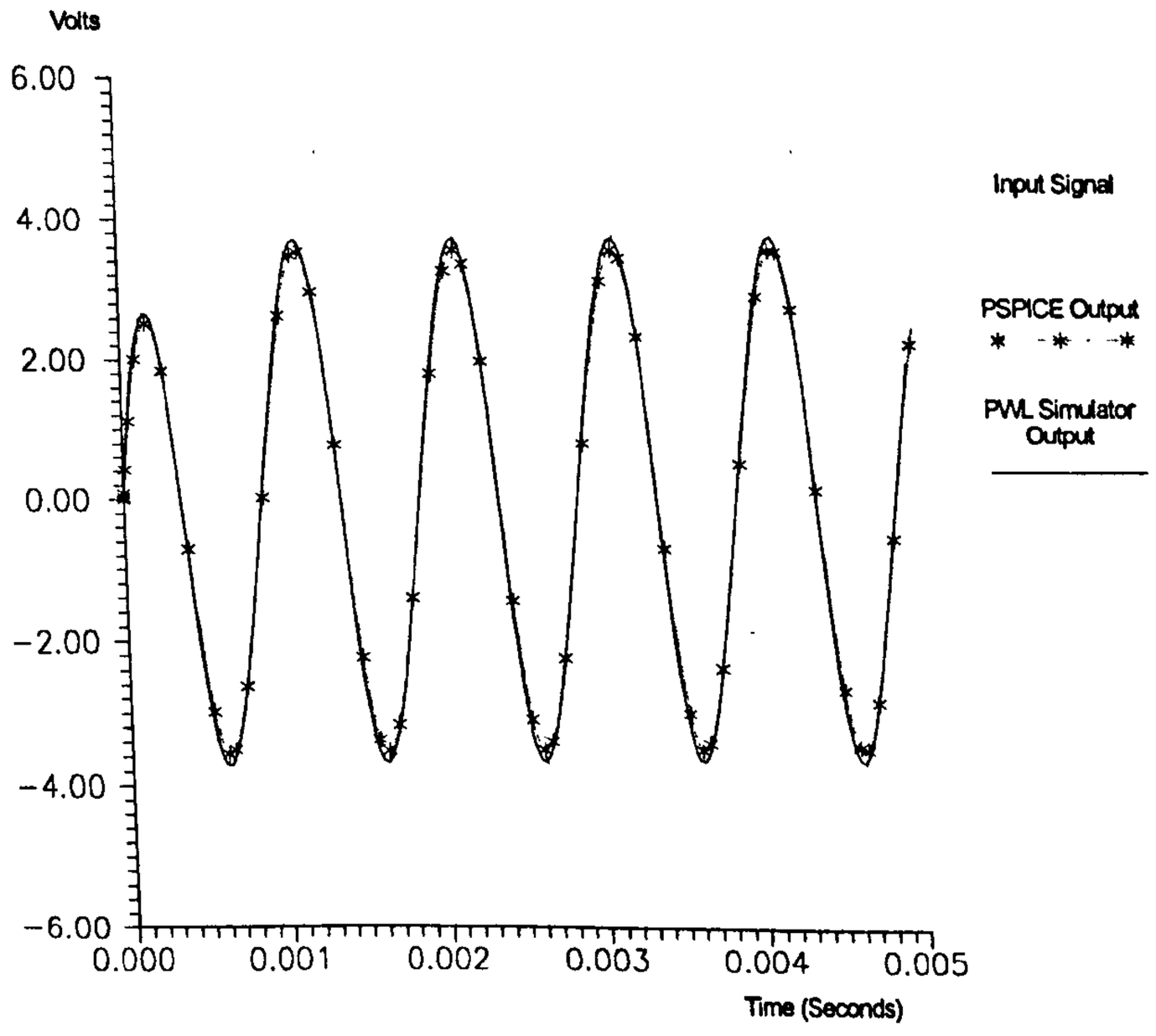


Figure 4-24. Response of High Pass Filter to Sinusoidal Input Waveform.

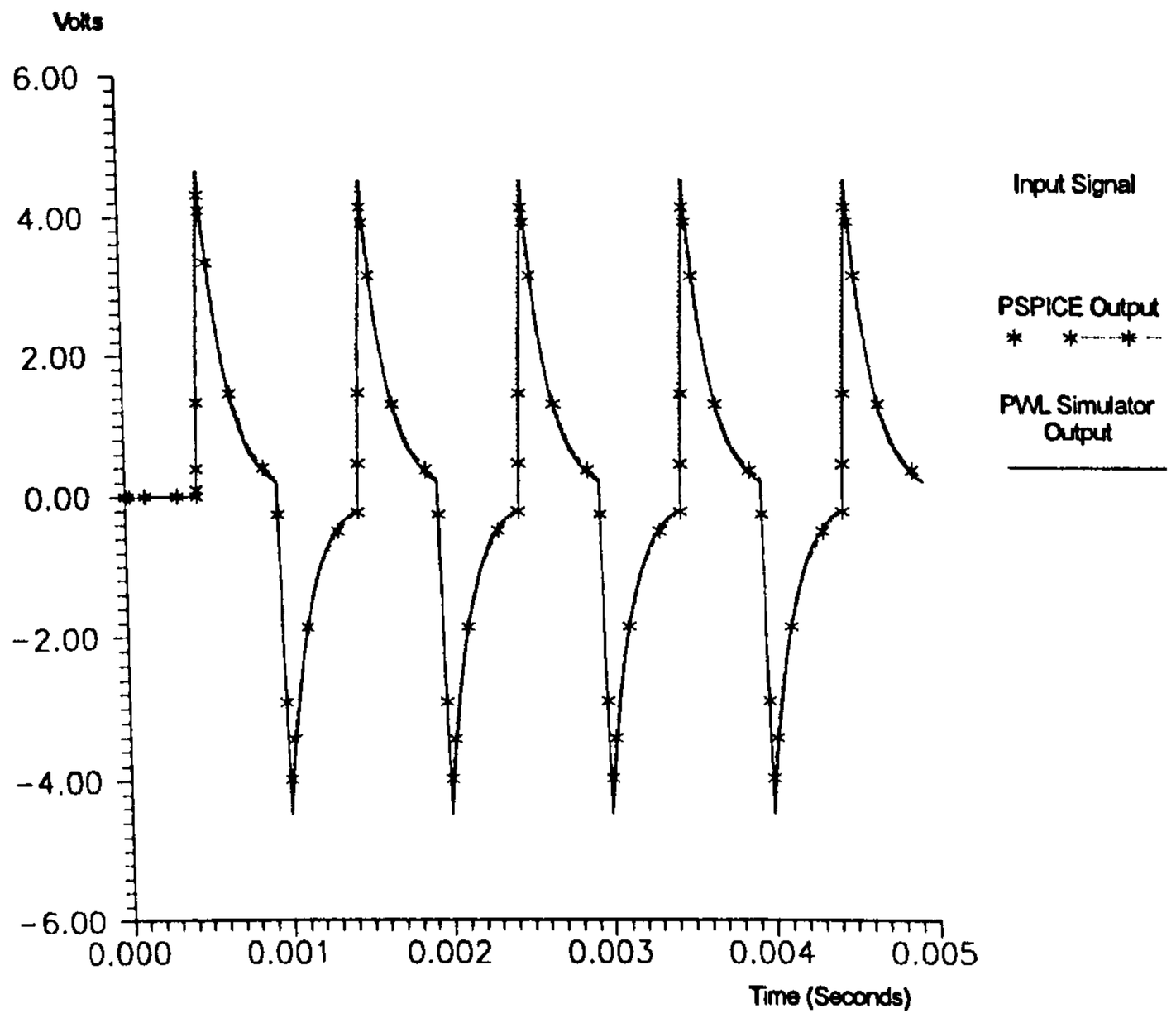


Figure 4-25. Response of High Pass Filter to Square Wave.

The performance of the PWL high pass filter model is almost identical to the PWL low pass filter model. This is expected due to the similarity in the structure and algorithms used in each model.

The multiplier and summing-junction models were tested with a variety of input waveforms and found to produce the expected results. Since both process two independently varying waveforms there is a likelihood of one input waveform ending before the other. There are three possible approaches that could be taken when this situation is detected:

1. the output waveform could be terminated and any unprocessed points on the other waveform ignored;
2. the terminated waveform could be assumed to remain at its final value for the remainder of the simulation;
3. the final segment of the terminated waveform could be extended to the end of the simulation by assuming a constant rate of change.

Approach 2 is used for the digital models. For consistency, it was decided to also use approach 2 with the analogue models. These models do not have primitive counterparts in SPICE-type simulators. No comparisons of the model performance were therefore made at this stage.

4.4.3 Consideration of Circuits with Feedback.

Feedback is used extensively in analogue circuits. The presence of feedback signals can invalidate assumptions that have been made about the state of signals in the forward path. The inherent delays that occur with digital models enable event-driven approaches to generate the correct responses. Analogue models are assumed to change state continuously, with no time delays between inputs and outputs. This requires the magnitude of all signals around a feedback path to be determined for each input waveform time step. SPICE-type simulators overcome this problem by simultaneously solving every circuit equation at each time step. This cannot be done with the proposed PWL modelling approach since the models are assumed to be unidirectional and hence must be evaluated in a particular order. Alternative approaches to determine the state of signals around feedback paths were investigated.

The classical representation of a feedback system is shown in Figure 4-26 where A and B represent signal transfer functions. The magnitude of the output signal *y(t)* for any input signal *x(t)* is given by:

$$y(t) = \left(\frac{A}{1 + AB}\right) \cdot x(t) \qquad \textbf{Equation 4-1}$$

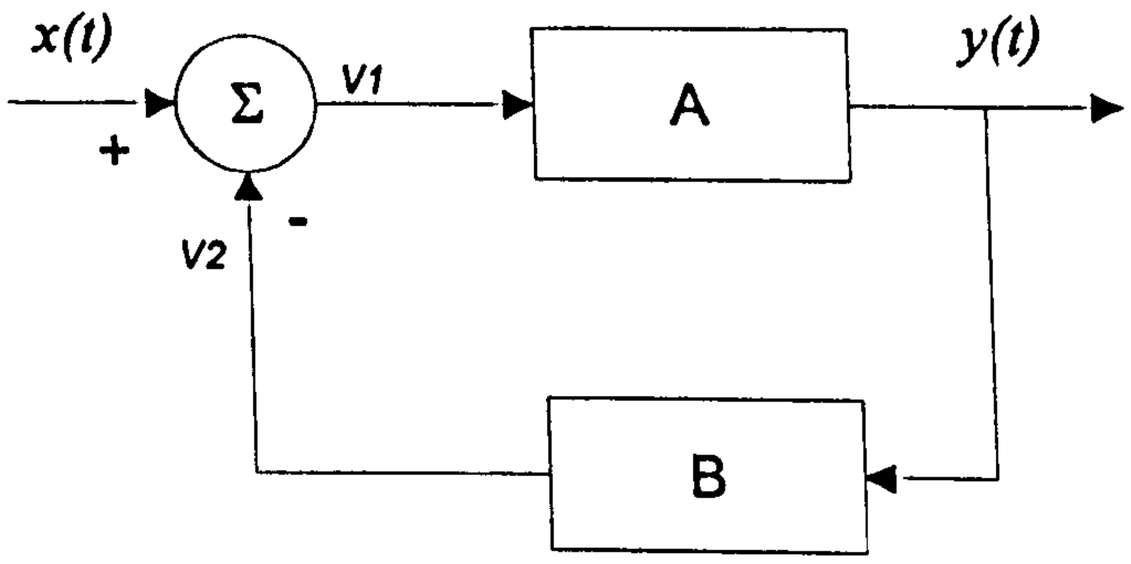


Figure 4-26. Classic Feedback System.

where the *AB* product term is known as the "loop gain". Equation 4-1 cannot be used directly to model the behaviour if A and B contain frequency dependent terms. It is therefore usually solved using the Laplace operator *s*. Simulators that facilitate transfer function modelling such as SMASH can model a circuit with feedback using Equation 4-1 if the circuit can be represented in the form of Figure 4-26. Unfortunately, obtaining correct expressions for A and B can itself be a complex operation. This technique is not well-suited to mixed-signal simulation using PWL waveforms: the waveforms would have to be converted to the frequency domain before they could be evaluated and the results converted back to the time domain for use by subsequent inputs.

It is possible to simulate a circuit with the structure of Figure 4-26 entirely in the time domain if an iterative approach is used. A simulation method using this approach was developed. The algorithm is shown in pseudo code in Figure 4-27. It starts by calculating the output level assuming no feedback signal (*Y_Target*). This is compared with the present output value (*Y_Inst*) and if different, the output value is increased or reduced by a fraction of the difference (set by *Iterate_Factor*). The feedback signal is then calculated from the new output value. This is subtracted from the input signal to form the input for the next iteration. This algorithm generates correct results for systems with small forward gains. Unfortunately, it was found that the iterations for systems with large forward gains would only converge if the output steps were small. Algebraic analysis of the algorithm showed that the iterations would only converge when:

```
FOR each point in input waveform
        {
        V1 = X;
        WHILE (Iteration < Max_Allowed_Iteration)
                {
                Y_Target = V1 * A;
                Output_Step = (Y_Target -Y_Inst) / Iterate_Factor;
                Y_Inst += Output_Step;
                V2 = Y_Inst * B;
                V1 = X - V2;
                }
        }
```

Figure 4-27. Pseudo Code for Iterative Simulation of Feedback Circuit.

$$(A \times B) < ((2 \times Iterate\_Factor) - 1) \qquad \textbf{Equation 4-2}$$

This convergence condition applies regardless of the closeness of the initial approximation to the correct output.

Many amplifiers used in mixed analogue/digital ASICs have very large forward gains ($>10^3$). This iterative approach is not suitable for such devices due to the large number of iterations that would be required.

4.4.4 Models of Operational Amplifier Circuits.

It is common to use op-amp circuits to implement feedback amplifiers. In many applications, the op-amp can be assumed to be a good approximation of an ideal device. One of the simplest op-amp circuits is the inverting amplifier shown in Figure 4-28. The analysis of this circuit is simple if the op-amp is ideal:

- V_{in} causes a current I_{R1} to flow in resistor $R1$. The magnitude of this current is given by

$$I_{R1} = (V_{in} - V^-) / R1 \qquad \textbf{Equation 4-3}$$

- No current flows into the inverting terminal so $I_F = I_{R1}$
- V_{out} is the voltage dropped across $R2$ by current I_F and is given by

$$(V^- - V_{out}) = I_F \cdot R2 \qquad \textbf{Equation 4-4}$$

- The inverting input is a "virtual earth" so $V = 0$ volts, simplifying the equations for I_{R1} and V_{out}.

If the circuit in Figure 4-28 is analysed by considering the current that flows around the

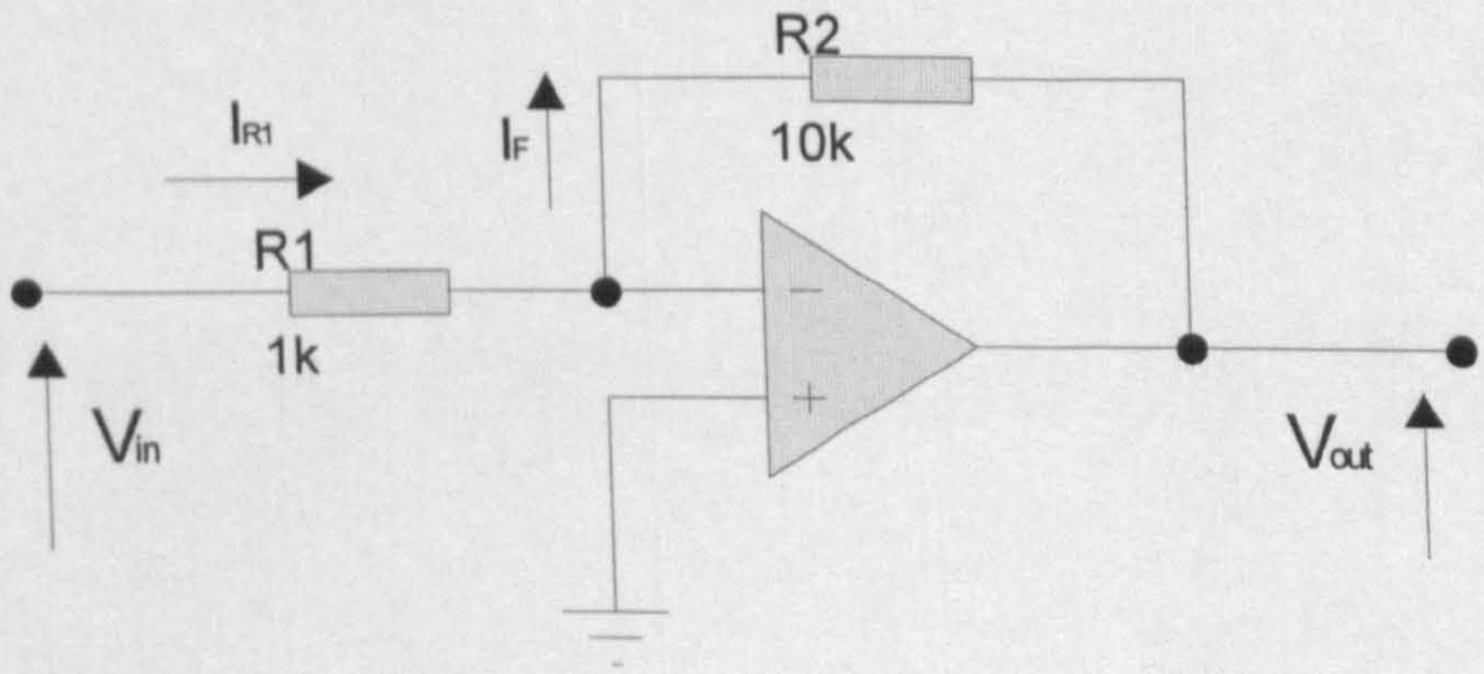


Figure 4-28. Inverting Amplifier using an Op-Amp.

feedback path, the output voltage can be determined directly from the input voltage: an iterative approach is not required. A PWL model was constructed for this circuit. It required two new component models to be created:

1. A resistor that generated a PWL current output waveform from a PWL voltage input waveform assuming the output terminal was connected to a reference voltage source (*TResistorVI*).
2. A resistor that generated a PWL voltage output waveform from a PWL current input waveform assuming the input terminal was connected to a reference voltage source (*TResistorIV*).

The first resistor model was used to represent *R1* whilst the second represented *R2*. The reference voltage was taken to be 0 volts. Since the op-amp was assumed to be ideal, no additional components were required to model its effect on the circuit. The PWL model produced correct results when simulated (i.e. $V_{out} = -\frac{R2}{R1} \cdot V_{in}$).

In many cases, it is desirable for op-amp models to include some of the effects resulting from the limitations of practical op-amps with non-ideal behaviour. Typical effects include input offset voltage, limited output voltage swing and limited output voltage slew rate. The ideal op-amp model can easily be modified to take account of these effects. A model of the inverting amplifier with non-ideal behaviour is shown in Figure 4-29. A new class (*TOpAmp*) representing op-amp models with this structure was derived from the

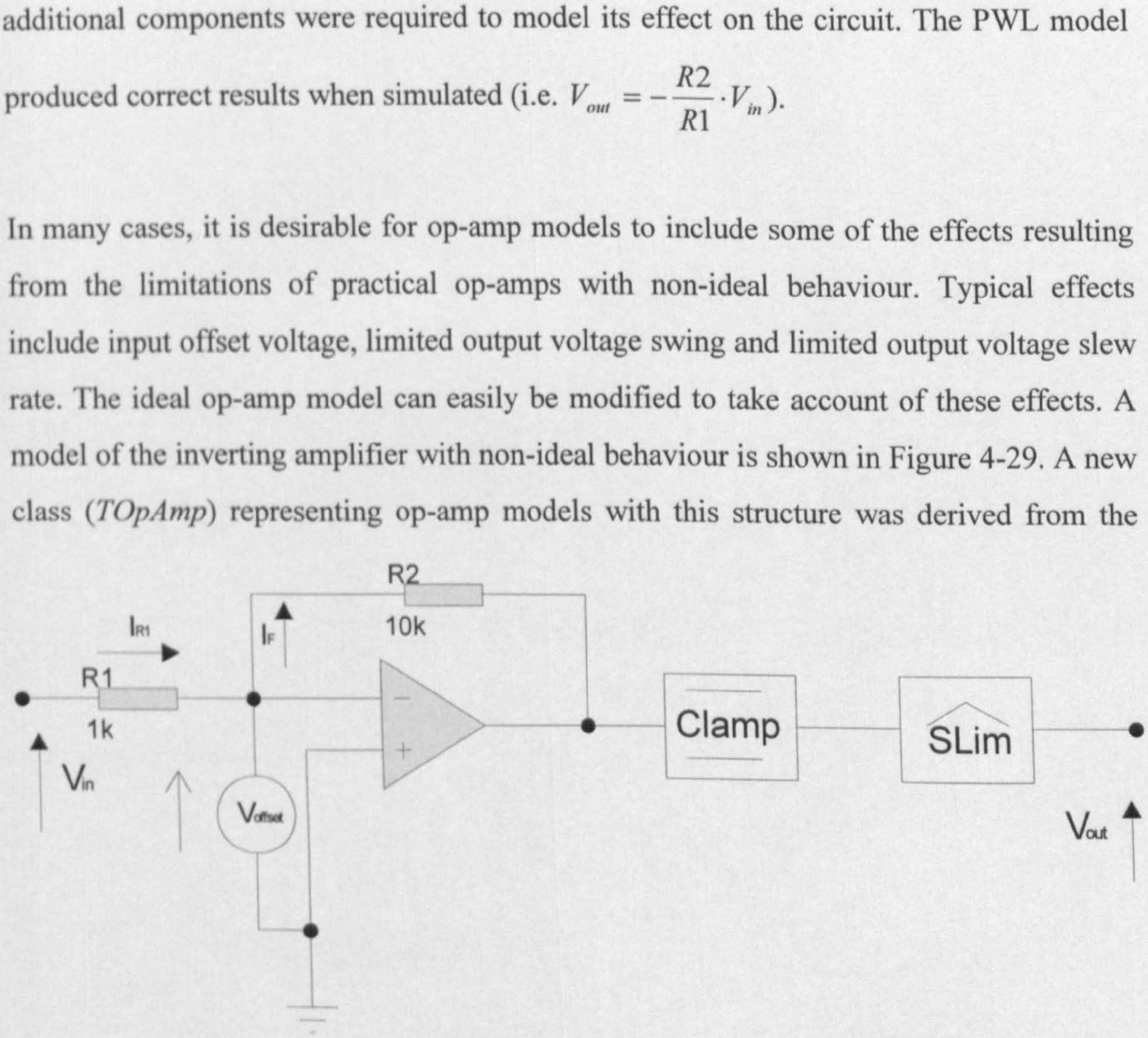


Figure 4-29. Inverting Amplifier with Non-Ideal Op-Amp.

TFeedbackAmp class discussed in section 4.2.2. It adds attributes to point to the clamp and slew rate limiting models, creates storage for the intermediate results and redefines the *run* operation. The input offset voltage is modelled by a voltage source between the inverting and non-inverting terminals. Its effect is to slightly reduce the voltage dropped across *R1* so reducing I_{R1} (and hence altering V_{out}). The clamp component compares the output level with the maximum possible positive and negative output voltage swings. It prevents *Vout* from exceeding these levels. The *Slew* block (*SLim*) monitors the rate of change of the output signal and limits its gradient. The clamp and slew components are each implemented by classes derived from the *TPWLComp1* class. They each include a flag that is used to store their operating region (NOT_LIMIT, POS_LIMIT, NEG_LIMIT). Interpolation of the input waveforms is required to determine when the boundary between two operating regions is crossed.

A model of a non-ideal inverting op-amp (*TInvOpAmp*) was derived from the *TOpAmp* class. Simulation results for this model with the parameters set to those of a uA741 op-amp are given in Figure 4-30. This shows the effect of slew-rate distortion on a sine wave. The results for the PWL model and PSPICE are in close agreement.

The model in Figure 4-29 doesn't include the effect of a limited bandwidth. This can be

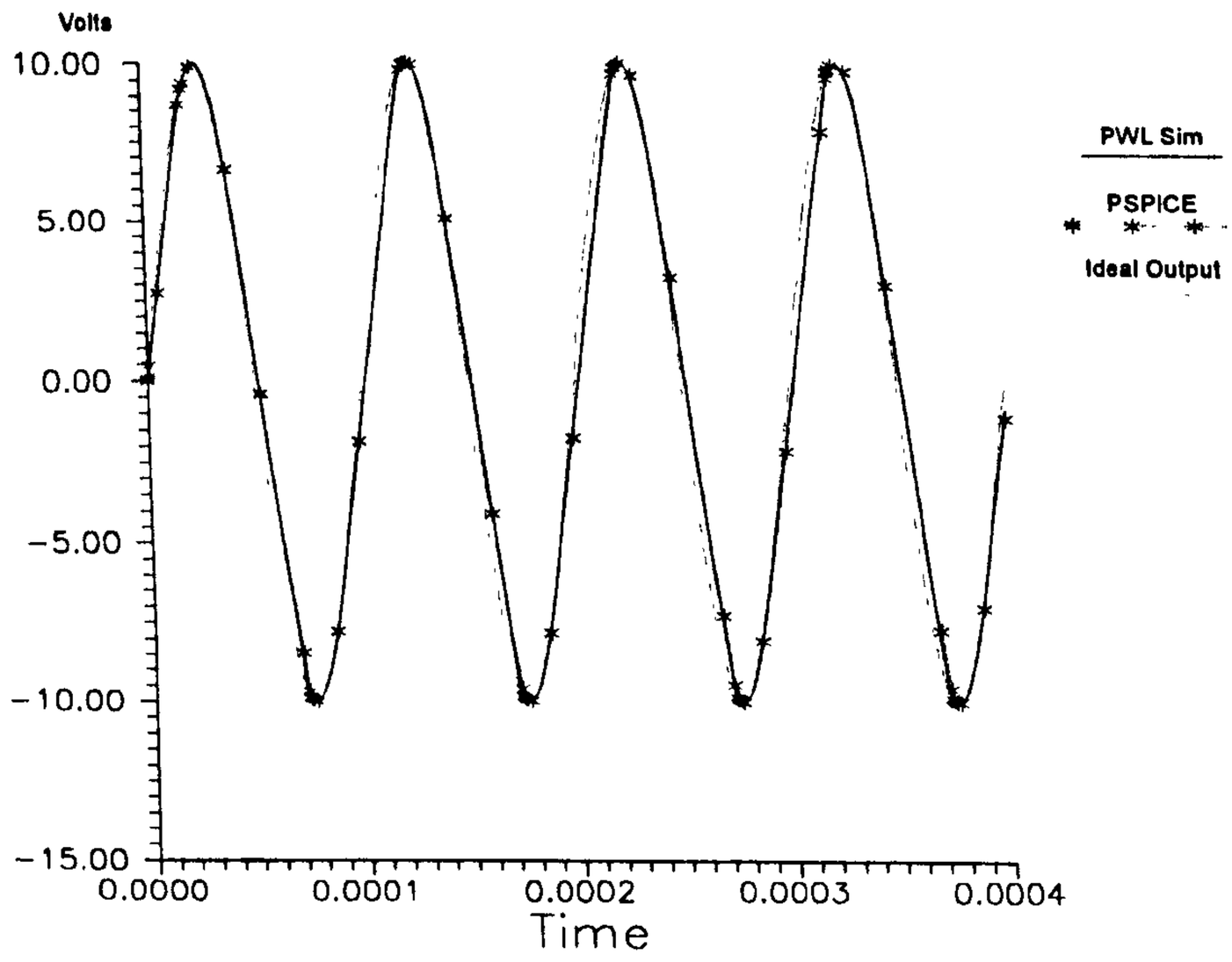


Figure 4-30. Simulation of Non-Ideal Op Amp.

modelled by a single pole low pass filter at the amplifier output (representing the stabilising capacitor between the output and intermediate stages) [55]. Unfortunately, the cut-off frequency of this filter depends on the amplifier gain, preventing the creation of a universal PWL model. This can be overcome if the feedback resistor is replaced by a passive integrator (low pass filter) as shown in Figure 4-31. At frequencies well below the filter cut-off frequency the circuit behaves the same as Figure 4-29 with a gain of *-RF/R1*. At frequencies above the cut-off frequency, the gain is reduced by 6dB per octave by the low pass filter, independently of any subsequent limiting by the *Clamp* and *SLim* elements.

The low pass filter model differs from the passive integrator discussed previously since it has a current input signal and produces a voltage output signal. The closed-loop bandwidth is given by:

$$B_{CL} = \frac{B}{1+\left|A_{CL}(ideal)\right|} = \frac{B}{1+\dfrac{R_F}{R_1}} = \frac{BR_1}{R_1+R_F} \qquad \textbf{Equation 4-5}$$

where B is the unity gain bandwidth (1MHz for a uA741). The cut-off frequency of the low pass filter is given by:

$$f_o = \frac{1}{2\pi R_F C_F} \qquad \textbf{Equation 4-6}$$

The value of *CF* is set automatically when the low pass model is instantiated to produce

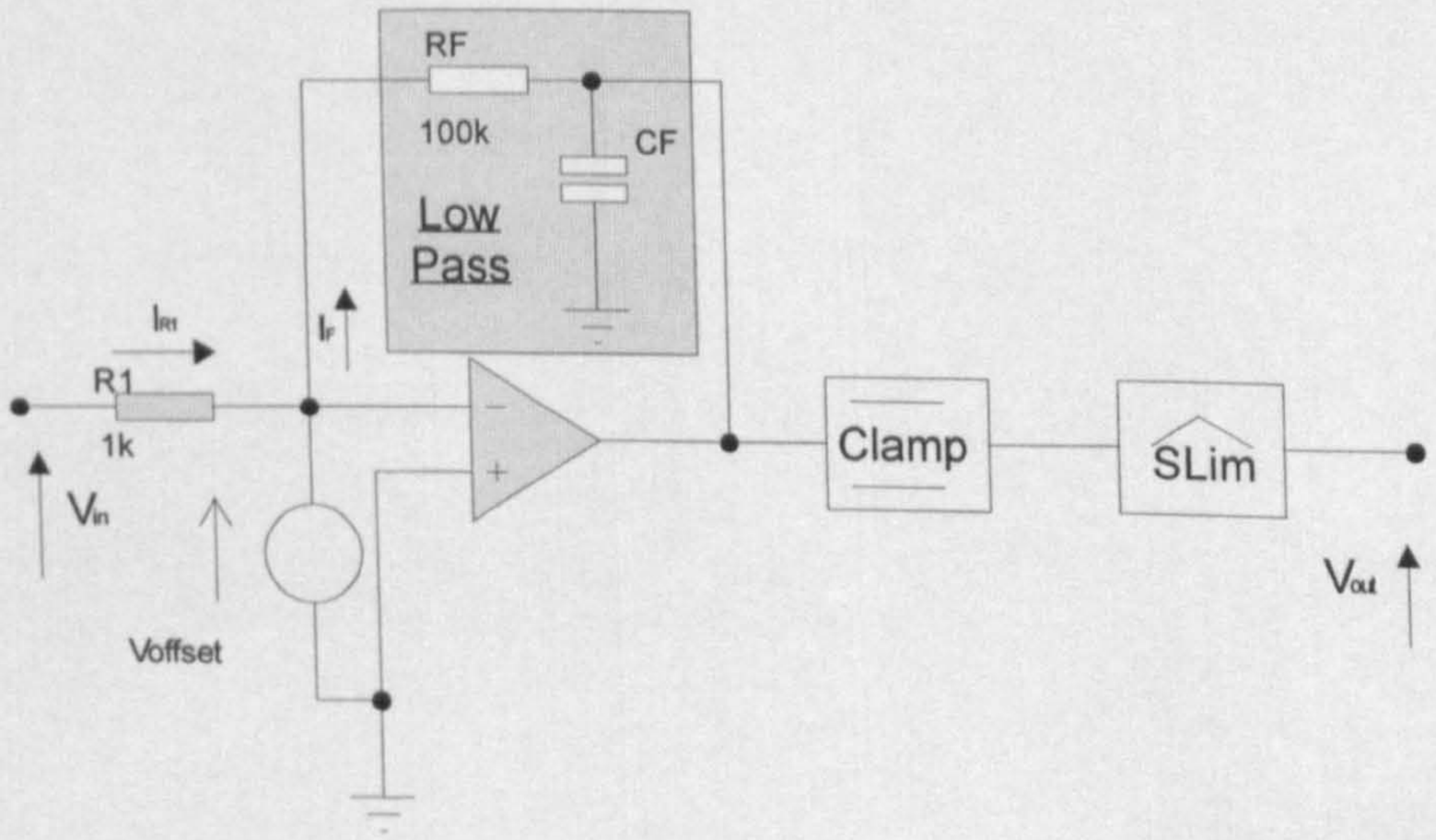


Figure 4-31. Inverting Op Amp Model with Limited Bandwidth.

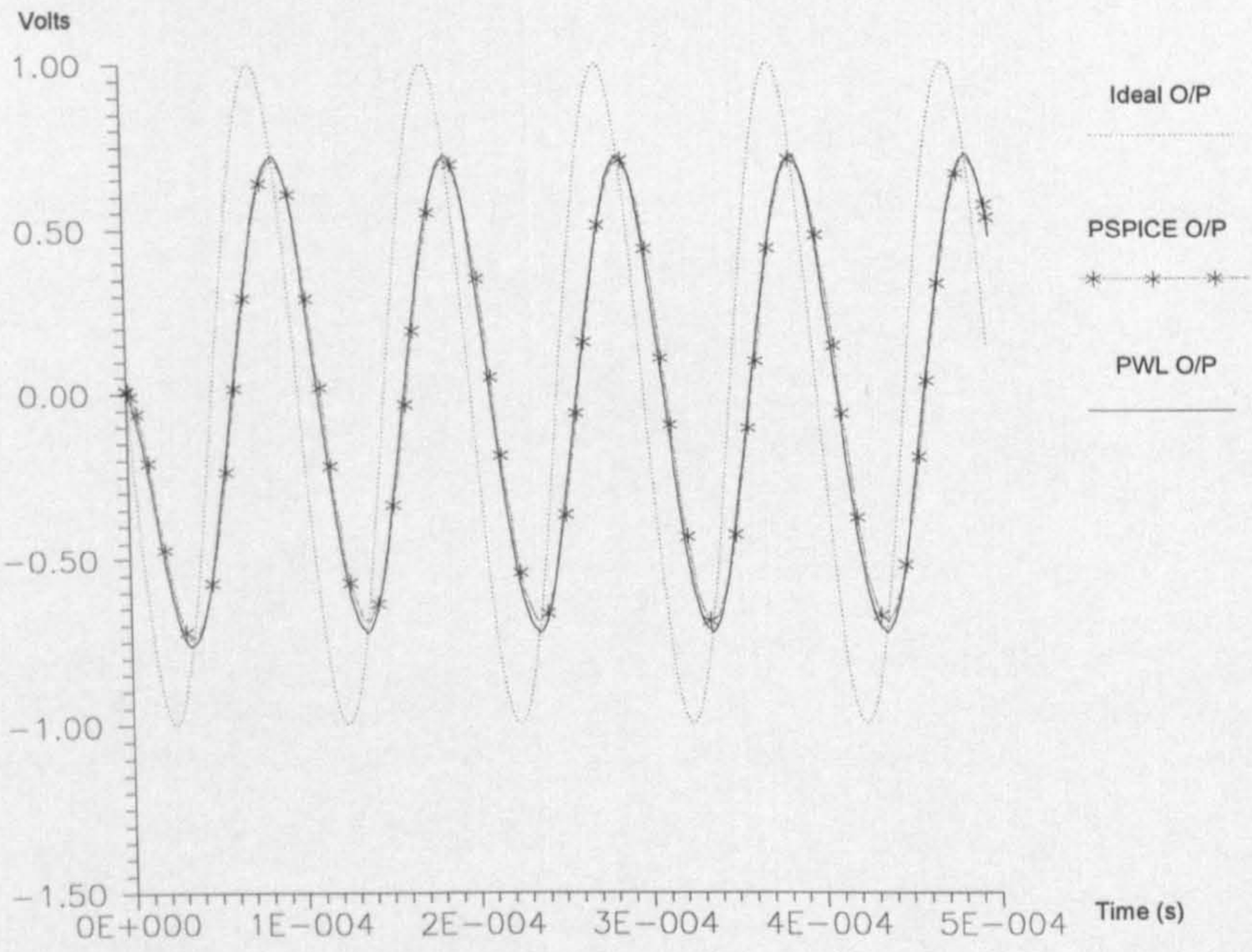


Figure 4-32. Effect of Limited Bandwidth on Inverting Amplifier.

the appropriate bandwidth for the given value of *R1*. It is calculated from:

$$C_F = \frac{1}{2\pi B \times \dfrac{R_1 R_F}{R_1 + R_F}} \qquad \textbf{Equation 4-7}$$

Simulation results for the circuit in Figure 4-31 are given in Figure 4-32 together with results from a PSPICE simulation and an ideal PWL amplifier model. This shows close agreement between the results for the non-ideal PWL model and PSPICE.

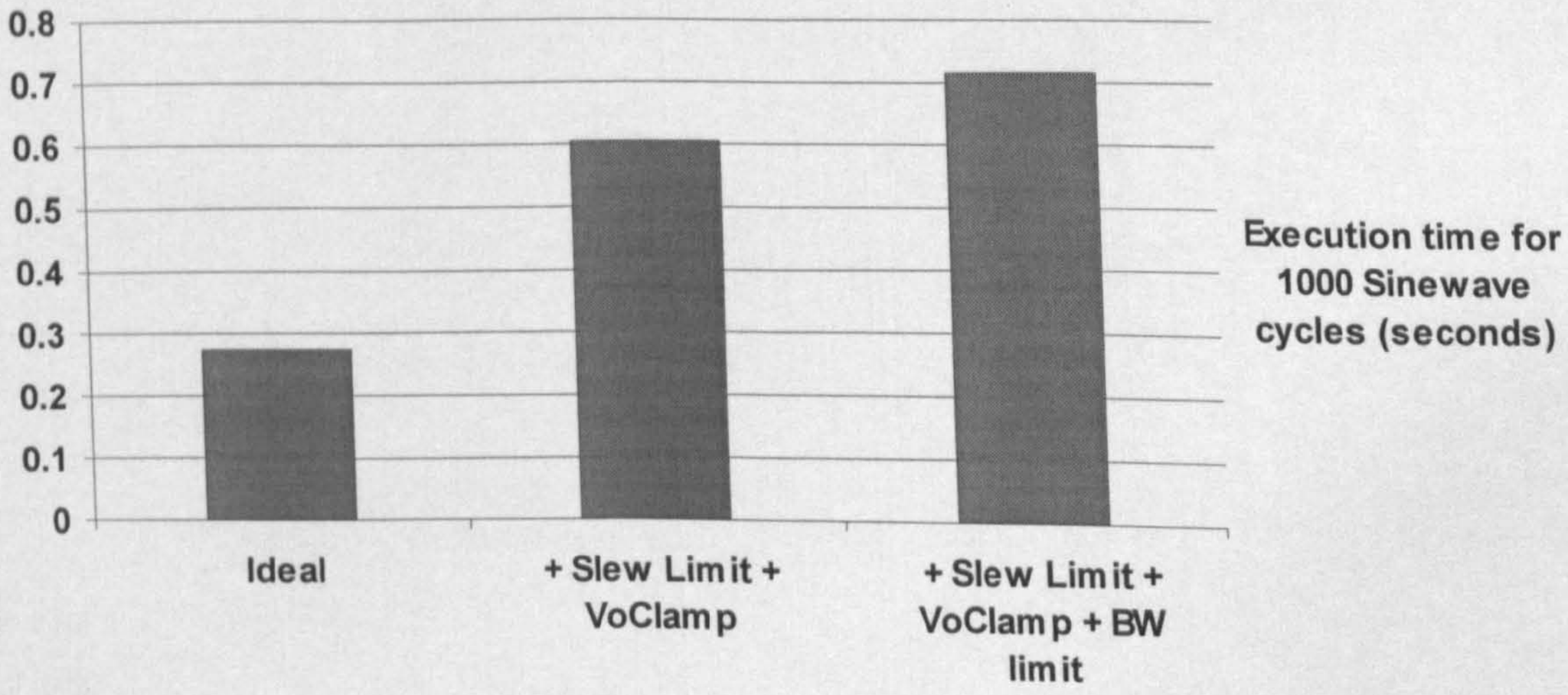
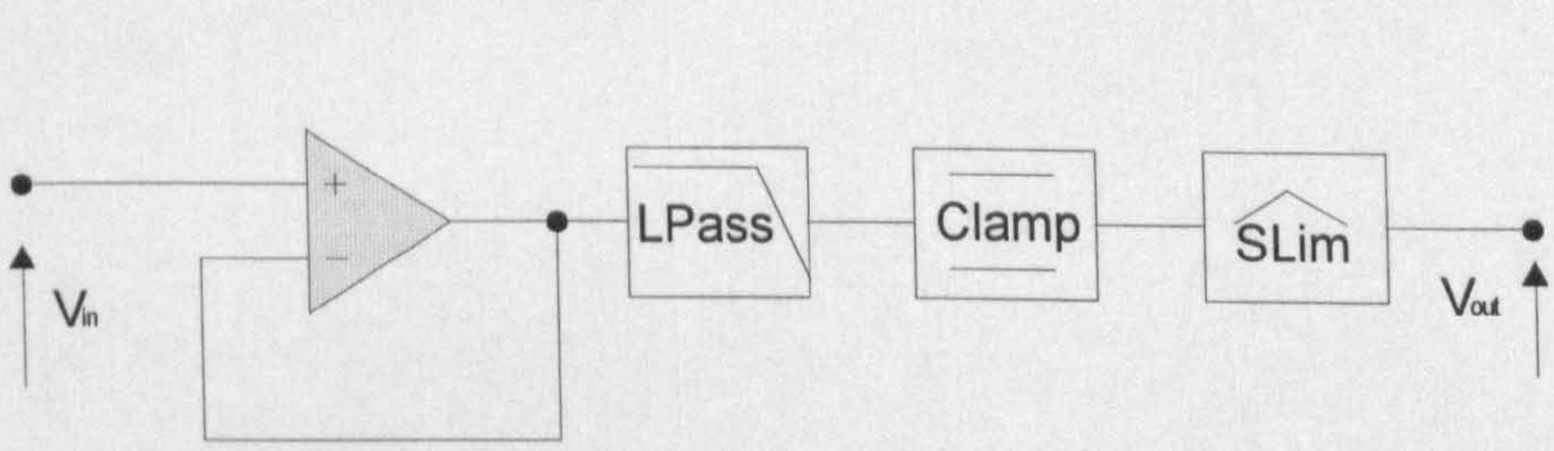


Figure 4-33. Simulation Times for PWL Inverting Op-Amp Models.

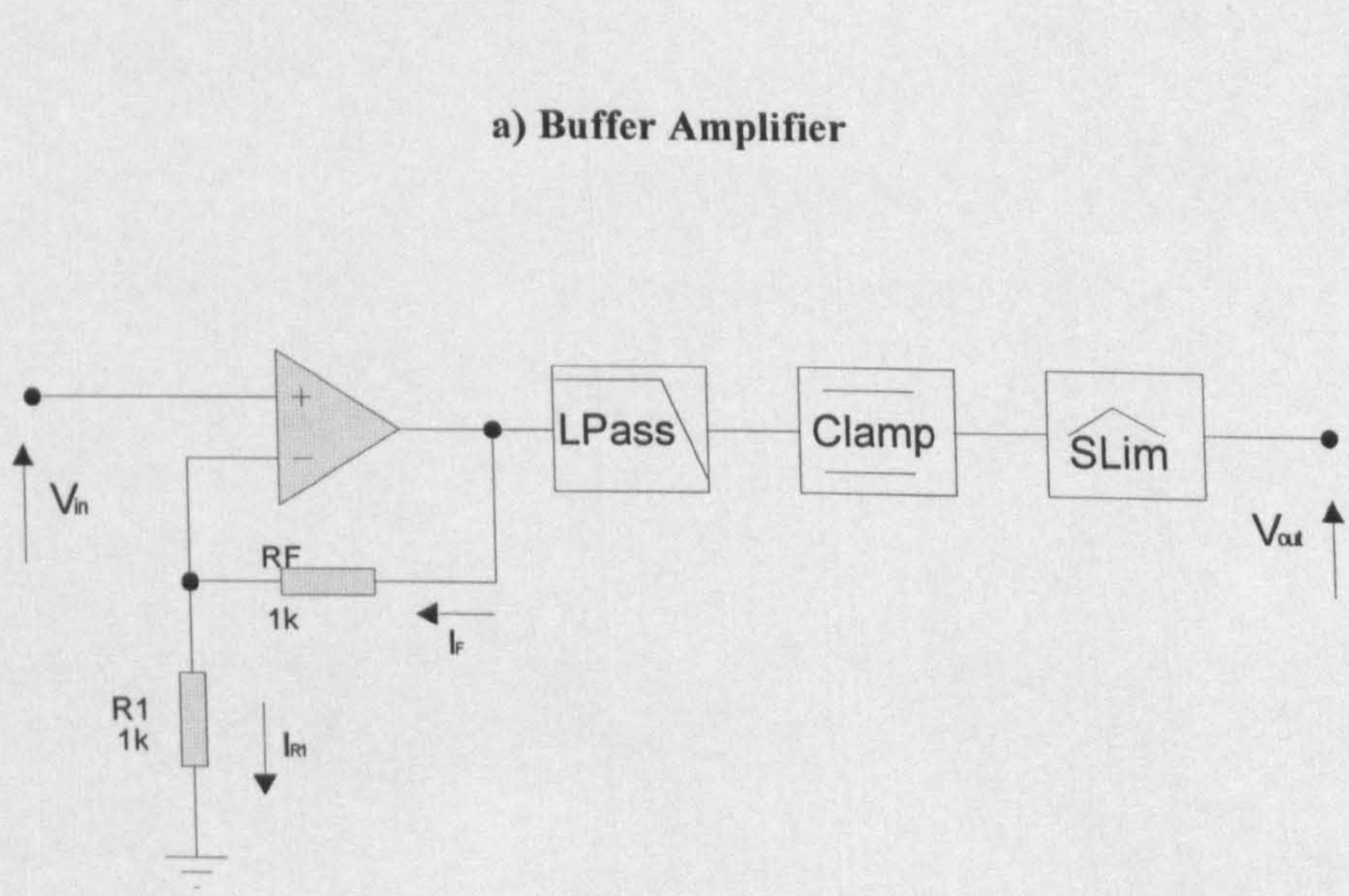
A comparison of the performance of the various op-amp models is given in Figure 4-33. This shows the increases in simulation time as the components representing the non-ideal characteristics are added to the ideal model. A PSPICE simulation of this circuit using a level 1 macro-model and the default settings (but with the output file disabled) took 256 seconds. The most complex PWL inverting op-amp model is therefore approximately 360 faster than the PSPICE macro-model with no significant loss of accuracy.

A similar approach was used to create models of non-inverting op-amp circuits. The simplest model is for the non-inverting buffer shown in Figure 4-34(a). In this model the output of the op-amp is assumed to be identical to V_{in}. The input waveform therefore directly drives the low-pass filter component. The low-pass filter uses the passive model previously described with the time constant corresponding to the unity-gain bandwidth of the op-amp. The voltage clamp and slew rate limit components are the same as used for the inverting op-amp model. It was also derived from the *TOpAmp* class.

The applications of buffer amplifiers are limited, the model of the non-inverting amplifier



a) Buffer Amplifier



b) Non-Inverting Amplifier

Figure 4-34. Models of Non-Inverting Op-Amp Circuits.

in Figure 4-34(b) provides a more useful building block. This is more complex than the buffer amplifier model since it requires models for the two resistors. The voltage at the inverting input terminal is assumed to be the same as the non-inverting input terminal voltage (the model can easily be modified to include a d.c. offset voltage source between the terminals if required). The input voltage waveform (or a d.c. shifted version) is therefore applied to resistor *R1*. This resistor is modelled by a *TResistorVI* object (i.e. it generates a current output from a voltage input and a reference voltage). The feedback resistor model generates a voltage based on its input current (set by *R1*) and input terminal voltage. This required a new model to be created since the input terminal voltage is a PWL waveform rather than a d.c. value. The new model (*TResistorIVV*) is derived from the *TPWLAComp2* class (like the summing junction model *TASum*). Its *Evaluate* operation calculates a voltage from the current input and then sums this with the voltage input to generate a voltage output. This approach allows the current and voltage input waveforms to be varying independently. Although this is not strictly necessary in this case, it enables resistor *R1* to be replaced by alternative components (e.g. a high pass filter) to model other types of non-inverting amplifier.

The low-pass filter component is the same as that used for the buffer amplifier. Unfortunately, the value of the time constant depends on the closed-loop gain of the amplifier. This can be easily determined from the values of *R1* and *RF*:

$$BW = \frac{B}{1 + \frac{RF}{R1}} \qquad \textbf{Equation 4-8}$$

The appropriate value of the time-constant is set automatically when the low-pass filter component is instantiated. The clamp voltage and slew rate limiter models are the same as used in the other op-amp models. This model was also derived from the *TOpAmp* class.

The performance of the non-inverting amplifier model is not as efficient as the inverting amplifier model: it requires more memory for its intermediate results and took approximately twice as long to process a 1kHz sine waveform with 1000 cycles (1.48 seconds). The increased execution time is due to the increase in the number components used and the complexity of the *TResistorIVV* model that maintains an interpolated event

queue for its two independently changing input signals and filters redundant points from its output waveform. The PWL simulation is still over 150 times faster than PSPICE.

4.4.5 Construction and Validation of Other Models.

Active filters are important building blocks in many mixed-signal applications. A simple active low pass filter based on an op-amp integrator is shown in Figure 4-35. Since the inverting input terminal acts as a virtual earth to signals, this circuit can be represented by the PWL model with a passive integrator shown in Figure 4-31. However, in this case *CF* represents a physical circuit component rather than an observed effect. The value of *CF* is specified together with *RF* when the model is instantiated. The effect of limited op-amp bandwidth is usually masked by the filter cut-off frequency so can be neglected without introducing significant errors: if a uA741 op-amp is used for the circuit in Figure 4-35 it will have a bandwidth of 500kHz but any signals of this frequency will be attenuated by over 50dB by the filter which has a cut-off frequency of 1kHz. Simulation results for the PWL model and PSPICE for this filter are shown in Figure 4-36. When the filter was driven by a 1kHz sinewave over 1000 cycles, the PSPICE simulation took approximately 330 times longer than POISE.

Various first and second order active filter models can be constructed using the inverting and non-inverting op-amp models and replacing the resistors with integrator or differentiator components. Versions of these components that produced a voltage output

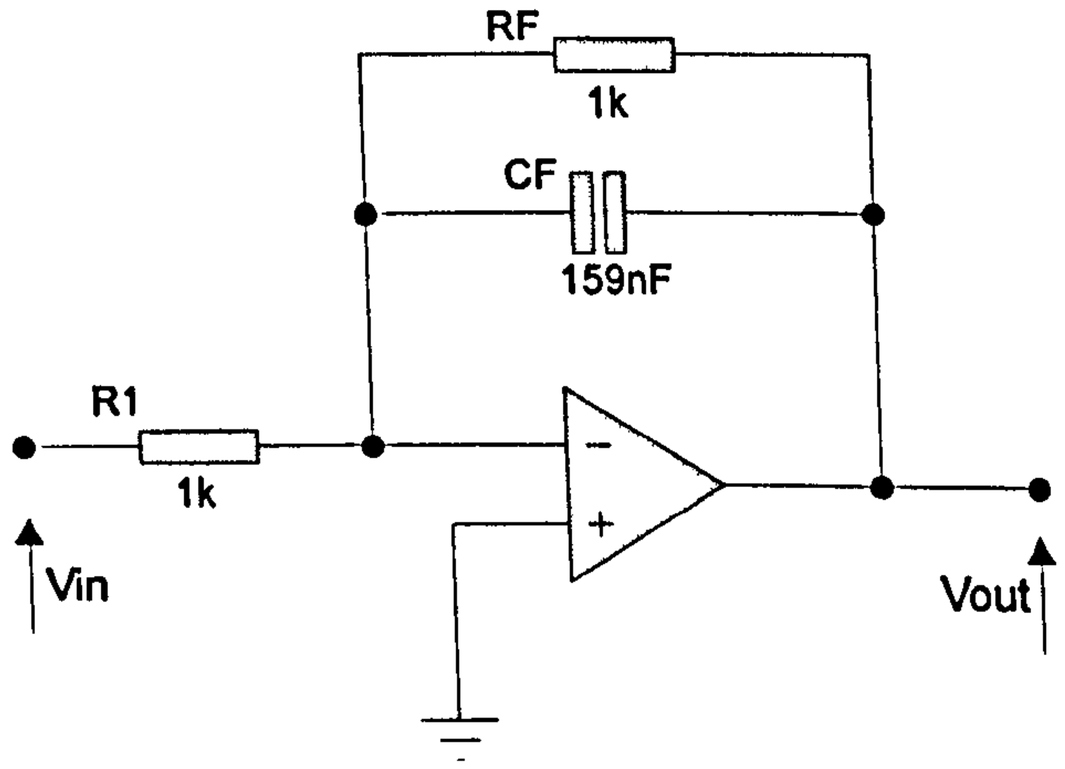


Figure 4-35. First Order Active Low Pass Filter.

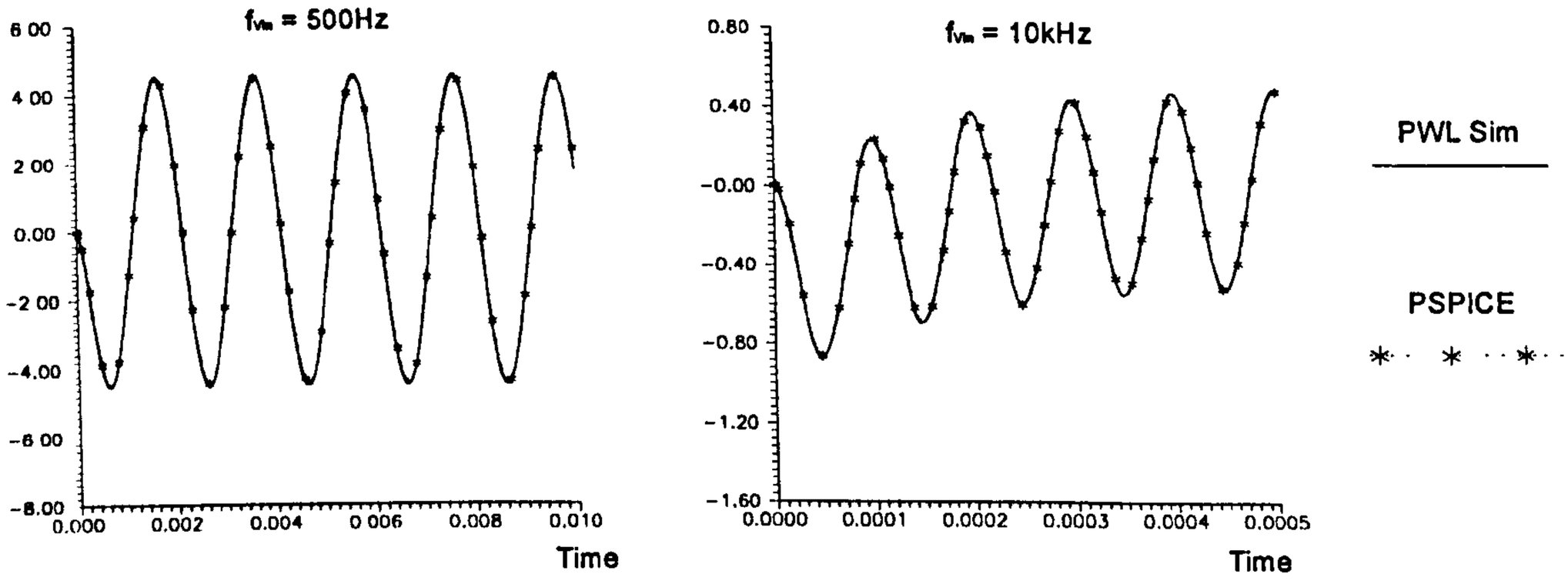


Figure 4-36. Simulation Results for Active Low Pass Filter.

from a current input or a current output from a voltage input were created. Whilst it is possible to use the integrator and differentiator components to create a filter model that has the required transfer response, the model structure will not necessarily correspond to the physical circuit since the bi-directional characteristics of individual resistors and capacitors cannot be represented by the PWL models. The PWL simulation results were compared with PSPICE. The level of correspondence between the PWL simulation and PSPICE was as expected from the results of the passive integrator, passive differentiator and non-ideal op-amp simulations. The performance of PWL simulations for filters using a single op-amp was similar to that of the low-pass filter in Figure 4-35.

4.5 Models of Mixed-Signal Circuits.

All of the models that have been described are derived from a common generic component class and use a common format for their input and output signals. These models can therefore be directly connected together in POISE to build models of mixed-signal circuits (see Appendix A, section 6.3.2). This can be illustrated by the model of the 4-bit digital to analogue converter in Figure 4-37. Two new components were required to create this model: a digitally-controlled switch and a current-summing amplifier. The switch model (*TSwitch1*) was derived from the same root class as digital logic gate models with a single input (*TPWLDComp1*). It has two additional attributes that correspond to the voltage levels corresponding to the switch's open and closed states. These attributes are set when a *TSwitch1* object is instantiated. Its *Evaluate* operation

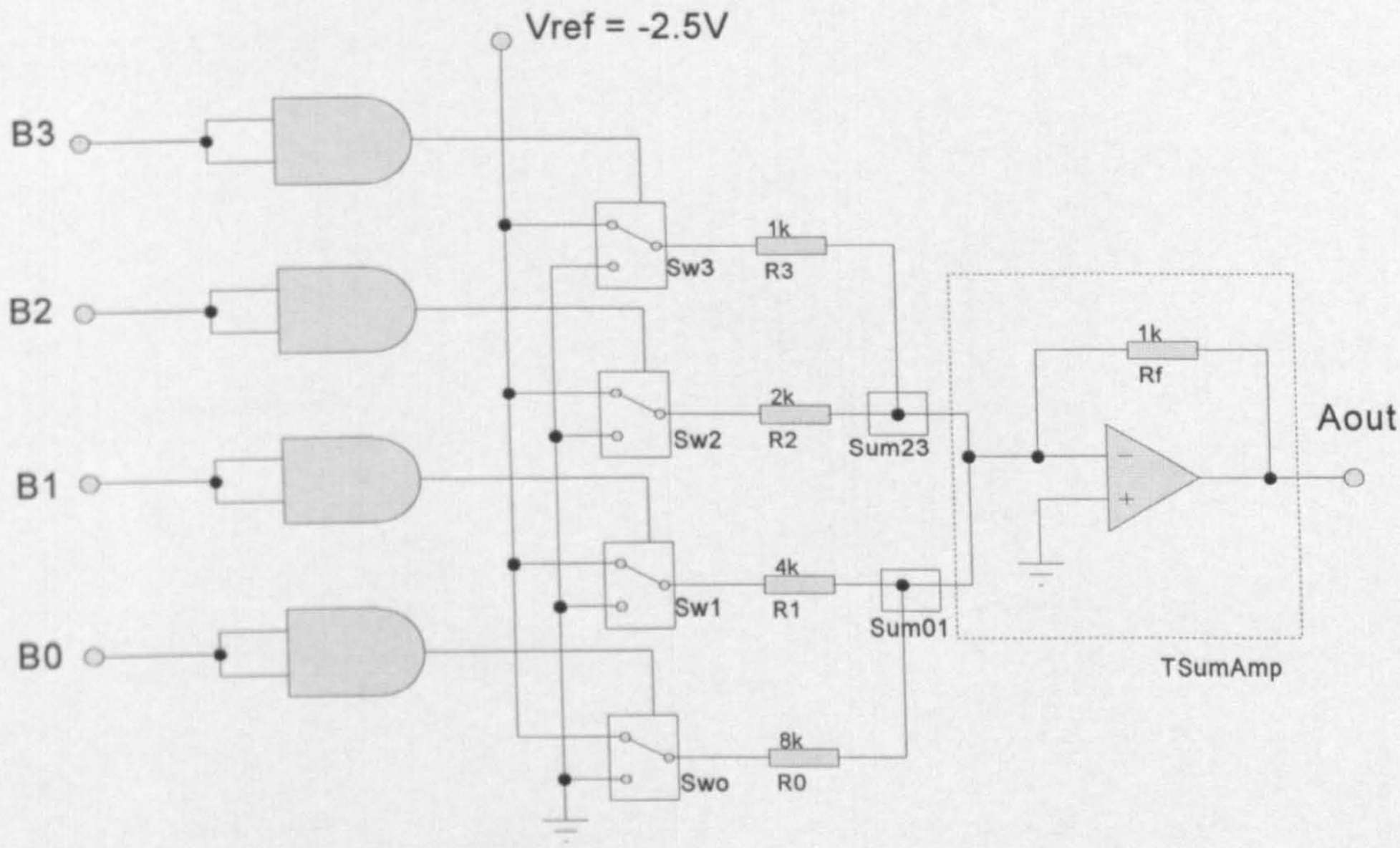


Figure 4-37. Model of a 4-bit Digital to Analogue Converter.

compares the input signal magnitude with a threshold value and sets the output to the appropriate level after a specified delay. The model of the current-summing amplifier (*TSumAmp*) is derived from the same root class as the op-amp models already discussed (*TOpAmp*). Unlike the other op-amp models, it has two input waveforms. Although the *TOpAmp* class is derived from the *TPWLComp1* class (root for classes with a single input waveform) no additional attributes are required or operations redefined by the *TSumAmp* class. This is because a *TOpAmp* object only instantiates its component models (passing them the identity of their input and output waveforms) and calls their operations. The *TSumAmp* constructor passes the identity of the input waveforms to the *TASum* constructor which it uses as its input model.

Simulation results for the 4-bit digital to analogue converter are given in Figure 4-38. This shows close agreement between the PWL model and PSPICE. The PWL simulation in POISE took less than 55ms to complete while the execution time for PSPICE with default options and the output file disabled was 37 seconds. The PWL simulation was therefore over 670 times faster than PSPICE with no significant loss of accuracy.

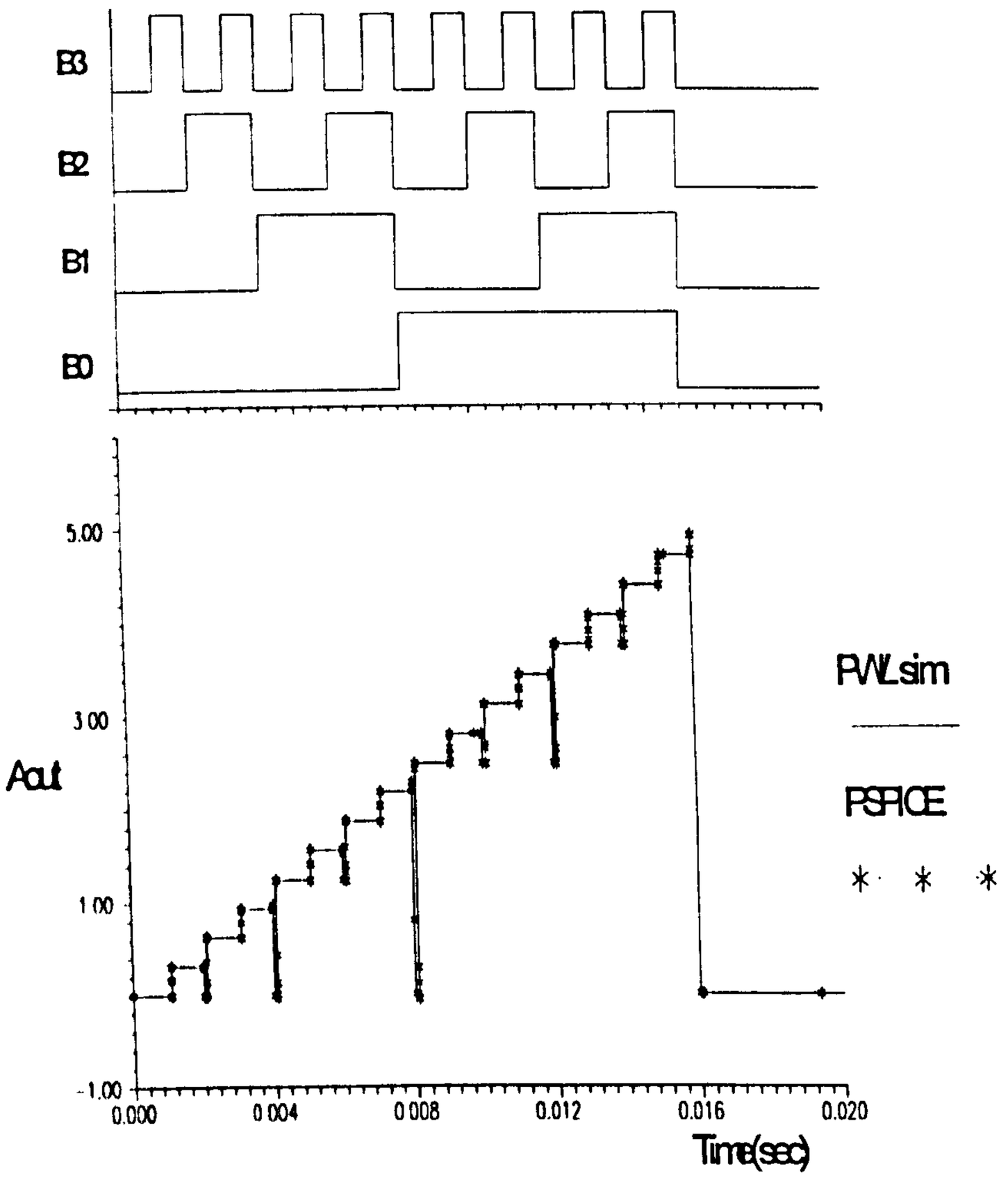


Figure 4-38. Simulation of 4-bit Digital to Analogue Converter.

The 4-bit digital to analogue converter was also simulated using SMASH. This used a behavioural model (compiled C code) for the op-amp. The switches were each modelled by an equation statement of the form:

IF V(control) > 2.5V

THEN output = -2.5V

ELSE output = 0V

The transient analysis results were similar to PSPICE and the PWL model. The execution time for SMASH was approximately 7 seconds: significantly faster than PSPICE as would be expected from the use of behavioural modelling. However, this is still over 100 times longer than the PWL simulation.

4.6 Conclusions.

A hierarchy of classes has been developed to enable an object-oriented approach to be used for mixed-signal simulation. The C++ language was selected to implement these classes. This decision influenced the design of the class hierarchy.

A pure object-oriented design requires all classes to be derived from a single base class. This approach was not used since C++ imposes strong type-checking to ensure design consistency. Forcing all objects to use a common base class would result in inefficiencies without offering significant advantages. Instead, all principal objects were derived from one of two base classes that represent generic signals and models respectively.

The class hierarchies were designed using software engineering methods and a related CASE tool. This enabled the classes to be designed such that the correlation and cohesion between classes was optimised.

The classes near the top of each branch of the class hierarchy are abstract and are never used to form objects directly. As the class hierarchies are descended, specialisations are added to the derived classes until they contain sufficient information to form signal or model objects. Each derived class inherits all of the attributes and operations of its parent classes. This simplifies the creation of new models: for example, an accurate model of a particular type of amplifier could be created from an ideal amplifier class by adding extra attributes and redefining some of the operations. The creation of a child class typically only requires a few lines of code to define how its behaviour is different from its parent. This approach makes the process of writing new models more efficient and less error-prone than conventional techniques.

Object-oriented models have been created for a variety of digital, analogue and mixed-signal components using the methods developed in Chapter 3. The models were evaluated using the Windows-based simulation environment (POISE) described in Appendix A by comparing the results against the observed behaviour of physical circuits and the output of commercial simulators.

The initial simulations identified a small number of minor errors and inadequacies in the object-oriented component class hierarchy. Consequently, the hierarchy went through several iterations during the model development and validation process. As the class structures became more mature, the effort required to make alterations tended to decrease since the changes were confined to those classes at, or near the top of the hierarchy and so were automatically inherited by all derived classes.

It was shown that the object-oriented approach greatly simplified the task of developing new component models once suitable parent classes exist. This was the case for all models of two-input logic gates: the models are all derived from a common parent class and so inherit its properties. The construction of a new model only requires one simple operation (*Evaluate*) to be defined.

Since all component classes share a common base class, they also have consistent external interfaces (although not necessarily identical). This simplified the construction of test-harnesses which could be reused for a number of different models (one model could be substituted for another as the interfaces were the same). It also made it possible for simple models to be connected directly together to form structural models of more complex circuits e.g. an exclusive-OR gate or digital to analogue converter. This approach was extended to create classes for structural models of general purpose circuits that instantiated several component classes e.g. the *TOpAmp* class. These classes invoked operations in their constituent classes in the appropriate order to produce the required behaviour. The interfaces of these structural model classes was also consistent with the simpler classes. This also makes it possible to use the models with hierarchical and mixed-level simulation methodologies.

The unidirectional characteristics of PWL models limit their application for circuits containing feedback. It was shown how these limitations could be overcome for digital circuits and for analogue circuits with low gains. Analogue circuits with high open-loop gains require alternative techniques. Unidirectional models were shown to produce

satisfactory results if the current around feedback loops was considered instead of the voltage.

The PWL models produced results with a level of accuracy similar to PSPICE and other commercial simulators for digital, analogue and mixed-signal circuits. The performance of the PWL simulator was found to be significantly faster than PSPICE for all types of circuit (typically 40 to 400 times faster). For pure digital circuits, the performance of the PWL simulator was comparable to SMASH (a commercial behavioural mixed-signal simulator). The PWL simulator was much faster than SMASH (over 100 times) when simulating a mixed-signal circuit (4-bit digital to analogue converter).

Models of digital components using PWC waveforms were not produced since these are not directly compatible with the analogue and mixed-signal PWL models. The performance of PWC models should be higher than PWL models as less waveform points are required to represent a given signal and no interpolation between points is necessary.

5. Overall Conclusions and Recommendations for Further Work.

The main objective of this research project was to design an improved methodology for simulating circuits containing both analogue and digital components, and to demonstrate this methodology by developing a range of behavioural models to represent commonly occurring components. To determine the issues involved, the characteristics of analogue, digital and mixed-signal simulation were studied and the requirements of an ideal mixed-signal simulator identified.

An investigation into the techniques and approaches that have been used for computer simulation of integrated circuits was carried out. It revealed that despite much research and development activity, an integrated mixed-signal simulator that is well-suited to the design of large mixed-signal circuits has yet to be released. Several approaches that seemed promising for the development of such a simulator were identified and further researched. These included the use of behavioural models to represent circuit elements and the representation of signals as piecewise-linear (PWL) waveforms. These two approaches were selected to create the new simulation methodology.

Behavioural simulation of mixed-signal circuits is likely to become more widely used when a standard mixed-signal hardware description language (VHDL-A) is approved by the IEEE. Consequently, the development of VHDL-A has been closely followed throughout this research project.

Various methods of representing continuous and discrete signals with PWL waveforms were investigated. A method of minimising the number of waveform points required to represent a signal with a particular degree of accuracy was developed. This method was independent of signal magnitude and frequency.

It was determined that a small set of building blocks could be used to construct behavioural models of any digital, analogue or mixed-signal circuit. These building blocks were developed and tested using optimised PWL waveforms. Using the same PWL format to represent input and output signals for digital and analogue models enabled any model to be directly connected to any other. This makes this approach particularly suitable for an integrated mixed-signal simulator.

The nature of the PWL waveforms resulted in models that only contained a few, relatively simple operations. The most complex models were those with frequency-dependent characteristics. Rather than use conventional (computationally expensive) methods to determine the response of such circuits to waveforms in the time domain, the behaviour was modelled by equivalent ideal passive integrator and differentiator circuits. The output waveforms produced by these equivalent circuits were obtained directly from the PWL input waveform segment at the current time step and the magnitude of the output at the end of the previous time step. Since no iterations were required, the execution time for each model was small. This method provided accurate results when the waveform points were appropriately-spaced. The models monitored the lengths of PWL waveform segments and automatically inserted extra points where necessary. Certain models could generate output waveforms with more points than strictly required to represent a signal with the required accuracy. The models detected these redundant points and automatically removed them, maintaining the efficiency of the PWL representation.

Conventional event-driven simulators use a global control mechanism to invoke models when one of their input signals changes state (known as the event queue). This technique was not used in the methodology proposed in this thesis since it would restrict the times at which waveform points could be placed. Instead, each model was responsible for reading the points from its input waveform within an allowed simulation time interval or window and generating its own internal event queue. This allowed the input waveforms for a model with two or more inputs to be generated independently and to have asynchronous points.

Object-oriented methods were identified as being well-suited for the development of simulation models for the proposed methodology. A class hierarchy was created to implement the required objects. Two distinct types of object were created: objects representing signal waveforms and objects representing models. Each of these have quite different characteristics and can be thought of as the equivalents of wires and components in a physical circuit. One of the principal advantages of this approach was the reduction in development time for new models: a new model class could be derived from existing, less specific classes with only the unique aspects of the new model requiring definition.

Models were created for a range of digital, analogue and mixed-signal circuits. It was shown how these models could be directly connected together to form structural models of more complex circuits. In each case, the models were simulated and the results compared with those from conventional simulators. This showed a close level of agreement between the PWL models and conventional techniques, thereby validating the proposed simulation methodology. The performance of the PWL models was typically between 40 and 400 times faster than conventional simulators with comparable levels of accuracy.

Circuits containing feedback revealed a potential limitation of the proposed methodology. This is related to the unidirectional characteristics of behavioural models and the nature of PWL waveforms. It was shown how these limitations could be overcome for circuits containing at least one digital model within the feedback loop (it provides a time delay around the loop and so prevents the feedback from being continuous). The conditions under which an iterative technique to determine the magnitude of continuous feedback signals could be used were also derived. This showed that an iterative technique was not suitable for analogue circuits involving high signal gains (e.g. with operational amplifiers). An alternative technique for modelling circuits involving operational amplifiers by considering PWL current waveforms was developed and tested. This was shown to produce results that were in close agreement with conventional analogue simulators. The performance of the PWL operational amplifier circuit models was up to 400 times faster than the conventional simulators.

A demonstration simulation system called POISE that ran in a 32-bit Windows environment on a PC computer was created to validate the models and to test their performance. This enabled any model to be tested independently and multiple models to be connected together to form structural models of more complex circuits. A relatively simple mechanism could be used to specify the required models whilst POISE was running since all models had a consistent interface, being derived from a common base class. POISE also provided utilities to manipulate and display waveforms.

A number of recommendations for future enhancements have been made. Key recommendations include the development of algorithms to initialise a mixed-signal circuit to the correct state, to partition a circuit into loosely-coupled blocks that can be simulated independently and to re-order the models in a circuit so that they are evaluated in the optimum order.

The amount of memory required by complex PWL waveforms limits the size of circuits that can be simulated. Since the waveforms are implemented by C++ classes, it is possible for them to be created and destroyed dynamically, i.e. during the course of a simulation. A mechanism to manage the creation and destruction of waveform objects during a simulation could dramatically reduce the memory required to simulate large circuits since only those waveforms related to the model currently running would need to be held in memory. The inclusion of such a mechanism should be a priority for any future development of the demonstration simulation environment.

The objects representing the component models can also be created and destroyed dynamically. This would make it possible for the model of a particular component to be replaced by an alternative one, perhaps at a different level of abstraction, whilst the simulator was running (i.e. an adaptive simulation approach). This could improve the efficiency of certain types of circuit that do not always need to be modelled at a low level. Alternatively, this technique could be used for circuits whose behaviour varies according to the input signals (e.g. a digitally-controlled potentiometer). This would enable a complex model to be implemented by several simpler (therefore requiring less memory) alternative models.

It has been shown how some non-linear op-amp circuits could be modelled using the proposed simulation methodology. There are many other op-amp based circuits for which there has not been time to develop models. It may also be possible to apply the methodology to other types of component, e.g. MOS transistors, zener diodes, switched capacitor circuits. These provide a wide scope for future investigations.

The models each process their input and output waveforms during the given simulation time window for as long as the input waveforms contain valid points. No interaction is required from the simulator program during this process. Since the coupling between models and the simulator is very weak, it may be possible to implement this simulation methodology in a multi-processor computing environment, i.e. each model could exist on a separate computer. This would allow models to be evaluated and the simulation to proceed in parallel. The time taken to transmit waveforms between computers is likely to be a major consideration, however, this might still be a practical approach if the models represent large, complex circuits and there is little interaction between the models.

6. Appendix A - POISE: a Windows-based Demonstration Simulation System.

6.1 Introduction.

Various "test bench" programs were written during the development of the models and modelling techniques described in earlier chapters. These test benches controlled the execution of the model code, provided suitable input waveforms for each model and saved the output waveforms. The test benches allowed the models to be "debugged", validated and their performance assessed. Other programs were written for such tasks as generating the required input waveforms and converting the output waveforms to a format that could be directly compared with the results from standard simulators (e.g. SPICE).

The object-oriented nature of the simulation models meant that the external interfaces of the models were similar (if not identical). This lead to strong similarities between the test benches. It became apparent that if a simulation environment was created that could be used with any model or circuit, it would remove the need to write a large number of test bench programs. The acronym POISE was chosen to describe this simulation environment standing for Piecewise-linear Object-oriented Integrated Simulation Environment. POISE served as a demonstration system for the modelling approaches proposed in this thesis. Its development is described in this appendix.

6.2 Identification of Requirements for Demonstration System.

The original test bench programs were written in C or C++ and ran in an MS-DOS environment on a PC computer. The input waveforms were read into memory from a binary file (as opposed to a text file) at the start of each simulation and the results written to a binary output file at the end of each simulation. This approach enabled the simulations to run as quickly as possible, without having to wait while data was transferred between disk and memory. A timer routine was invoked once the input waveforms had been read and stopped before the output waveforms were written. This provided an accurate measure of how long a simulation took to run - comparisons could

then be made with the performance of other models and other simulators. This timer routine was required since the execution time of the test bench programs was often dominated by the time required to read and write waveform files.

To make fair comparisons between the performance of different models and modelling approaches often required simulations to be carried out over a large number of waveform cycles. This presented a problem for test benches running in an MS-DOS environment where each data element (e.g. an array of PWL waveform points) was required to be wholly contained within a single 64 kilobyte segment in memory. The maximum number of cycles that could be processed in a single simulation was limited to about 1500 for digital waveforms and about 150 for sinusoidal waveforms. A decision was made to use a different operating environment for POISE to overcome this limitation. The Windows WIN32S environment was selected since a suitable development tool was available (Borland C++ V4.5). The size of waveforms that could be loaded into memory in a simulator running under WIN32S is only practically limited by the amount of memory on the computer. Several commercial simulators that run in a Windows environment were available. Direct comparisons could therefore be made with the performances of these simulators as they all ran in a common environment.

POISE required a user interface to enable input waveforms to be selected and the simulation time displayed (as with the MS-DOS test bench programs). The other requirements are related to the universal nature of POISE. Three different approaches could be adopted to facilitate a simulation environment that could work with any model:

1. Use conditional compilation to automatically generate an executable environment unique to the model under test.
2. Generate linkable object code for each model (e.g. a Windows "Dynamic Link Library" (DLL)) and call the appropriate code when the simulation environment is loaded.
3. Generate a single executable simulation environment containing all models. Select the required model when the simulation environment is running.

Approach 1 can generate an executable program containing only instructions relevant to the particular model under test. Consequently, this program can be efficient in terms of execution speed and computer resources required. Its main disadvantage is the requirement to compile the environment before each simulation. Approach 2 requires a large number of object modules to be created. Since an object-oriented hierarchy has been used to create the models, much of the code in each of these modules will be identical. This approach is therefore inefficient in terms of the number of files and disk storage required. Approach 3 will result in a larger executable program which must be loaded into memory, reducing the resources available for storing waveforms. However, the object-oriented hierarchy results in most models only requiring a few lines of code. Therefore an executable program created using this approach might not be significantly larger than one produced by approach 1. Approach 3 has the advantage over the other two of allowing multiple models to be simulated in succession without having to recompile or reload the program. The increased flexibility it offers is advantageous for a demonstration system. Approach 3 was therefore adopted for POISE.

The demonstration system would be of limited use if it could only simulate one model at a time: commercial simulators process circuits consisting of many interconnected models. The simulation environment must therefore allow similarly complex circuits to be defined and processed correctly. This requires mechanisms to select the models to be invoked (i.e. the model objects to instantiate) and to specify the parameters and signals associated with each model instance. These mechanisms are complicated by the possibilities of having multiple instances of any model and of output waveforms driving multiple inputs. The ability to define circuits consisting of multiple models introduces problems related to the order in which the models are evaluated and feedback between models. These problems were discussed in Chapter 4. The development of an algorithm to correctly handle all types of feedback is outside the scope of this project. It was therefore decided that feedback between analogue models would not be supported by POISE and that models would be evaluated in the order in which they were specified. Feedback between models of digital gates (where there is a delay between the input and output signals changing state) would be supported provided the models are specified in an appropriate order. This approach introduces the possibility of a simulation entering an indefinite loop if

inappropriate models or signal waveforms are used. It requires a mechanism to monitor the simulation time and number of iterations around a loop so that the simulation process can be terminated in a controlled manner if an error condition arises.

Several utility programs were written for the MS-DOS test benches to create suitable input waveforms, display the results and convert waveform files to other formats. These features are incorporated and enhanced in POISE.

6.3 System Design.

The functions required of POISE can be split into three main categories:

1. Management of Simulation Environment.
2. Specification of circuits and waveforms.
3. Simulation of specified circuit.

The first category provides the overall control for the other two. These activities are summarised in Figure 6-1.

6.3.1 Management of Simulation Environment.

The main body of the POISE program is responsible for the overall management of the simulation environment. This involves the creation of the windows that provide the user

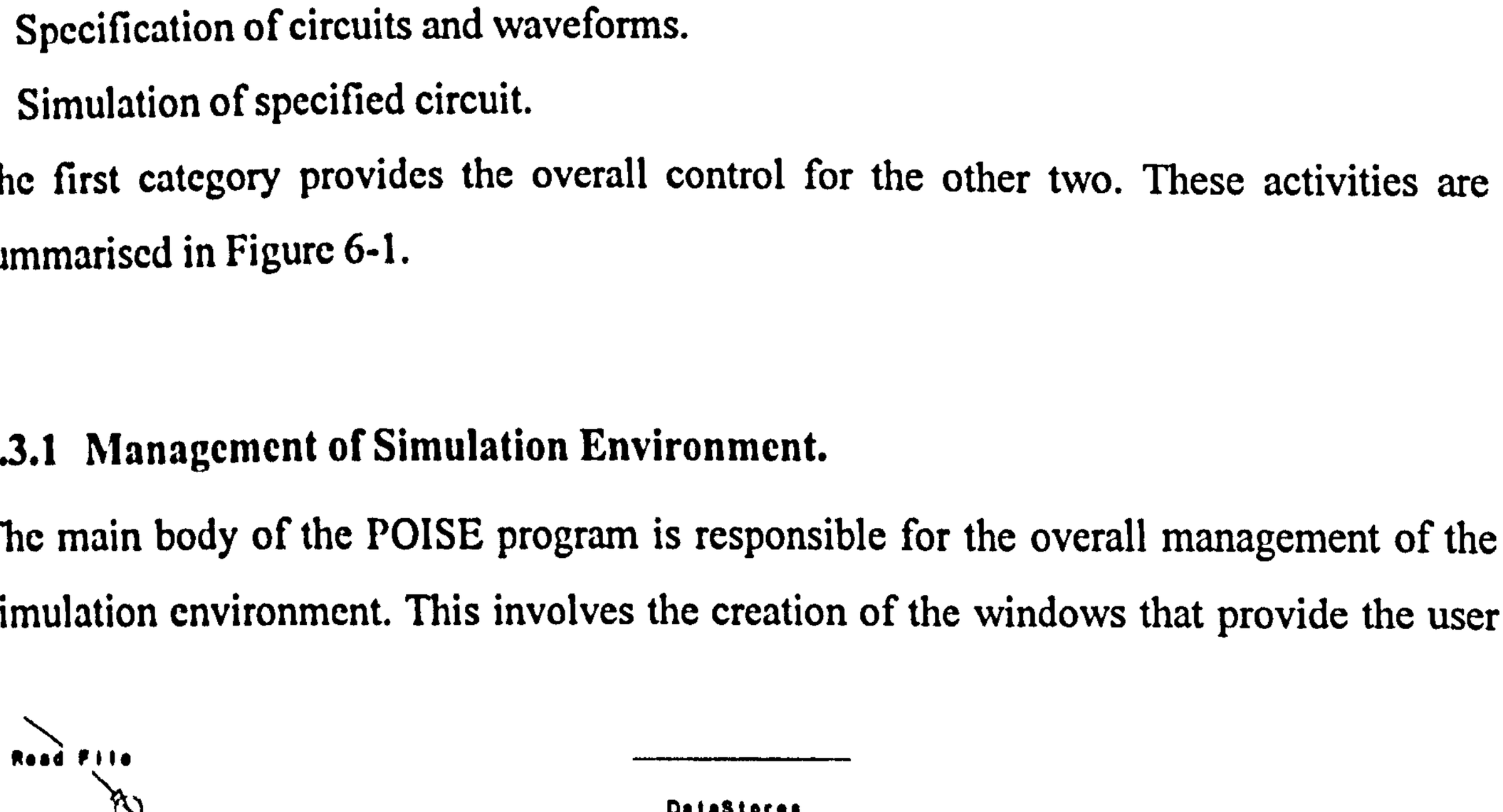


Figure 6-1. Program Structure of POISE.

interface and the servicing of menu commands associated with the main window. Writing programs to run in a Windows environment can be a complex task as it involves calling routines to create and manipulate various windows, interpret mouse commands, select items from menus, etc. The routines that Windows provides for these tasks are written at a low level and do not support object-oriented programming techniques. Libraries of interface routines are available to simplify the task of writing Windows programs and using object-oriented methods. The Borland C++ Object Window Library (OWL) version 2.5 was used to create POISE.

A program written using OWL consists of two main classes. The first is derived from the OWL *TApplication* class. It is instantiated by the main function (called "*OwlMain*) and contains an *InitMainWindow* operation. This instantiates the second class, derived from the OWL *TFrameWindow* class. These classes in POISE are called *TSimApp* and *TSimAppWnd* respectively. The *TSimAppWnd* class defines the operations that collectively form POISE and contains the attributes that are required to control them. Its structure is shown in Figure 6-2. This shows that the *TSimAppWnd* class contains a large

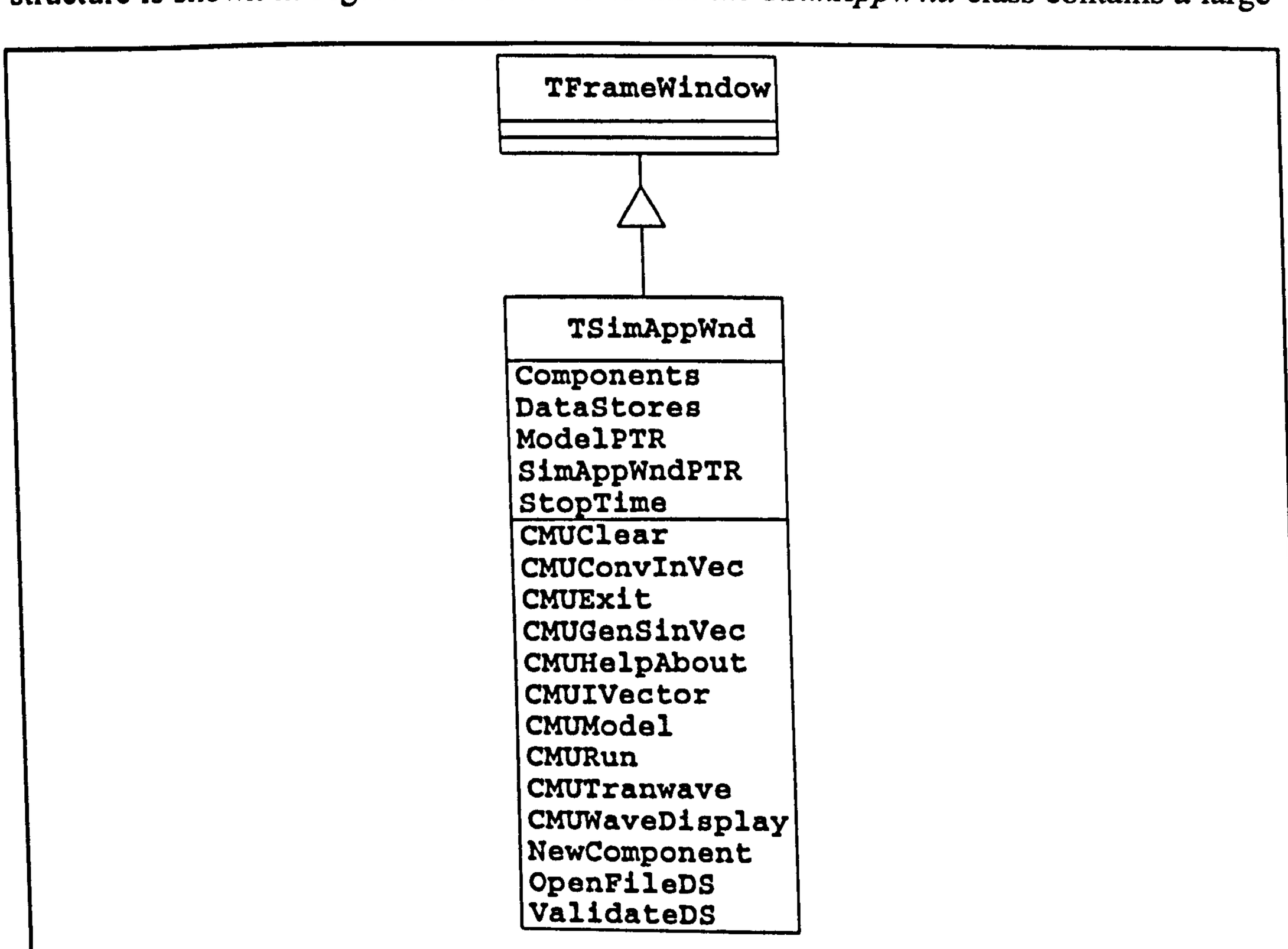


Figure 6-2. Structure of class defining main POISE Window.

number of operations. The operations with names starting "CMU.." are invoked in response to the user selecting an option from a pull-down menu. The simulation can therefore be set up and performed interactively.

6.3.2 Specification of Circuits and Waveforms.

A key issue in the design of POISE was the method used to create an executable form of a circuit description consisting of one or more models. One of the major advantages of object-oriented programs is the capability to create and destroy objects while the program is executing (i.e. at "run time"). In C++ this is done with Constructor and Destructor functions associated with each class that are invoked by the "*new*" and "*delete*" operators respectively. Since component models and waveforms are each implemented as classes, objects representing specific models and waveforms ("instances") can be created and destroyed whilst POISE is running.

In order to process a collection of instantiated model and waveform objects as an electrical circuit, some mechanism must exist to identify how the objects are connected and the order in which they should be evaluated. The *new* operator returns a C++ pointer to an object that has been successfully created. The type of this pointer will reflect the class of the object created, e.g. if a *TSumAmp* object is created the pointer will be of type *TSumAmp**. The strong type checking used in C++ prevents this pointer being assigned to a pointer of a different type (e.g. *TXOR**). However, since all model classes are derived from the *TGenComponent* base class, the pointer returned by the *new* operator when a model object is instantiated can be legally assigned to a pointer of type *TGenComponent**. A collection of pointers of type *TGenComponent** can therefore be used to identify the models that have been created. These pointers can then be used to invoke the appropriate operations within each model to simulate the circuit (e.g. *Reset*(), *Run*()). Similarly, all signal waveform classes are derived from the *TDataStore* base class so a collection of pointers of type *TDataStore** can identify all waveforms that exist.

A library of utility classes, written in C++ was provided with the Borland C++ compiler. A container class template from the Borland C++ Class Library (*TIDoubleListImp*) was

used to store the pointers to the models. This container class enabled the pointers to be held in a "linked list" i.e. the pointers were stored in the order that they were added with new pointers being appended to the end of the list and extra memory allocated automatically. Associated with this class is an "iterator" class (*TIDoubleListIteratorImp*) that steps through each pointer in the container in turn and so allows the models to be evaluated in the correct sequence. The iterator includes a *Restart* operation that enables the list of models to be stepped through more than once. Rather than store the model pointers directly in the container, they were first copied into a new class (*TModelInfo*) together with a string containing the name of the model instance (e.g. "OpAmp1"). This enables the container to be searched for a named model instance which can then be accessed via the pointer. The structure of these classes is illustrated in Figure 6-3: the *TModelList* class is formed from the *TIDoubleListImp* template while the *TModelListIterator* is formed from the *TIDoubleListIteratorImp* template.

A similar approach was used to record the identities of waveform objects: a *TDataStoreList* container class stores pointers to *TDStoreInfo* objects, each of which contains a pointer to a waveform together with its name. The *TDataStoreList* class does

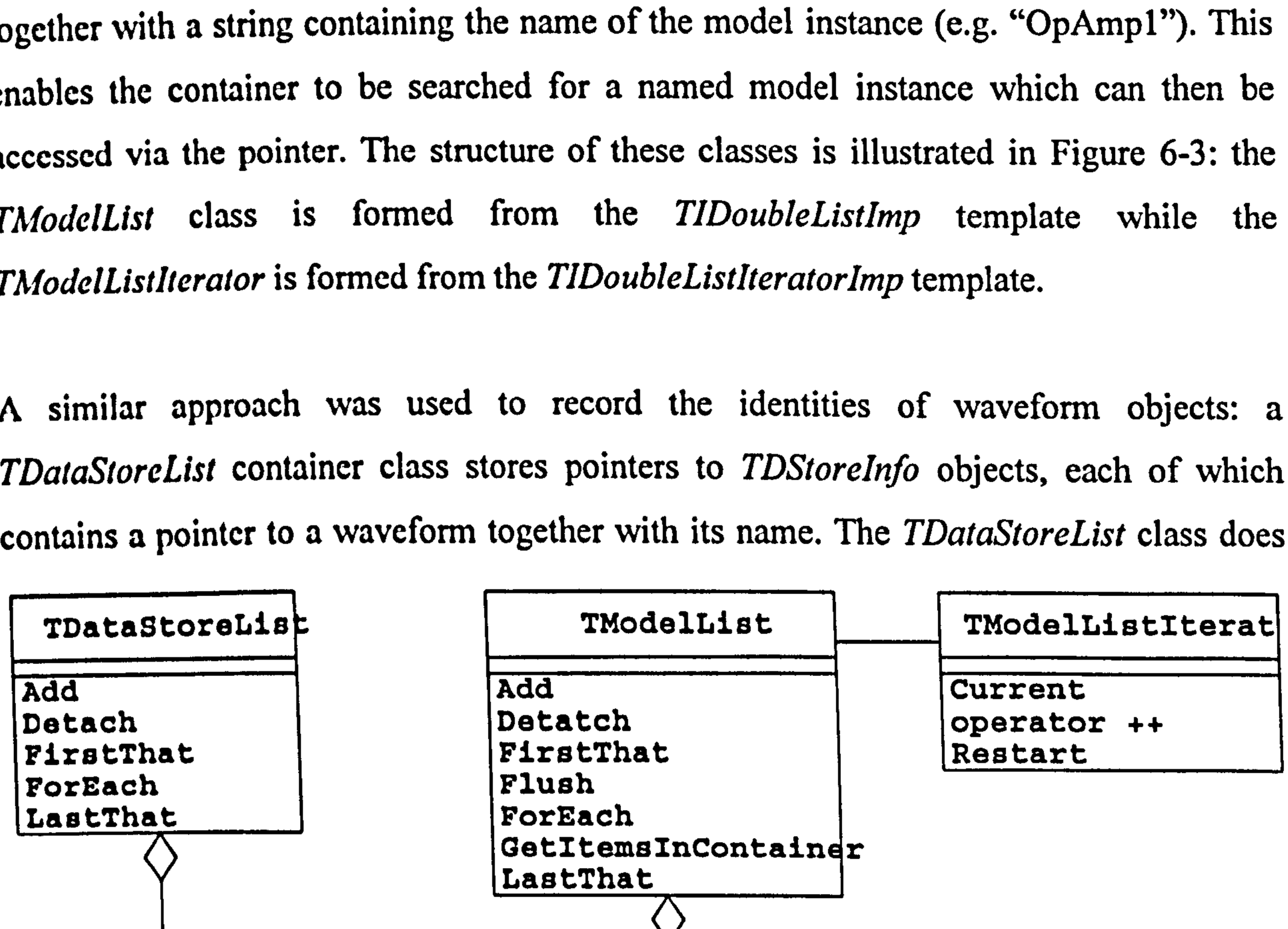


Figure 6-3. Container Classes used in POISE.

not require an iterator since it is only used when waveforms are created: when a waveform name is specified, the *TDataStoreList* is searched to discover if a waveform of that name already exists, if not the waveform is created and its details added to the end of the *TDataStoreList*. The structure of the *TDataStoreList* class is also shown in Figure 6-3.

The *TDataStoreList* and *TModelList* classes are used to instantiate the *DataStores* and *Components* objects respectively as shown in Figure 6-1 and Figure 6-2.

The specification of a circuit involves the selection of appropriate models, customisation of the model parameters and allocation of input and output signal waveforms. This process is performed in POISE via several interactive forms. The first form is a "listbox" that is invoked by an "Add Model" menu item (this is implemented by the *CMUModel* operation in Figure 6-2). The listbox allows the user to scroll through a list of available model types and select one. A second form is then invoked that allows the user to enter the model name, model parameters and input and output waveform names. Each model instance and each waveform is required to have a unique name. POISE checks the *DataStores* container for input waveforms with the given names. If it finds one, the address is passed to the model constructor. If the input waveforms have not already been loaded into POISE, another interactive form is invoked that permits a waveform file to be selected and opened. New input waveforms and the output waveforms are added to the *DataStores* container when the model is created. A pointer to the model and its name are added to the *Components* container.

The forms were created using the Borland Resource Workshop tool. This provides graphical symbols such as selectable buttons and scroll bars. The interfaces to the forms were implemented as individual classes and derived from base classes in the Borland object windows library.

6.3.3 Simulation of Specified Circuit.

The simulation is started by a "Run" menu item that invokes the *CMURun* operation in Figure 6-2. The simulation process essentially consists of running each model in turn, in the order specified (the order in which the models were selected). If an input signal for a model is generated as an output waveform by another model, the model providing the output waveform must have been created before the model it drives. The model creation process will then automatically set the input waveform pointer in the driven model to the address of the output signal provided both waveforms have the same name. This can be illustrated by the circuit shown in Figure 6-4: the model for *Inv1* must be created first, this will add pointers to the *In1* and *Node1* waveforms to the *DataStores* container. When the *AND1* model is created, new waveforms will be created for *In2* and *Out1*. The *Node1* waveform already exists so its address can be passed to the appropriate input of *AND1*. When the simulation is performed, the *Inv1* model will be allowed to run for a specified simulation time interval; the *AND1* model will then be allowed to run for the same time interval, provided the *In2* and *Node1* waveforms both contain valid points. This process can be repeated if the simulation time has not reached the specified simulation run time (circuits with feedback may require an iteration of this loop for each event that is generated in the feedback waveform). The operation of the simulation algorithm implemented in POISE is illustrated by the pseudo code in Figure 6-5 (the *SetState* and *SaveState* operations enable an output waveform to drive inputs on more than one model and were described in Chapter 4).

A timer routine is started when the *CMURun* operation is invoked and stopped when the simulation has completed (but before the waveform files are written to disk). This was used to measure the simulation times.

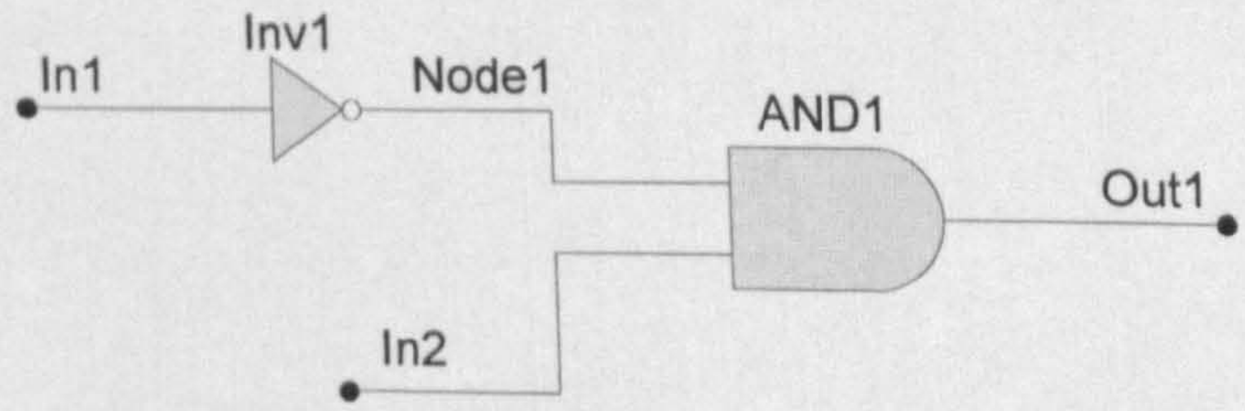


Figure 6-4. Naming of Signals and Components.

```
WHILE (ModelPTR->GetStatus() !=END_MODEL && LoopCount < MaxIteration)
        {
        WHILE (iterator valid)
                {
                ModelPTR = iterator.Current()->Address();
                ModelPTR->SetState();
                ModelPTR->Run(FreeRunTime);
                ModelPTR->SaveState();
                iterator++;
                }
        iterator.Restart();     //go back to 1st model
        LoopCount ++;
        }
```

Figure 6-5. Simulation Algorithm used in POISE.

6.3.4 Other Facilities.

The other facilities provided by POISE fall into two categories: on-line help and waveform conversion.

Windows applications can support comprehensive on-line help features to assist the end user. As POISE is only intended as a demonstration system for the algorithms developed as part of this project, comparatively little time was spent developing the on-line help system. The Microsoft help compiler was used to generate a help program (written using a common word processor) to provide information about how the model parameter forms should be completed. This is activated by a button on the "edit model parameter" form. Other help files could be produced in the same way if POISE were to be developed to a level where it was run by inexperienced users.

The waveform conversion facilities were written as stand-alone programs that were "spawned" as "child processes" from within POISE (i.e. each program creates and runs in a new window). These programs generate binary waveform files corresponding to various types of waveform; convert binary waveform files to formats supported by other tools (e.g. Matlab) and draw waveforms inside a window.

These utility programs complete the graphical user interface (GUI) for POISE. An example of the user interface is given in Figure 6-6.

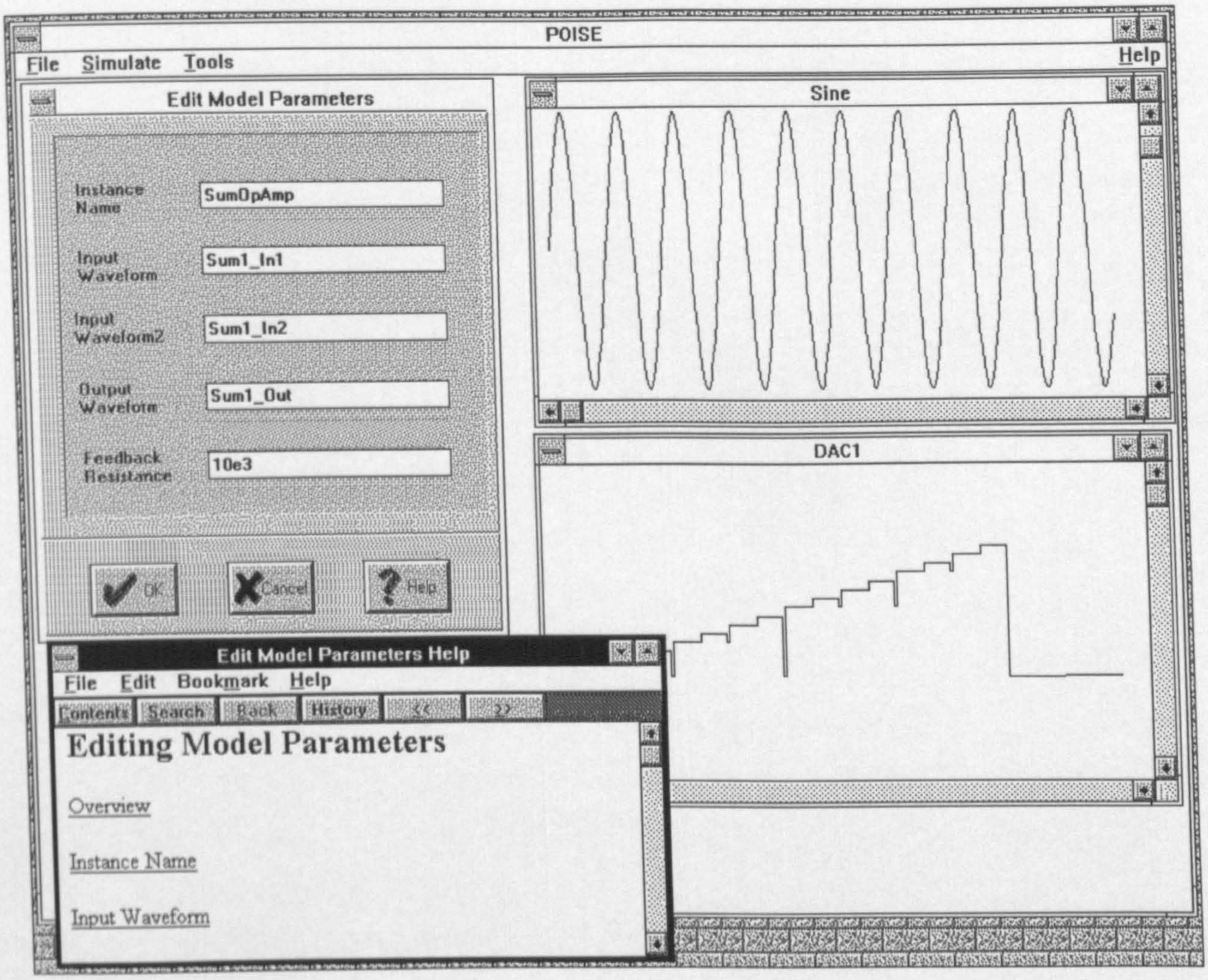


Figure 6-6. Example of User Interface for POISE.

6.4 Evaluation of Demonstration System and Recommendations for Future Enhancements.

POISE enabled component models to be tested both independently and as interconnected circuits without the need to produce a test bench program for each case. It permitted complex models to be simulated with waveforms containing a large number of points (the major weakness with the MS-DOS test bench programs). It was used for the validation of the models and evaluation of their performance discussed in Chapter 4 and so met its basic requirements.

During the model validation process, various issues related to the limitations of POISE and the features it provided were revealed. One of the major limitations was concerned with the allocation of memory for the waveforms. POISE allocates memory for all waveforms and models when the circuit is specified, prior to running the simulation. This approach prevents the execution of the simulation from being held up while waveform files are transferred between disk and memory (a comparatively slow process). Unfortunately, the memory for each waveform is only allocated when the waveform object is instantiated. This allows a relatively simple structure to be used for the waveform but prevents its size from being subsequently increased. Sufficient memory must therefore be allocated to each waveform to ensure it can contain all the points required. This is not a problem for input waveforms that are read from a file and whose size is therefore known in advance. However, the number of points in an output waveform cannot be determined until a simulation has finished. The required size of output waveforms must be estimated when they are instantiated. Writing points to a waveform beyond the end of its allocated memory is not prevented by C++ but will result in unreliable data and is likely to cause the program to crash. A worst-case estimate of the waveform size must therefore be used: e.g. the output waveform of a two-input digital model could contain as many points as the total number of points in both input waveforms. This approach causes the space allocated to output waveforms to increase significantly as the number of models in a signal path between the circuit inputs and outputs increases. Over cautious estimates can result in the number of points written to an output waveform only occupying a small proportion of the memory allocated for it. Models that can generate output waveforms containing many more points than their input waveforms (e.g. integrator and differentiator blocks) require high estimates of their size to provide a margin of safety. This can lead to very inefficient use of memory. Since a limited amount of memory is available, there is a limit to the size of circuit and associated waveforms that can be simulated. The approach for allocating memory does not make the best use of the resources available. Dynamic allocation and de-allocation of memory as the simulation is run could improve this situation.

The waveform objects follow the principle of encapsulation and hide the details of their internal structure and operation from the models that process them. An input waveform

object provides a model with the current point or moves to the next point when so requested by the model. This requires that an input waveform is only processed by one model at a time. The approach used in POISE simulates each model independently, in the specified sequence so the possibility of processing a waveform by more than one model at a time does not arise. However, a user could specify that both inputs of a two-input model were driven by a common waveform. This would cause the waveform points to be interpreted incorrectly. POISE detects if a common waveform is specified for both inputs of two-input models and automatically creates a second waveform that is a copy of the specified waveform to drive the second input. Each waveform can then be processed independently by the model. This technique works correctly with waveforms that are read from files since all of the points are fixed prior to the simulation. It does not work with waveforms generated by other models since the points in the second waveform are copied when it is instantiated and are not updated during a simulation. There is no elegant solution to this problem, it would be better to prevent the user from specifying a common input waveform. Alternatively, a functionally equivalent model with a single input could be substituted.

POISE uses an event-driven approach with each model maintaining its own local event queue. It supports feedback where the feedback loop contains one or more digital models (since there is a time delay between input and output events for digital models). The simulation of complex circuits with feedback requires correct initialisation of all nodes. POISE does not attempt to determine the initial state of all circuit nodes. Its simulation capabilities could therefore be expanded if a suitable initialisation algorithm was developed and incorporated.

The order that models are invoked during a simulation is fixed by the order in which they are created. The user must know the optimum order to specify the components to obtain the correct results. This requirement could be removed by adding an algorithm to automatically derive the circuit connectivity from the specified model and signal names. This algorithm should also be able to partition a large circuit into suitable blocks that could each be simulated independently to prevent feedback from forcing all models to use small simulation intervals.

Circuits are specified in POISE using several interactive forms. This provides a simple user interface when a circuit consists of a small number of models but requires a considerable effort to enter more complex circuits. Support for a textual description of a circuit that could be read from a file would reduce this effort and make it possible to re-run a simulation without having to re-specify the circuit. The emerging VHDL-A standard would be a good choice for the circuit description as it would enable direct comparisons to be made with other mixed-signal simulators. The development of VHDL-A is discussed in Appendix B.

Although POISE provided an environment to validate the component models it was of limited use for identifying errors in the models. This was due to the WIN32S environment not being supported by the Borland C++ debugger tool. A number of test benches that ran in a standard 16-bit windows environment were therefore required for debugging models. These allowed the model code to be stepped through and the contents of variables displayed. This problem should be resolved in future releases of the Borland tools allowing the models to be debugged within POISE.

6.5 Conclusions.

A demonstration simulation system known as POISE that runs in a 32-bit Windows environment has been developed. This enabled the simulation models to be validated without having to produce a large number of dedicated test-bench programs. The circuits to be tested are specified from within POISE by selecting appropriate models and waveforms. The size and complexity of the circuit and waveforms is only limited by the computer memory available: POISE has been used to correctly simulate circuits containing 100 logic gate models and with waveforms containing 120,000 points.

POISE also provides integrated tools for creating, displaying and converting waveforms. The user interface consists of pull-down menus and interactive pop-up forms. Limited on-line help is provided.

POISE was written using the C++ language and an object-oriented approach. This simplified the program development and enabled utility classes supplied by the compiler vendor to be used. Classes from another supplied library (OWL) were used to generate the user interface and to provide the interface to the Windows operating environment.

Several limitations were found whilst POISE was being used to validate the models. These limitations have been discussed and suggestions made as to how they might be overcome. Recommendations for future enhancements have also been made.

7. Appendix B - Implications of VHDL and VHDL-A to Mixed-Signal Simulation.

The VHDL hardware description language was primarily designed for the specification and simulation of digital systems [56]. VHDL simulation is now supported by all the major CAD vendors. It is becoming the preferred manner to describe digital circuits at the chip level [57]. One reason for the increasing popularity of VHDL is the availability of synthesis tools that can generate a chip layout from a VHDL description [58]. Unfortunately, VHDL has very limited capabilities for analogue and mixed-signal circuits. It has been demonstrated that VHDL could be used for modelling and simulating analogue functional blocks [59] but it cannot be used to represent discrete analogue components (e.g. resistors, capacitors, etc.). During the 1992 re-standardisation process it was decided that VHDL should be extended to facilitate the description of any analogue circuit. Due to the number of other modifications that were already proposed to be incorporated into the 1992 standard, it was decided to create a sub-committee (1076.1) that would consider the analogue extensions and produce an interim standard language VHDL-A [60]. This interim language will exist in parallel with VHDL until the next re-standardisation process in 1997 when they will be merged. The interim standard VHDL-A is due to be approved in 1996.

The release of VHDL-A will enable designers of mixed-signal systems to take advantage of the hierarchical design methodologies that are currently used in digital designs. At the present time, synthesis tools for mixed-signal ASICs are mostly experimental. VHDL-A will provide a standard format for the definition of mixed-signal circuits and so will enable commercial synthesis tools to be developed that can be used with CAD tools from any vendor. (This has already happened with VHDL based synthesis tools). The availability of good tools will encourage more designers to choose mixed-signal ASIC solutions since it will simplify the design process (one of the biggest barriers to implementing a mixed-signal ASIC).

The AHDL-A standard will take a different approach to VHDL for simulation. VHDL defines a standard simulator so that a VHDL circuit description will produce an identical set of results with any vendor's simulator. It is not desirable to define a standard VHDL-A simulator due to the wide range of simulation methodologies in use, as already discussed. Neither is it possible to define a standard performance or level of accuracy for a VHDL-A simulator since these two parameters are frequently traded against each other to suit particular applications. VHDL-A will therefore only specify the types of analogue analysis that should be supported, i.e.

- DC operating point computation
- transient analysis
- small-signal AC analysis
- noise analysis
- distortion analysis

These are the minimum types of analysis that must be supported in the language. This does not imply that a simulator should support every type of analysis. VHDL-A is a superset of VHDL. A VHDL-A simulator must therefore be able to process a VHDL description and produce the same results as a VHDL simulator. This may be impossible for some mixed-signal simulator types.

Since VHDL-A will not be linked to a single simulator type, designers of mixed-signal systems will have a choice of alternative simulators to use. This is significant as it will enable a VHDL-A file to be simulated by different simulators at different stages of the design process. For example, the behaviour of a complex analogue block could be simulated with a high accuracy simulator to verify its operation. The validated VHDL-A description of the block could then be used in a large design and simulated with a more efficient but less accurate simulator. This approach will encourage the creation of VHDL-A models that can be reused in future designs.

8. References.

1 L.W. Nagel and D.O. Pederson, "Simulation program with integrated circuit emphasis (SPICE)," Memo ERL-M382, University of California, Berkeley,1973.

2 "IEEE Standard VHDL Language Reference Manual", IEEE Std. 1076-1993, IEEE Computer Society Press, 1993.

3 J. Bellorin and C. Fishbourne, "Object-Oriented analysis of a flexible batch production system," IEE Computing and Control Engineering Journal, Vol. 4, N0. 5, October 1993, pp.223-238.

4 C. Ausfelder, E. Castelain and J-C. Gentina, "An Object-Oriented Simulation Tool to Validate the Dynamic Behaviour of Flexible Manufacturing Systems," in Modeling and Simulation 1992, J. Stephenson (Ed), Society for Computer Simulation International (SCS), 1992, pp.528-532.

5 D. Koch and J. Warschat, "Object-Oriented Management of Product Information for Concurrent/Simultaneous Engineering," International Conference on Concurrent Engineering and Electronic Design Automation (CEEDA'94), Poole, 1994, pp.343-348.

6 J.M Bavcco and A.M.W. Smeulders, "Objects for Simulation: Smalltalk and Ecology," Simulation, Vol. 62, No. 1, January 1994, pp.42-56.

7 L.W. Nagel, "SPICE2, A Computer Program to Simulate Semiconductor Circuits," Technical Report ERL-M520, University of California, Berkeley, 1975.

8 R.L. Geiger, P.E. Allen, N.R. Strader, "VLSI Design Techniques for Analog and Digital Circuits," McGraw-Hill, New York, 1990.

9 W. Christopher, "SPICE 3A7 Users Manual," Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, 1986.

10 K.S. Kundert, "Sparse Matrix Techniques," Circuit Analysis, Simulation and Design (Part 1), Ed. A.E. Ruehli, Elsevier Science Publishers B.V, 1986.

11 F.G. Aldridge, "Analogue/Digital Laboratory (A/D Lab) A solution for the analogue digital design environment," Electronic Design Automation Conference, London, 1987.

12 D. Johnson, "Mixed-mode simulator accurately models real-world designs," Computer Design, August 1,1988, pp.65-67.

13 E. Meyer, "Mixed-signal simulators take divergent paths," Computer Design, January 1, 1990.

14 T.J. Barnes, D. Harrison, A.R. Newton, R.L. Spickelmier, "Electronic CAD Frameworks," Klewer Academic Publishers, 1992, ISBN 0 7923 9252 3.

15 I.E. Getreu, "Behavioral Modeling of Analog Blocks using the Saber Simulator," IEEE 32nd Midwest Symposium on Circuits and Systems, Urbana-Champaign, USA, August 1989.

16 B. Johnson, T. Quarles, A.R. Newton, D.O. Pederson, A. Sangiovanni-Vincentelli, "SPICE3 Version 3F User's Manual," Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, 1992.

17 F.L. Cox, W.B. Kuhn, J. Murray, S. Tynor, "Code-Level Modeling in XSPICE," IEEE International Symposium on Circuits and Systems (ISCAS'92, San Diego, 1992.

18 P.J Mayo, "SMASH: A mixed signal, multi-level simulator," ECAD Educational User Group Meeting, Bristol, January 5, 1995.

19 M. Bloom, "Mixed-mode simulators bridge the gap between analog and digital design," Computer Design, January 15, 1987, pp.51-65.

20 G. Lindsay, "A Mixed Mode ASIC Design Tool for System Engineers," International Conference on Concurrent Engineering and Electronic Design Automation (CEEDA), Bournemouth, March 1991.

21 H.W. Klein, "Mixed analog-digital design in the 1990s," Technology Information Publishing, Aptos, CA, USA, 1991.

22 P.E. Allen and W.M. Zuberek, "Mixed-mode, analogue-digital simulation using SPICE-like circuit analysis programs," Journal of Semicustom ICs Vol.8, No.1, Elsevier Science Pub, England, 1990.

23 M.C. Chain, "Simulation of Mixed Switched-Capacitor/Analog/Digital Circuits with Arbitrary Clocks and Signals," IEEE 34th Midwest Symposium on Circuits and Systems, Monterey, USA, 14-17th May, 1991, Vol.2, pp.566-571.

24 J. White and A. Sangiovanni-Vincentelli, "Relaxation Techniques for the Simulation of VLSI Circuits," Kluwer Academic Publishers, The Netherlands, 1987.

25 B.R. Chawla, H.K. Gummel and P. Kozah, "MOTIS- an MOS Timing Simulator," IEEE Transactions on Circuits and Systems, Vol. CAS-22, December 1975, pp.901-909.

26 R.A Saleh and A.R. Newton, "Mixed-Mode Simulation," Kluwer Academic Publishers, The Netherlands, 1990.

27 B. Hennion and P. Senn, "A New Algorithm for Third Generation Circuit Simulators: the One-Step Relaxation Method," 22nd IEEE/ACM Design Automation Conference (DAC), Las Vegas, USA, June 1985.

28 J. Benoski, J. Besnard, S. Gai, M. Magni and E. Profumo, "Mozart-MM: A mixed-mode and multi-level simulation system," International Symposium on Circuits and Systems (ISCAS'91), Singapore, 1991, Vol.4, pp.2387-2390.

29 E. Lelarasmee and A. Sangiovanni-Vincentelli, "The Waveform Relaxation Method for Time Domain Analysis of Large Scale Integrated Circuits," IEEE Transactions on CAD of IC and Systems, Vol.1, No.3, July 1982, pp.131-145.

30 P. Odent, L. Claesen and H. De Man, "A combined Waveform Relaxation - Waveform Relaxation Newton algorithm for efficient parallel circuit simulation," IEEE European Design Automation Conference (EDAC), 1990, pp.244-248.

31 R. Chadha, C. Visweswariah and C.-F. Chen, "M3-A Multilevel Mixed-Mode Mixed A/D Simulator," IEEE Transactions on Computer-Aided Design, Vol.11, No.5, May 1992, pp.575-585.

32 B.D. Ackland and R.A. Clark, "Event-EMU: An Event Driven Timing Simulator for MOS VLSI Circuits," IEEE International Conference on Computer Aided Design, 1989, pp.80-83.

33 D. Overhauser, I. Hajj and Y.-F. Hsu, "Automatic Mixed-Mode Timing Simulation," IEEE International Conference on Computer Aided Design, 1989, pp.84-87.

34 Y.-H. Shih and S.M. Kang, "ILLIADS: A New Fast MOS Timing Simulator Using Direct Equation-Solving Approach," 28th ACM/IEEE Design Automation Conference (DAC), 1991, pp.20-25.

35 Y.-H. Jun and I.N. Hajj, "A Mixed-Mode Simulator for Digital/Analog VLSI Circuits Using an Efficient Timing Simulation Approach," IEEE International Symposium on Circuits and Systems (ISCAS'91), Singapore, 1991, Vol.4, pp.2383-2386.

36 W.M.G. Bokhoven, "Piecewise Linear Analysis and Simulation," in Circuit Analysis, Simulation and Design Part 2, A.E. Ruehli (Ed), Elsevier Science Pub. 1987.

37 M.T. van Stiphout, J.T.J. van Eijndhoven and H.W. Buurman, "PLATO: A New Piecewise Linear Simulation Tool," IEEE European Design Automation Conference (EDAC), 1990.

38 T.A.M Kevenaar and D.M.W. Leenaerts, "A flexible hierarchical piecewise linear simulator," Integration, the VLSI Journal, No.12, Elsevier Science Pub, 1991, pp.211-235.

39 R. Griffith and M.S. Nakhla, "Mixed Frequency/Time Domain Analysis of Non-linear Circuits," IEEE Transactions on Computer-Aided Design, Vol.11, No.8, August 1992, pp.1032-1043.

40 R.A. Cottrell, "Event-Driven Behavioural Simulation of Analogue Transfer Functions," IEEE European Design Automation Conference (EDAC), 1990.

41 R.A. Cottrell, "Mixed analogue-digital simulation of ASICs using transfer function models," Journal of Semicustom ICs, Vol.7, No.4, Elsevier Science Publishers, 1990, pp.21-25.

42 C. Visweswariah and R.A. Rohrer, "Piecewise Approximate Circuit Simulation," IEEE Transactions on Computer-Aided Design, Vol.10, No.7, July 1991, pp.861-870.

43 G. Ruan, J. Vlach, J.A. Barby and A. Opal, "Analog Functional Simulator for Multilevel Systems," IEEE Transactions on Computer-Aided Design, Vol.10, No.5, May 1991, pp.565-76.

44 C.J. Terman, "Simulation tools for digital LSI design," PhD. thesis, Massachusetts Institute of Technology, September 1983.

45 R. Kao and M. Horowitz, "Timing Analysis for Piecewise Linear Rsim," IEEE Transactions on Computer-Aided Design, Vol. 13, No.12, December 1994, pp.1498-1512.

46 G.R. Boyle, B.M. Cohn, D.O. Pederson and J.E. Solomon, "Macromodeling of Integrated Circuit Operational Amplifiers," IEEE Journal of Solid-state circuits, Vol. SC-9, No.6, December 1974.

47 G. Krajewska and F. E. Holmes, "Macromodeling of FET/Bipolar Operational Amplifiers," IEEE Journal of Solid-State Circuits, Vol. SC-14, No. 6, December 1979, pp 1083-1087.

48 C. Turchetti and G. Masetti, "A Macromodel for Integrated All-MOS Operational Amplifiers," IEEE Journal of Solid-State Circuits, Vol. SC-18, No. 4, August 1983, pp. 389-394.

49 V. M. Ma, J. Singh and R. Saleh, “Modeling, Simulation and Optimization of Analog Macromodels,” IEEE Custom Integrated Circuits Conference, Boston, USA, 3-6 May 1992, pp. 12.1.1-4.

50 M. Rumsey, "Behavioural Modelling of Mixed Analog-Digital Circuits," 7th International Conference on Custom and Semi-Custom ICs, London, 1987.

51 M. Rumsey and J. Sackett, "An ASIC Methodology for Mixed Analog-Digital Simulation," 26th ACM/IEEE Design Automation Conference (DAC), 1989, pp.618-621.

52 G. Gielen, E. Liu, A. Sangiovanni-Vincentelli and P. Gray, "Analog Behavioural Models for Simulation and Synthesis of Mixed-Signal Systems," IEEE European Design Automation Conference (EDAC), 1992.

53 J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorenson, “Object-Oriented Modeling and Design,” Prentice-Hall, 1991.

54 R. A. Saleh, T. Inoue and S. Ido, “Enhanced Circuit Simulation: Expectations, Problems, Implementation and Integration,” Electronics and Communications in Japan, Part 3, Vol. 74, No. 11, 1991, pp. 101-111.

55 W. D. Stanley, “Operational Amplifiers with Linear Integrated Circuits,” 3rd Edition, Macmillan College Publishing Company, New York, 1994.

56 R. Lipsett, C.F. Schaefer and C. Ussery, "VHDL: Hardware Description and Design," Kluwer Academic Publishers, The Netherlands, 1989.

57 J. Hillawi, "VHDL's survival depends on constancy and evolution," New Electronics, January 1991, pp.26-27.

58 G. Hale and S. Redmond, "VHDL and synthesis - living up to the promises," Electronic Product Design, October 1993, pp.47-49.

59 B.R. Stanisic and M.W. Brown, "VHDL Modeling for Analog-Digital Hardware Designs," IEEE International Conference on Computer-Aided Design, Santa Clara, USA, 1989.

60 J. Hillawi, "VHDL 92 deals with past criticisms," New Electronics, September 1991, pp.17-18.