ZESZYTY
NAUKOWE
POLITECHNIKI
ŚLĄSKIEJ

Jerzy J. DĄBROWSKI

PIECEWISE LINEAR APPROACH TO FUNCTIONAL-LEVEL MACROSIMULATION OF ANALOGUE & MIXED A/D SYSTEMS

ELEKTRONIKA
z. 11

WYDAWNICTWO POLITECHNIKI ŚLĄSKIEJ

GLIWICE 2000

POLITECHNIKA ŚLĄSKA
ZESZYTY NAUKOWE
Nr 1419

Jerzy J. DĄBROWSKI

PIECEWISE LINEAR APPROACH TO FUNCTIONAL-LEVEL MACROSIMULATION OF ANALOGUE & MIXED A/D SYSTEMS

Gliwice 2000

PL ISSN 1231-1596

Wydawnictwo Politechniki Śląskiej
ul. Akademicka 5
44-100 Gliwice
tel./fax (0-32) 237-13-81

www.wydawnictwo.polsl.gliwice.pl
wydawnictwo@polsl.gliwice.pl

Nakl. 100+83 egz. Ark. wyd. 10 Ark. druk. 8,25 Papier offset. kl. III 70 x 100, 80 g
Oddano do druku 24.07.2000 r. Podpis. do druku 24.07.2000 r. Druk ukończ. w sierpniu 2000 r.

Fotokopie, druk i oprawę wykonał „ROLEK", Gliwice, ul. Kazimierza Wielkiego 4

To the memory of my mother

PREFACE

Due to the current proliferation of ICs the need of more and more powerful design tools is becoming apparent. It spans from basic synthesis up to sophisticated verification for all design levels. Great effort has already been made to cover this need in digital domain. In the analogue domain, however, circuits are usually designed and verified at the device level, despite the fact that device verification level is very CPU intensive. In particular, the analogue or mixed A/D simulation of large systems performed at the so-called SPICE-level suffers from an excessive amount of computer resources even when provided with some speed-up mechanisms. Hence, following the top-down design strategy, the analogue modelling and simulation adequate also for higher levels of abstraction are actually important objectives. On the other hand, high-level, rough models are usually no more sufficient. Consequently, a new generation of models/macromodels (e.g. functional level) tend to comprise more specifications, in order to provide a designer with more detailed verification results prior to step down to lower design levels. Clearly, timing specifications are of particular interest in most cases. Since the respective timing models are usually based on differential equations, their typical implementation requires very CPU-time consuming procedures.

Recently, a novel piecewise linear (PWL) approach has been proposed to cope with this problem [DAB94, DAB95, DAB99F]. It can be applied to analogue or mixed analogue/ digital networks represented mainly at the functional level. The PWL approach feature: the PWL signals, inertial building blocks as basic modelling units and the explicit simulation algorithm to solve network equations. Also digital primitives can be performed accurately using a similar PWL technique [RUA91, DAB96K] rather than logic states. However, the PWL approach cannot be easily extended to complex digital blocks defined by behavioural description, since it is difficult to propagate adequately the non-Boolean values (rise, fall) through behavioural models [BRE76, ARM88]. Hence, for an inner part of the complex digital model the behavioural description based on logic states is preferred, whereas the input and output stages of it use the PWL technique, when interfacing with analogue models. In fact, they play a role of analogue-to-logic and logic-to-analogue converters that enable signal propagation between the analogue- and digital domain.

Over the last five years the primary ideas evolved to more advanced macrosimulation techniques covering a spectrum of PWL macromodelling issues and the relevant simulation algorithms. They comprise feedback loops or high-order analogue blocks, and on the other hand, the PWL waveform relaxation or "error-compensated" PWL approximation. An effort has been made to implement the PWL approach in VHDL as well.

The purpose of preparing this book was to summarise the work and experience gained with PWL macrosimulation over that period of time. Several people contributed to this work. The author would like to thank his colleagues of Silesian University of Technology, Dr. Jacek Konopacki who developed some primary models and assisted during the early simulation experiments, and Dr. Andrzej Pułka who performed the VHDL implementation.

Special thanks are given to Prof. Adam Macura who headed the Division of Circuit and Signal Theory at Institute of Electronics, Silesian University of Technology over the past three decades, for his fruitful encouragement to writing this monograph.

The financial support of Polish Committee for Scientific Research (KBN) under grant No. 8T11B 04 113 is also gratefully acknowledged.

Finally, the author would like to thank his wife and daughters for their patience and understanding during the many days and weekends spent on preparing this monograph.

CONTENTS

SPIS TREŚCI

1. INTRODUCTION

Modelling and simulation play a major role in the process of designing electronic circuits. By using a simulator, a designer is able to evaluate the performance of the design prior to enter the expensive manufacturing process. As the algorithms became reliable and the models mature, simulators along with other nowadays CAD tools have completely replaced the traditional breadboarding techniques, in particular when designing VLSI circuits.

Basically, there are two main approaches to simulating an electronic circuit. The first assumes the circuit to be a continuous dynamical system, usually described by a set of differential equations with electrical variables such as voltages or currents. The involved *circuit simulator* after resolving this set of equations numerically is expected to provide detailed waveforms at the circuit's nodes and branches. Beside this so-called *transient analysis*, also other tasks exist in circuit simulation, such as *DC analysis*, *AC analysis* or *Fourier analysis*. For a particular circuit analysis usually individual algorithms and models are required [CHU75, RUE86, MCC88, OGR95].

The other approach to simulation makes use of some prescribed discrete states of the circuit rather than of continuous node voltages or branch currents. The in-between states are ignored in this simulation, so it is only adequate for a limited class of circuits. These, however, are the most popular digital (or logic) circuits. The relevant digital (or logic) simulation is based primarily on simple logic operations (Boolean or multiple-valued) with dynamics modelled by delaying the signal transitions between the discrete levels [SZY76, BRE76, DEM81, HAY82, BRY87, SAL94].

The logic simulation and the circuit simulation, also referred to as electrical, began to emerge in 1950s following the use of the first computers. Until 1970s each of them has been developed separately [PED84]. The early logic simulation has been oriented merely towards gate-level and register transfer-level modelling, whereas the circuit simulation towards transistor-level modelling, usually for analogue circuits.

The need of more efficient simulation tools has been stimulated by the progress in technology and designing of ICs. Several new concepts have been developed and incorporated into existing simulation techniques (e.g. partitioning, relaxation, ordering, macromodelling) [BOY74, RAB79, RUE86, WHI87, VAC93]. Also a new simulation level, the *switch-level*, adequate for MOS digital networks has been proposed and implemented in several ways [BRY87, KIM88, RAO89].

At present the VLSI designing is a hierarchical process, which starts usually with a high-level behavioural (and/or architecture) description of the circuit based on primary specifications. The synthesis proceeds "top-down" by translating this high-level description to lower levels such as register/functional level, gate-, switch-, transistor level and it terminates at the physical-mask level. The synthesis is accompanied by design verification that concerns design rules checking

electrical rules checking and predicting of the circuit performance. Simulation tools must support the latter task. Since the verification is performed at different design levels, the needed simulators (including models/macromodels) must span from lower- to upper levels of abstraction. Moreover, to cope with mixed A/D circuits the required class of algorithms and models must be substantially enlarged as compared to purely digital designs.

1.1 Levels of simulation

To enter the designing process a complex circuit is initially described at the *behavioural level* in a hardware description language such as VHDL, VHDL-AMS or HDL-A [IEE93, IEE98, ANA94]. Some virtual components of the circuit (system) to be designed may be defined as structural- or behavioural blocks. The structural blocks define interconnections between blocks, whereas behavioural describe a set of operations to be performed in the system. After compiling the model code, a behavioural simulator can be used to verify the system input/output behaviour. This simulation might be used e.g. for verifying of the system timing for a signal processor or checking a communication protocol for a local area network. In this phase of designing, specific internal structures are not considered.

At the *register-transfer level* (RTL) the system behaviour is mapped into a structure of functional blocks such as ALUs, MUXs, registers, DACs or sample-hold amplifiers. Simpler units like logic gates might be accepted as well. Primarily, the term RTL has been reserved for digital systems [BRE75]. At present it covers additionally mixed A/D systems, and hence, it is alternatively called the *functional level* [RUA91]. At this level, simulation is used to verify correctness of register transfers, ALU operations or interfacing with analogue functional blocks and propagation of analogue signals. The functional level (or RTL) simulation is well suited to evaluate alternative architectures of the system to be designed. In typical cases, however, the timing specifications it provides are rather crude. Certain failures, such as races and hazards for a digital subsystem or incorrect waveforms (e.g. ringing, overshoot) for its analogue counterpart might not be detected.

Gate-level or (*logic*) simulation is used to verify the logical correctness of the digital network, when represented by logic gates such as inverters, NANDs, NORs of flip-flops. The simplest models use two Boolean logic states, accomplished by unknown state X and high impedance state Z, if applicable [BRE76, RAG89]. Most gate-level simulators adopt *event-driven* and *selective-trace* algorithms to make use of latency of the digital network. That is, if a change of a logic signal occurs (called the *event*) at the gate input, the gate output needs to be evaluated. Otherwise, it is said to be *latent*, and no CPU time is spent on this gate. In fact, all gates (called the *fan-outs*) with input connected to the node, where an event occurred are scheduled to produce a transition at the output after the prescribed *delay time*. To control the logic events a *time queue* or *time wheel* is used. This kind of simulation is capable of detecting glitches, races or hazards. An important application of gate-level simulation is fault simulation performed in order to determine which potential circuit faults (usually *stuck-at* faults) could be detected by given test patterns. More subtle effects arising for MOS circuits like bi-directionality or charge sharing can also be modelled at the expense of more logic states introduced. However, there are two problems with this approach. These are: low accuracy of the obtained simulation results and extra work required to translate the design to a form suitable for the simulator. That is, in some cases extra gates must be added to the original gate structure to meet the simulator requirements. As a consequence, a drawback also appears during the layout verification due to inconsistency between the two databases [KON95].

For the MOS digital circuits the *switch-level* simulation is preferred. The switch-level logic simulators model a MOS circuit as a set of nodes connected by transistor switches [HAY82, BRY84, BRY87]. Node voltages are represented by a few discrete logic levels, usually 1, 0, X and U (uninitialised). The switch model of a transistor is controlled by a voltage level applied to its gate. Transistors are assigned discrete *strength values* that reflect channel conductances when fully on. Additionally, the nodes are considered to provide capacitive charges. Hence, they are assigned the so-called *sizes*. The power- and ground node are assigned the maximum size, whereas the other nodes smaller sizes accordingly to their relative capacitance values. Switch-level simulators usually partition the transistor network into channel-connected components (rather than into logic gates), which consist of transistors connected by drain- and source nodes. The signal paths they constitute (a path begins at power- or ground node) are then examined to determine node states. Like in a gate-level simulator, also here the event-driven algorithms are exploited. The interaction between different paths occurs only at transistor gate terminals. This approach allows to model mutually coupled elements (bi-directionality), charge sharing or pre-charged busses. Some rough timing specifications can be incorporated as well. However, precise timing information is not usually provided.

This has led to the development of *switch-level timing simulation*, sometimes referred to as switch-level simulation *oriented electrically* (rather than logically). Several approaches have been used to perform this kind of simulation [CHW75, TER83, KIM88, RAO89, VIS91, KON95]. The switch-level timing simulators seem to bridge the existing gap between electrical and logic simulation, since these simulators apply simplified (in different ways) methods of electrical simulation to digital networks. A variety of methods have been used to provide timing delays: table look up, empirical equations, RC trees, Asymptotic Waveform Evaluation (AWE) or inverter analysis techniques. More accurate results may be obtained with the so-called *variable-accuracy timing simulation* [TSA86, KIM88, VIS91] that also is considered as a kind of switch level simulation. A comprehensive overview of the switch-level simulators is given in [KON95].

Electrical simulation (or *circuit simulation*) provides the detailed timing waveforms of a circuit. It is mainly involved with the transistor level of a design. At this level a circuit is represented by RLC elements, diodes, transistors or voltage- and current sources. The circuit equations might be arranged with the modified nodal-analysis technique based on the Kirchhoff laws, and next solved with a stable implicit integration method followed by the Newton algorithm, finally supported by the sparse Gaussian elimination [NAG75]. Electrical simulation is the most accurate, but requires large memory resources and the longest CPU-time as compared to the other simulation levels. Besides, circuit simulation has to cover other needs, typical in designing of analogue circuits, such as DC characteristics, frequency behaviour or temperature analysis.

The different simulation levels can be compared as shown in Table 1.1 [SAL94]. The relative run-time cost and accuracy is provided for a hypothetical digital VLSI circuit. Clearly, the progression from behavioural level to electrical level provides an increase in accuracy at the expense of more CPU-time required. On the other hand, the maximum size of the circuit to be simulated decreases while the accuracy of simulation increases. This table reflects, to some extent, also the hierarchical designing process.

Apparently, the analogue circuits have not been addressed at the gate- and switch-level simulation. In fact, when extended to mixed A/D circuits, those levels are adequate to cover a class of analogue macromodels [BOY74, RUE78, RAB79, CAS91] as well, although circuit simulation technique is usually used for them. Observe that this gives rise to the *mixed-mode simulation*. On the other hand, in order to distinguish from the transistor level, the analogue macromodel-level might be considered too. Clearly, this classification is by no means strict.

Table 1. Relative cost and accuracy of different simulation levels

Level	Relative Cost	Capability and Accuracy
Behavioural	1	Algorithmic verification, some timing information
RTL/Functional	10	Functional verification, some timing information
Gate	100	Functional verification, first-order timing information
Switch	1000	Functional verification, first-order timing information
Timing	10000	Detailed waveform information, variable accuracy
Electrical	1000000	Most accurate form of simulation

There are several reasons for using two or more simulation levels simultaneously. One common situation arises for mixed signal A/D circuits, as mentioned above. In this case analogue portions of the circuit might be treated with a circuit (electrical) simulator, whereas the digital counterparts, with a logic or timing simulator. Clearly, an interface between different domains would be required for this *mixed-mode simulation*. Besides, an ideal simulator for VLSI circuits could be thought as one, which merges the speed and efficiency of logic simulators with the accuracy and detail of a circuit simulator. Usually, the detail and accuracy provided by the circuit simulator are required only for some critical portions of the circuit. That is, a simple gate-level or switch-level simulation is adequate to verify the most circuitry of the digital VLSI design, while some portions, such as sense amplifiers in memory circuits or sub-circuits, tightly coupled via transmission transistors, might require more detailed modelling- and simulation level.

Basically, the CPU-time can be substantially reduced with the mixed-mode simulation by choosing computationally less expensive models whenever it seems to be reasonable [SAL94]. Different levels of the design hierarchy can be addressed simultaneously. For example, at the RTL level the gate-level logic models can be used with no need of an extra interface. Hence, this kind of simulation is referred to as *mixed-level simulation*. Recently, also the *mixed-domain simulation* has been introduced, e.g. to simulate switched-capacitor circuits [CHA92, SAL96]. In this approach the Laplace *s*- or *z*-domain can be combined with the time domain.

1.2 Evolution of electrical simulation

The standard circuit simulation, as implemented in SPICE2 program [NAG75], has two drawbacks when applied to large circuits. First, the sparse-matrix solution time grows super-linearly with the size of the circuit. Second, different circuit variables tend to change at very different rates. The standard simulator forces then every differential equation to be discretized with the smallest time step relevant to the fastest-changing circuit variable. This approach becomes particularly inefficient, when most of the circuit variables are inactive i.e., are not changing at all.

Several techniques have been invented and applied to avoid solving large sparse matrices and allow different equations of the system to use individual time steps when integrated. These techniques fall in two classes: *direct decomposition* and *relaxation based* (or indirect). The direct methods (or *tearing*, as they are called) use the factorisation at linear- or nonlinear equation level,

and are aimed at retaining the convergence and stability properties of the standard circuit simulation [RAB79, HAJ80, DEM87]. However, the tearing decomposition methods seemed to influence the standard circuit simulation less than the relaxation-based techniques.

It was the orientation towards MOS technology that has fastened the role of the relaxation-based simulators [NEW84, WHI87]. In fact, they all exploit the key feature of MOS circuits, i.e. *unidirectionality*. MOS transistors are in principle unidirectional because the gate of the device is insulated from the drain and the source of the device. It means that, if the effects of small capacitances between the gate and the other terminals of the device are neglected, then the gate voltage is independent of the voltages at those terminals.

The relaxation technique was first used in the simulator MOTIS in 1975 [CHW75] although it was not recognised as a relaxation simulator at that time. In MOTIS the Jacobi-semi-implicit integration method was exploited and accompanied by table-look-up transistor models. The speed-up obtained, as compared to standard simulation, was up to two orders of magnitude with a relatively good accuracy to verify the timing of signals. Hence, it was called the *timing simulator*.

The improved MOTIS, MOTIS-C [Fan78], used the Seidel-semi-implicit integration algorithm. This simulator proved to be more efficient than MOTIS when the circuit equations were processed in the order that follow the signal flow in the circuit. However, in MOTIS-C the equations were ordered arbitrarily before being processed. The program SPLICE [NEW79] was the first, which introduced dynamic ordering of circuit equations via the event-driven and selective-trace algorithms adopted from gate-level logic simulation. A concept of the *analogue event* had to be applied.

Unfortunately, feedback-loop circuits or circuits with tightly coupled nodes (by floating capacitors or transmission transistors) exhibited instability when analysed with Gauss-Seidel or Gauss-Jacobi semi-implicit integration algorithms. Various methods have been investigated to overcome this drawback. The most important of them were suppose the *iterated timing analysis* (ITA) [SAL87] and *one-step-relaxation* (OSR) [HEN85], which implemented the Gauss-Seidel-Newton algorithms. ITA greatly improved the capabilities of nonlinear relaxation methods by carrying the outer relaxation loop until convergence as opposed to simple timing simulators. ITA was implemented in SPLICE3 simulator [SAL87]. OSR performed additionally carrying the inner linearisation loop until convergence. Speed-up factors up to two orders of magnitude have been reported for those simulators as compared to the conventional approach.

At the same time, it was also noted that relaxation techniques might be applied not only for the solution of algebraic nonlinear (or linear) equations but also directly the solution of differential equations describing the circuit. This gave rise to the *waveform relaxation technique* (WR) [LEL82], which solves equations with respect to variables that are functions of time (waveforms), unlike the other relaxation methods which compute vector of variables. The Gauss-Seidel and Gauss-Jacobi algorithms have been used in the waveform relaxation. The latter is particularly well suited to parallel computing [WHI85] implemented in RELAX2 simulator. WR algorithms proved to converge under mild assumptions when applied to MOS circuits. However, several refinements have been introduced to the WR to improve the convergence and reduce the CPU-time required [DEB87, WHI87]. These include partitioning, ordering, latency and windowing techniques. On a single processor the speed-up factor was shown to be up to an order of magnitude for large circuits as compared to standard simulation.

A key aspect of decomposition-based simulation techniques is that the computational overhead needed rises almost linearly with the problem size, whereas the standard simulation features an exponential dependence. Various decomposition approaches reflect also a kind of hierarchy in circuit description as shown in Fig.1.1 [VAC93].

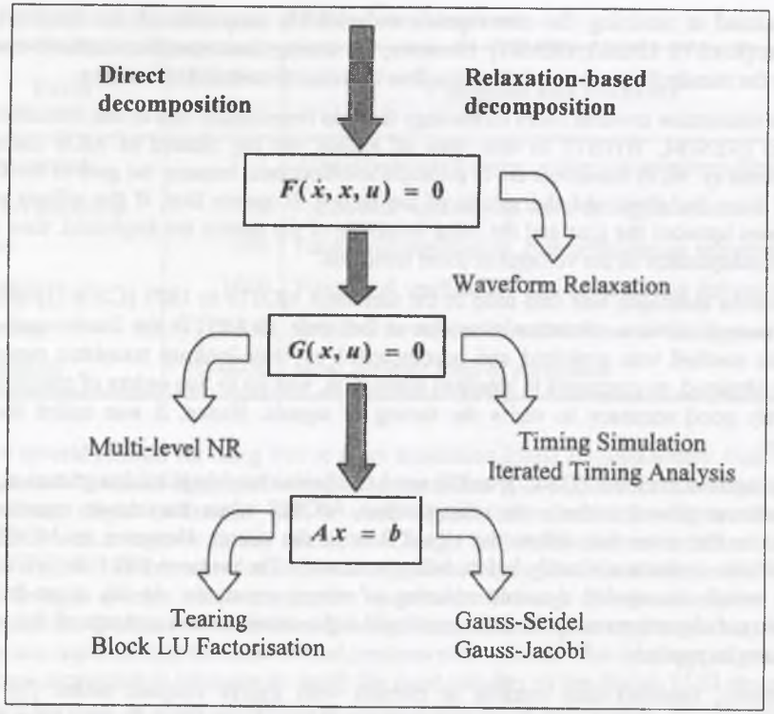

Fig.1.1. Application of decomposition to circuit simulation

Observe that a hole is present on the direct decomposition side at the level of differential equations, because at this level it becomes similar to relaxation-based (indirect) decomposition.

Another important aspect involved with simulation is the *macromodelling* [BOY74, RUE78, RAB79, BRO88, CAS91]. The primary goal of macromodelling is to reduce substantially the CPU-time of simulation. Clearly, some reasonable constraints had to be imposed on macromodelling, such as qualitative and quantitative similarity feature, good numerical possedness and predictive ability [CHU80]. As a result, macromodelling provides a trade-off between speed and accuracy. There are several approaches that can be used in macromodelling: network reducing/simplification, approximating of terminal characteristics, curve and table fitting or mathematical functional modelling (build-up technique). Almost all simulation levels exploit macromodelling, although in some cases it is not emphasised, so that the modules (subcircuits) used are simply referred to as "models" rather than "macromodels", like in case of gate-level simulation.

1.3 PWL simulation

Several simulators make use of *piecewise linear* (PWL) techniques. The primary goal of PWL techniques is to avoid the linearisation step i.e., the Newton-Raphson iterations used in standard circuit simulation. Several algorithms are available for this purpose [CHU75, CHU86, BOK87]. Basically, each nonlinear element is represented by PWL equations so the simulation algorithm only deals with linear systems of equations. Careful monitoring of changes of PWL regions to

avoid numerical inconsistency becomes an important task. Due to linear models (equations) this simulation has better convergence than standard simulation, and supports component modelling at all levels of abstraction.

The PWL approach has also been used for waveforms to improve the efficiency of signal representation in mixed-mode and mixed-level simulators. Additionally, unification of models could be exploited in some cases.

Van Stiphout et al. implemented a PWL mixed-signal simulator PLATO that used an event-driven approach for transient analysis [VAN90]. The events are grouped into two classes: PL events and dynamic events. PL events occur when a vector reaches the end of a PWL region. 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 discretisation of the event times and thereby reducing the number of separate events. Further improvements to efficiency were made by forcing the related circuit blocks to use the same minimum step size, reducing the recomputation of step sizes required.

Kevenaar and Leenaerts use similar methods to PLATO in their simulator called PLANET that exploits system hierarchy to run more efficiently [KEV91]. Unlike previous PWL simulators, PLANET partitions a 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 time steps to be used. The subcircuits are connected together by a set of topological equations describing the system hierarchy. The advantages of this approach over non-hierarchical methods increase with the complexity of the simulated system. Some comparison to SPICE estimates for a simple OpAmp macromodel are given.

Kruiskamp and Leenaerts used PLANET for simulating circuits with PWL macromodels they derived [KRU96]. Analogue and A/D circuits are addressed at transistor- and macromodel level. A behavioural approach is used to obtain the macromodels. Because a similar data format is used for all models, mixed-level simulation can be performed by applying a single simulation algorithm.

Rsim is an experimental switch-level simulator that was developed by Terman [TER83]. Rsim is capable of simulating large MOS circuits up to three orders of magnitude faster than SPICE. Kao and Horowitz added PWL models to Rsim to form a new simulator called Mom [KAO94], which preserves the efficiency of Rsim for digital circuits but improves the accuracy for more "difficult" subcircuits. These include CMOS dynamic RAM sense amplifiers, ECL logic gates or BiCMOS buffers. For such circuits Mom is as good as a mixed-mode simulator combining circuit simulation with switch-level timing simulation.

Griffith and Nakhla have used PWL waveforms in a novel simulator for nonlinear frequency dependent circuits [GRI92]. This simulator is designed for transient analysis of high-speed (microwave) circuits where improperly terminated connections can adversely affect the transmission of signals. The connections cannot be simulated correctly by conventional lumped impedance interconnect models and must use distributed transmission line models instead. These models are only defined in the frequency domain. The simulator replaces nonlinear terms in the circuit equations by a set of PWL time-dependent waveforms. This reduces the nonlinear equations to a linear equation that can be solved in the frequency domain. The transient response is obtained from frequency domain solution using inverse Laplace transform.

Cottrell used PWL waveforms to create a behavioural mixed-signal simulator [COT90]. Its analogue models represent the transfer functions associated with analogue functional blocks such as amplifiers, filters, comparators and D/A converters. The model ports are unidirectional and are

either classified as inputs or outputs. Passive RLC elements are not allowed. Analogue signals are classified as voltage or current PWL waveforms. Transfer functions specified in the frequency domain are converted to the time domain, using inverse Laplace transform before simulation. Then the responses are solved with Forward Euler integration algorithm and local truncation error control. Each model can use an individual time step. The local truncation error reflects the second derivative of the waveform that is next simplified to PWL form. A speed-up of two orders of magnitude as compared to SPICE has been reported.

Visweswariah and Rohrer have used PWL and piecewise constant waveforms in a prototype event-driven simulator called SPECS [VIS91]. The simulator uses empirical table models of I-V characteristics to represent electronic devices. Voltages are PWL waveforms while currents are represented by piecewise constant waveforms. The simulator assumes that circuit only consists of voltage-dependent current sources and linear capacitors. All the table models are evaluated with a single time queue. Events correspond to the time instants that current waveforms change level and voltage waveforms change rate. Feedback loops cause events to be rescheduled resulting thereby in occurrence of iterations. A mechanism to assure convergence of the iterations has been introduced. For digital MOS circuits the simulator was 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 multi-level simulator [RUA91], This simulator was designed to operate with both analogue and digital functional blocks. It uses an event-driven approach and predicts the time of a new event from the gradient of signal waveforms. Logic gate models operate directly on PWL waveforms, and are provided with an adequate inertial delay mechanism including capacitive loading effect. The delay depends also on input rate. Since unknown logic states are avoided by using continuous PWL waveforms, the digital circuitry can be connected directly to the analogue portion of the circuit, and no signal conversion is required for this simulator. Interval algebra is used to evaluate the initial conditions in a circuit. The simulation results presented are optimistic, however, more complicated digital blocks can be modelled only at a gate level. Besides, the timings that the analogue models provide seem to be rather rough, despite the fact they can strongly influence the overall circuit behaviour.

The latter work was the primary inspiration to the author, and to overcome the drawbacks observed, a new PWL simulator has been proposed in [DAB94] and next improved in [DAB95, DAB99F]. It is a prototype functional-level simulator provided with efficient analogue macromodels, which model timing specifications and basic nonlinarities of the functional units [DAB96K]. In contrast to other PWL simulators, mentioned above, an explicit analysis technique based on approximation is proposed to proceed the macromodels. Complex digital units are allowed as behavioural logic models, since the PWL approach cannot directly be used for them. A virtual interface between PWL- and digital domain is defined within the simulator. Since the PWL macromodels save the CPU-time, this simulation technique can be also used in iterative processes required for simulation of feedback loop structures [DAB97W, DAB99W]. Most of those approaches have been implemented in VHDL as well [DAB98D, DAB98E, DAB99].

1.4 Outline of the monograph

The underlying objective of the work presented was to develop analogue macromodels and a supporting simulation technique capable of verifying effectively complex electronic designs at the functional level. The models were expected to represent basic timing/frequency specifications and be computationally as efficient as possible, approaching to some extent the models used in logic simulation. In order to verify the functionality of mixed-signal A/D designs, the prospective macrosimulation technique and the logic simulation were assumed to be compatible.

As a result, simple, yet accurate models have been derived, and explicit formulas have been proposed to evaluate their timing responses. As compared to the standard analogue simulation a number of small integration steps usually produced to solve for the involved differential equations, have been replaced (whenever possible) with a few linear segments due to the accuracy imposed. Consequently, with the reduced number of timing points, those models might be viewed as discrete objects, and hence, could be treated like gates in logic simulation using the event-driven analysis technique.

The author's work cited in the previous section is summarised and discussed thoroughly in the following chapters of this monograph. It was intent of the author to provide a clear and unified description covering possibly all issues of the PWL approach to macrosimulation at the functional level. Some new results, not published so far, such as Section 2.4 or Chapter 6, are also included.

The organisation of the book is as follows. In Chapter 2, a concept of the PWL approach is introduced. The nonlinear characteristics of analogue units are assumed to be of PWL type. A class of unidirectional basic building blocks is used to model the transient effects. Basic integration algorithms common in analogue simulation are investigated for analysing effectively the building blocks. To overcome the drawbacks observed a novel method that converts the smooth time responses of those blocks into PWL waveforms is proposed. For this purpose a non-iterative approximation algorithm is derived. A refinement patterned after the PWL approximation algorithm is added to the standard trapezoidal rule, so that it can be exploited as an alternative tool. The PWL approximation- and the trapezoidal rule-based techniques are compared, and their pros and cons are discussed. Finally, also other building blocks useful in analogue modelling are addressed.

In Chapter 3, the PWL macromodels of simple functional units, such as comparator or amplifier are derived. Second order effects relevant to their timing specifications are accounted for. The model performances are compared with the respective SPICE estimates.

Mixed-mode simulation using the PWL technique is addressed in Chapter 4. A concept of the PWL event is introduced and the event-driven simulation algorithm is formulated. It is based on the event scheduler- and time-queue concept, primarily only used in logic simulation. Interfacing between the PWL- and logic domain is also discussed. The chapter is accomplished by a simulation example obtained with the prototype mixed-mode simulator.

In Chapter 5, feedback loop structures are shown to be critical for the PWL simulation in a sense that they may require iterations. To support the PWL technique the waveform relaxation (WR) is used, since it also operates on waveforms. The convergence and stability properties of the WR-based PWL algorithm are considered. A few theorems are formulated and proved resulting in a convergence criterion of practical use. The simulation examples are included as well.

Chapter 6 refers to the synthesis problems arising with higher-order analogue models. An effect of the PWL error accumulation is emphasised in this chapter. To compensate for those errors a refinement is introduced to the PWL model and followed by the corresponding modifications in the approximation algorithm. Also the second-order building blocks are introduced to simplify the synthesis and reduce the approximation errors. Parallel and tuned cascade structures are proposed to perform the synthesis. In case of cascade structures the trapezoidal rule-based PWL algorithm is found to be more accurate than its approximation-based PWL counterpart.

Chapter 7 is concerned with implementation of the PWL models and algorithms in VHDL. Since the PWL models may be viewed as discrete objects, they are well suited to be implemented in this discrete environment. The basic building blocks are defined as behavioural VHDL models (entities) based on the explicit formulas available for those blocks. More complex PWL models are supported by structural approach. Besides, the digital nature of VHDL facilitates modelling mixed signal A/D networks. However, additional signal conversions between the PWL- and standard logic domain are required in this case. Practical simulation results illustrate this approach.

Chapter 8 provides final conclusions and a summary to all the chapters.

2. PWL MACROSIMULATION TECHNIQUES

Two main issues, model representation and signal processing, should be addressed to characterise an approach to macrosimulation technique. At the functional simulation level, analogue units like amplifiers, adders, comparators or D/A converters are dealt with rather than transistors or circuit primitives, e.g.: resistors, capacitors, etc. Hence, the functional-level models make usually use of basic algebraic operators to characterise the static behaviour, and simple inertial blocks or integrators to perform the transient effects.

In this chapter a concept of *piecewise linear* (PWL) macrosimulation technique is developed. An analogue network (sub-system) is assumed to be composed of the so-called *basic building blocks*. In the presented approach each of these blocks is usually provided with some algebraic operator and an inertial (or integration) mechanism that together form a *unidirectional* unit. This is adequate not only because of the functional level addressed here, where the bi-directional loading effects might be neglected, but also because of the MOS technology commonly used when designing complex A/D networks. It is well known that the input impedance of a MOS transistor gate is high, and so is the input impedance of any functional block of a system in typical cases.

Using additionally some explicit formulas to evaluate the timing responses of those blocks, a similarity between such a problem statement and the logic simulation can be observed. In fact, logic simulation proved to be the most efficient computationally as compared to other simulation techniques. Hence, it is useful to adopt the basic highlights of logic simulation, such as *event*, *selective trace* and *time queue* to the macrosimulation technique derived. As those issues have already been introduced in mixed-mode simulation [SAL94] (as mentioned in Section 1.2), the presented approach is not entirely new. Here, however, the simulation becomes even more CPU-effective because of the PWL technique used (both for DC characteristics and signal representation), the unidirectionality assumption and timing analysis performed with explicit formulas as opposed to standard approaches used for sets of differential equations.

In order to take advantage of the mentioned above mechanisms as much as possible the analogue signals are postulated to take the PWL form. In fact, this choice is as a kind of tradeoff between the number of evaluations (points computed) per block output, and the accuracy required to match shapes of the output signals. Apparently, the PWL representation of a smooth waveform is more accurate than the step function-based representation, and as compared to the quadratic case fewer computations per point are required. Besides, at this level of abstraction it seems reasonable to sacrifice some accuracy for the reduced number of integration steps to be done, unless unrealistic results are produced. As a consequence, the resulting time steps are expected to be relatively large, so the number of steps to be done should be relatively small.

From the broader perspective, the application of the event-driven mechanisms adopted from logic simulation to simulation of an analogue network may be viewed as a relaxation technique, where the blocks are analysed due to signal flow. In particular, using the selective trace approach corresponds to the Gauss-Seidel algorithm in a sense that the component blocks are taken for analysis separately. These terms will be explained in Chapter 4 in context of using a mixed technique (PWL/logic) to simulation of A/D networks.

2.1 Concept of PWL macrosimulation

In most cases the real analogue units consist of subcircuits that exhibit inertial or integrating properties. Therefore, any macromodel of an analogue unit can consist of a few such building blocks to mimic the timing behaviour, the DC transfer function and the output loading effects.

Assume the constitutive relation for the basic building block in the form of

$$T\dot{x} + x = f(x_{inp}) \tag{2.1}$$

where $f(\cdot)$ denotes its static (DC) characteristic represented as a PWL function, and T is the time constant (Fig.2.1). To obtain a pure static block, its dynamics have to be removed by letting T=0 in eqn.(2.1). On the other hand, when capacitive loading effects at the output x must be accounted for, the time constant takes the form of

$$T = R_s(C_o + C_s) \tag{2.2}$$

where C_o is a charging capacitance at the output and C_s, R_s are respectively the internal output capacitance and resistance. Equation (2.1) can be generalised into a multiple input case, e.g. for a multiplier or adder.

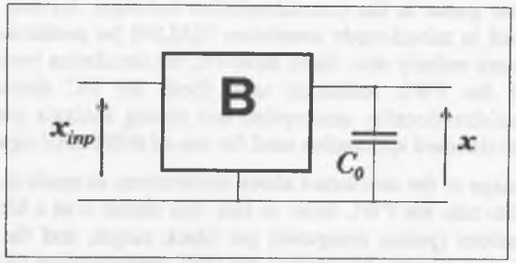

Fig. 2.1. Basic building block as two-port

Using a PWL signal x_{inp}, a superposition $f[x_{inp}(t)]$ provides some signal $u(t)$, which is of PWL form too. Hence, u consists of some PWL segments, each limited by two subsequent breakpoints t_i, t_{i+1} and defined as: $u_i(t) = u_{i0} + r_i \cdot (t - t_i)$, $t \in [t_i, t_{i+1}]$. Observe that those breakpoints have their origin in the breakpoints either of x_{inp} or of the $f(\cdot)$ function.

A number of integration algorithms are available to provide a solution for (2.1) [CHU75, OGR94]. However, as the analysis procedure has been assumed to work with relatively large time steps, only the so-called A-stable integration methods are well suited for this purpose. The most popular of them (used in standard analogue simulation) are backward-Euler (**BE**), trapezoidal rule (**TR**) and 2nd order

Gear algorithm (**G2**). In fact, most circuit simulators use these algorithms for their guaranteed stability properties so that their step sizes can be selected based on accuracy estimates only. For the purpose of analysing the structure of blocks, each defined by (2.1), those algorithms are expected to take preferably an explicit form of

$$x_k = g[x_{k-1}, u_i(t_k), h_k] \tag{2.3}$$

or similar to it, where x_k, x_{k-1} are estimates of the exact solution $x(t_k)$, $x(t_{k-1})$, accordingly. The integration step $h_k = t_k - t_{k-1} > 0$ and $t_k, t_{k-1} \in [t_i, t_{i+1}]$. Besides, h_k should be calculated with respect to the accuracy imposed.

Recently, also a novel, so-called *PWL approximation algorithm* has been proposed to solve explicitly for (2.1), like (2.3) does [DAB95, DAB99F]. The PWL algorithm is based on the exact solution of (2.1) and makes use of a unique non-iterative approximation procedure to provide the relevant timing response in a PWL form rather than the original smooth waveform.

Consequently, all the analogue signals propagating through a network are assumed to be of the PWL form, and any explicit analysis procedure (of the mentioned above) may be applied subsequently to the component blocks, like in gate-level logic simulation. As will be shown, the mechanisms typical of logic simulation may be adopted in those PWL approaches to assure maximum effectiveness.

In this context, in the remaining of this work, any model composed of the basic building blocks (block) with PWL input- and PWL output waveforms, obtained by discretisation, will be referred to as the *PWL model.* In fact, the PWL output of each building block is obtained by discretisation. In case of the PWL approximation-based algorithm the smooth solution of (2.1) is discretised, and hence the relevant PWL waveform is generated. Unlike this, standard numerical algorithms used in PWL modelling discretise the differential equation, so that the relevant discrete solution is provided directly.

In the following sections, first the application of the mentioned above standard algorithms (BE, TR, G2) to solve for (2.1) is investigated. In this case the obtained discrete points are assumed to be breakpoints constituting the PWL output waveform. The local truncation error mechanism is used to control the accuracy of that PWL solution, like in typical applications. The TR algorithm is shown to be well suited to work with relatively large steps as opposed to the BE and G2. Next, the new PWL approximation-based technique is derived in detail. A refinement patterned after the PWL approximation algorithm is added to the TR, so that it can perform large steps too, and can serve as an alternative PWL approach. The PWL approximation- and the TR-based techniques are compared in accuracy and the computational overhead required. Finally, other macrosimulation issues pertaining to additional building blocks such as integrator or multiplier are addressed.

2.2 PWL simulation by standard algorithms

In order to investigate the application of the mentioned above standard algorithms to solve for (2.1), first we invoke the relevant recurrent formulas [CHU75, OGR94]:

$$x_k = h_k \dot{x}_k + x_{k-1} \qquad \text{for BE algorithm,} \tag{2.4a}$$

$$x_k = \frac{h_k}{2}(\dot{x}_k + \dot{x}_{k-1}) + x_{k-1} \qquad \text{for TR algorithm,} \tag{2.4b}$$

$$x_k = \beta_0 \dot{x}_k + \alpha_1 x_{k-1} + \alpha_2 x_{k-2} \quad \text{for G2 algorithm,} \qquad (2.4c)$$

where $\beta_0 = \dfrac{h_k(h_k + h_{k-1})}{2h_k + h_{k-1}}$, $\alpha_1 = \dfrac{(h_k + h_{k-1})^2}{h_{k-1}(2h_k + h_{k-1})}$, $\alpha_2 = \dfrac{-h_k^2}{h_{k-1}(2h_k + h_{k-1})}$,

and the integration steps are: $h_k = t_k - t_{k-1}$, $h_{k-1} = t_{k-1} - t_{k-2}$. Consequently, x_k, x_{k-1}, x_{k-2} are estimates of the exact values of $x(t_k)$, $x(t_{k-1})$, $x(t_{k-2})$, respectively. A similar relation holds for the time derivatives as well. The formulas (2.4) could be instantiated for the basic building block and put into the form like (2.3) as follows.

Assuming $\dot{x}_k \approx \dot{x}(t_k)$, $x_k \approx x(t_k)$, and by definition $f[x_{inp}(t_k)] = u_k$, from (2.1) one obtains

$$\dot{x}(t_k) = \frac{f[u_{inp}(t_k)] - x(t_k)}{T} \approx \frac{u_k - x_k}{T} = \dot{x}_k,$$

so that:

$$x_k = \frac{x_{k-1} + \frac{h_k}{T} u_k}{1 + \frac{h_k}{T}} \quad \text{for BE algorithm,} \qquad (2.5a)$$

$$x_k = \frac{(1 - \frac{h_k}{2T})x_{k-1} + \frac{h_k}{2T}(u_{k-1} + u_k)}{1 + \frac{h_k}{2T}} \quad \text{for TR algorithm,} \qquad (2.5b)$$

$$x_k = \frac{\alpha_1 x_{k-1} + \alpha_2 x_{k-2} + \frac{\beta_0}{T} u_k}{1 + \frac{\beta_0}{T}} \quad \text{for G2 algorithm} \qquad (2.5c)$$

The formulas (2.5) are of practical use when supported by the local truncation error (LTE) mechanism to control the actual step size h_k. The LTE is defined in general as $\varepsilon_k = x_k - x(t_k)$, and in terms of the Lagrange rest of the Taylor function expansion (provided the former point x_{k-1} matches perfectly the exact value $x(t_{k-1})$) it equals:

$$\varepsilon_k = \frac{h_k^2}{2} x^{(2)}(\theta_k) \quad \text{for BE algorithm,} \qquad (2.6a)$$

$$\varepsilon_k = \frac{h_k^3}{12} x^{(3)}(\theta_k) \quad \text{for TR algorithm,} \qquad (2.6b)$$

$$\varepsilon_k = \frac{3\beta_0 h_k^2 - \alpha_2 h_{k-1}^3 - h_k^3}{6} x^{(3)}(\theta_k) \quad \text{for G2 algorithm,} \qquad (2.6c)$$

where $\theta_k \in (t_{k-1}, t_k)$. As the precise value of θ_k is unknown, a commonly used practice is a substitution $\theta_k = t_{k-1}$. Consequently, the required step size h_k can be obtained from (2.6) by putting the assumed accuracy ε_0 for ε_k e.g., $h_k = \sqrt{\dfrac{2\varepsilon_0}{|x^{(2)}(t_{k-1})|}}$ for the BE algorithm, where

$$x^{(2)}(t_{k-1}) = \frac{\dot{u}(t_{k-1}) - \dot{x}(t_{k-1})}{T} \approx \frac{r_{k-1}T - u_{k-1} + x_{k-1}}{T^2}.$$

Apparently, in case of (2.6*a*) and (2.6*b*) solving for h_k is a simple task, whereas it appears cumbersome for (2.6*c*). To cope with this problem first rewrite (2.6*c*) using the definition for β_0 and α_2 (shown above) as

$$\varepsilon_k = \frac{h_k^2(h_k + h_{k-1})^2}{6(2h_k + h_{k-1})} x^{(3)}(\theta_k) \qquad (2.6c')$$

The step size for G2 algorithm can be found directly from (2.6*c* ') for the assumed accuracy $\varepsilon_0 = \varepsilon_k$ and $x^{(3)}(\theta_k) \approx \dfrac{d}{dt}\left(\dfrac{\dot{u}(t_{k-1}) - \dot{x}(t_{k-1})}{T}\right) \approx \dfrac{-r_{k-1}T + u_{k-1} - x_{k-1}}{T^3}$ using e.g. the Newton-Raphson iterations. However, this approach is not computationally efficient, so a simpler method would be useful.

For this purpose, here a two-step predictor-corrector method is proposed. In particular, the predictor assumes fixed step i.e., $h_k = h_{k-1}$. Hence, $\varepsilon_k = \dfrac{2h_k^3}{9} x^{(3)}(\theta_k)$ and the predicted step h_{k0} may be found from

$$h_{k0} = \sqrt[3]{\frac{4.5\,\varepsilon_0}{|x^{(3)}(t_{k-1})|}}.$$

Next, the corrector evaluates the resulting LTE based on (2.6*c* ')

$$\varepsilon_{k0} = \frac{h_{k0}^2(h_{k0} + h_{k-1})^2}{6(2h_{k0} + h_{k-1})} x^{(3)}(t_{k-1})$$

and provides the actual step h_k from the relation

$$\left(\frac{h_k}{h_{k0}}\right)^3 \approx \frac{\varepsilon_0}{\varepsilon_{k0}}.$$

Now, we apply the above formulas to analysing the basic building block (2.1) for some test signals. The unit step $u(t) = 1(t)$ that is a special case a PWL segment, and the normalised linear input $u(t) = t \cdot 1(t)$ are used. For brevity the time constant T is normalised too.

The relevant simulation results are presented in tables 2.2*a-c* and 2.2*a-c* for the accuracy of $\varepsilon_0 = 0.05$ and $\varepsilon_0 = 0.1$, respectively. Note that $\varepsilon_k = x_k - x(t_k)$ is here, in fact, a total error rather than the local truncation error, since except of the first step $x_{k-1} \neq x(t_{k-1})$. Additionally, for G2 algorithm the LTE estimate is presented based on (2.6*c'*). Observe also that the G2 algorithm needs support in the first step because x_{k-2} is not available in this case. The TR is used for this purpose.

Apparently, the BE algorithm produces much smaller steps than the other two techniques. On the other hand, the second order algorithms TR and G2 are oscillatory inclined. In case of step responses, the overshoots or ringing can be observed for them. In a physical sense, this may be viewed as an unrealistic result. Detailed comparison shows those effects to be much smaller for the TR approach. Besides, the TR does not accumulate errors ε_k (as opposed to G2); it produces the largest steps, and is computationally more efficient than the two-step G2 algorithm.

The shaded cells (in the tables) emphasise error values that are bigger than ε_0 or correspond to unrealistic, in a physical sense, results i.e., overshoots or ringing in the step response. As shown, the TR algorithm is best suited for the timing analysis performed as compared to the other two methods. Because of it in the remaining of the monograph it is preferred over the BE and G2 methods. Moreover, observe that the total errors ε_k of the TR are much smaller than ε_0 (Tables 2.1*b* and 2.2*b*). Hence, the TR might be expected to keep the accuracy imposed with even larger steps, if a more precise step control mechanism (based so far on (2.6*b*)) were used for it. A solution of this problem is proposed in Section 2.4.

Table 2.1*a*. Time responses of test block obtained with **BE** algorithm for $\varepsilon_0 = 0.05$

$u = 1(t)$, $x(0) = 0$						$u = t \cdot 1(t)$, $x(0) = 1$					
k	t_k	h_k	x_k	$x(t_k)$	ε_k	k	t_k	h_k	x_k	$x(t_k)$	ε_k
1	0.316	0.316	0.240	0.271	-0.031	1	0.224	0.224	0.858	0.823	0.035
2	0.679	0.363	0.422	0.493	-0.050	2	0.468	0.244	0.813	0.720	0.092
3	1.102	0.423	0.608	0.668	-0.060	3	0.741	0.273	0.798	0.694	0.103
4	1.607	0.505	0.740	0.800	-0.060	4	1.049	0.308	0.857	0.749	0.107
5	2.227	0.620	0.840	0.892	-0.052	5	1.401	0.352	0.999	0.894	0.105
6	3.018	0.791	0.911	0.951	-0.040	6	1.820	0.409	1.234	1.137	0.097
7	4.078	1.060	0.957	0.983	-0.026	7	2.296	0.486	1.581	1.497	0.084
8	5.603	1.525	0.983	0.996	-0.013	8	2.888	0.592	2.067	2.000	0.067
9	8.028	2.425	0.995	1.000	-0.005	9	3.635	0.747	2.738	2.688	0.050

Table 2.1*b*. Time responses of test block obtained with **TR** algorithm for $\varepsilon_0 = 0.05$

$u = 1(t)$, $x(0) = 0$						$u = t \cdot 1(t)$, $x(0) = 1$					
k	t_k	h_k	x_k	$x(t_k)$	ε_k	k	t_k	h_k	x_k	$x(t_k)$	ε_k
1	0.843	0.843	0.593	0.570	0.023	1	0.669	0.669	0.666	0.693	-0.027
2	1.981	1.138	0.888	0.862	0.026	2	1.513	0.844	0.918	0.954	-0.035
3	3.731	1.750	0.993	0.976	0.017	3	2.654	1.140	1.764	1.794	-0.030
4	8.140	4.409	1.003	1.000	0.003	4	4.408	1.755	3.415	3.432	-0.017
5	13.988	5.848	0.999	1.000	-0.001	5	8.817	4.409	7.814	7.817	-0.003
6	22.422	8.434	1.001	1.000	0.001	6	14.665	5.848	13.667	13.665	0.002

Table 2.1*c*. Time responses of test block obtained with **G2** algorithm for $\varepsilon_0 = 0.05$

$u = 1(t)$, $x(0) = 0$							$u = t \cdot 1(t)$, $x(0) = 1$						
k	t_k	h_k	LTE_k	x_k	$x(t_k)$	ε_k	k	t_k	h_k	LTE_k	x_k	$x(t_k)$	ε_k
1	0.843	0.843	-	0.593	0.570	0.023	1	0.669	0.669	-	0.666	0.693	-0.027
2	1.654	0.816	0.050	0.859	0.810	0.051	2	1.265	0.596	-0.051	0.771	0.829	-0.059
3	2.917	1.258	0.048	1.008	0.946	0.062	3	2.069	0.804	-0.048	1.244	1.322	-0.078
4	6.481	3.564	-0.047	1.061	0.998	0.063	4	3.227	1.158	-0.048	2.226	2.306	-0.080
5	7.728	1.247	-0.060	1.033	1.000	0.033	5	10.965	7.738	0.047	9.879	9.965	-0.107
6	9.792	2.064	-0.048	1.007	1.000	0.007	6	11.738	0.773	0.082	10.675	10.738	-0.063
7	13.261	3.469	-0.048	0.997	1.000	-0.003	7	13.477	1.739	0.047	12.466	12.477	-0.012

Table 2.2*a*. Time responses of test block obtained with **BE** algorithm for $\varepsilon_0 = 0.1$

$u = 1(t),\ x(0) = 0$						$u = t\cdot 1(t),\ x(0) = 1$					
k	t_k	h_k	x_k	$x(t_k)$	ε_k	k	t_k	h_k	x_k	$x(t_k)$	ε_k
1	0.447	0.447	0.309	0.361	-0.052	1	0.316	0.316	0.836	0.774	0.062
2	0.985	0.538	0.551	0.627	-0.076	2	0.679	0.363	0.794	0.693	0.101
3	1.652	0.667	0.731	0.808	-0.077	3	1.103	0.424	0.886	0.767	0.119
4	2.514	0.862	0.856	0.919	-0.063	4	1.608	0.505	1.129	1.009	0.120
5	3.693	1.179	0.934	0.975	-0.041	5	2.228	0.620	1.549	1.443	0.106
6	5.434	1.741	0.976	0.996	-0.020	6	2.967	0.739	2.178	2.070	0.108
7	8.321	2.887	0.994	1.000	-0.006	7	3.941	0.974	3.047	2.979	0.068
8	14.095	5.774	0.999	1.000	-0.001	8	5.315	1.374	4.359	4.324	0.035
9	28.237	14.142	1.000	1.000	0.000	9	7.447	2.132	6.461	6.448	0.013

Table 2.2*b*. Time responses of test block obtained with **TR** algorithm for $\varepsilon_0 = 0.1$

$u = 1(t),\ x(0) = 0$						$u = t\cdot 1(t),\ x(0) = 1$					
k	t_k	h_k	x_k	$x(t_k)$	ε_k	k	t_k	h_k	x_k	$x(t_k)$	ε_k
1	1.063	1.063	0.694	0.655	0.039	1	0.843	0.843	0.657	0.704	-0.047
2	2.640	1.577	0.964	0.929	0.035	2	2.063	1.165	1.263	1.317	-0.054
3	5.858	3.218	1.008	0.997	0.011	3	3.880	1.817	2.890	2.921	-0.031
4	11.171	5.313	0.996	1.000	-0.004	4	8.812	4.932	7.808	7.813	-0.015
5	17.865	6.694	1.002	1.000	0.002	5	15.506	6.694	14.508	14.506	0.002

Table 2.2*c*. Time responses of test block obtained with **G2** algorithm for $\varepsilon_0 = 0.1$

$u = 1(t),\ x(0) = 0$							$u = t\cdot 1(t),\ x(0) = 1$						
k	t_k	h_k	LTE_k	x_k	$x(t_k)$	ε_k	k	t_k	h_k	LTE_k	x_k	$x(t_k)$	ε_k
1	1.063	1.063	-	0.694	0.655	0.039	1	0.843	0.843	-	0.657	0.704	-0.047
2	2.217	1.154	0.099	0.973	0.891	0.082	2	1.659	0.816	-0.100	0.940	1.040	-0.099
3	5.173	2.956	0.094	1.100	0.994	0.106	3	2.919	1.260	-0.096	1.902	2.027	-0.125
4	6.603	1.430	-0.113	1.055	0.999	0.057	4	6.401	3.482	0.094	5.280	5.404	-0.125
5	8.765	2.162	-0.096	1.013	1.000	0.013	5	7.661	1.260	0.012	6.596	6.662	-0.066
6	12.306	3.541	-0.096	0.996	1.000	-0.004	7	9.733	2.072	0.096	8.720	8.733	-0.013

2.3 Simulation by PWL approximation technique

Recently, a new algorithm has been proposed to calculate a PWL response for the building blocks used, neither with overshoots nor ringing [DAB95, DAB99F] as happens for standard algorithms with large steps. To give insight in this approach assume that $x(0) = x_0$ and for a given input segment $f[x_{inp}(t)] = u(t) = u_0 + rt,\ t \in [0, t_{mx}]$. Hence, solving for eqn. (2.1) gives an explicit formula for x consisting of the transient and the steady state components

$$x(t) = (x_0 - u_0 + rT)e^{-t/T} + r(t - T) + u_0 \tag{2.7}$$

The main objective here is to get a PWL approximation of (2.7) to enable further propagation of the signal x in a linearised form.

For this purpose first split the time interval $[0, t_{mx}]$ into subintervals $[0, t_1]$, $[t_1, t_2]$, ... $[t_n, t_{mx}]$. For each subinterval $[t_i, t_{i+1}]$ a segment of a PWL approximating signal x_{lin} is defined by its end points that are assumed to lie on the curve x. Hence $x_{lin}(t_i) = x(t_i)$ and $x_{lin}(t_{i+1}) = x(t_{i+1})$. In fact, given t_i, the time instant t_{i+1} has to be found (Fig.2.2).

To control the accuracy of this approximation, the Chebyshev measure may be used. It has been found the most advantageous to develop an efficient approximation algorithm presented below. Consequently, our objective can be formulated as an optimisation task, that is to maximise the distance $d = t_{i+1} - t_i$ with some constraints and given t_i:

$$\underset{t_{i+1}}{Maximise}\ d : \left\{ d = t_{i+1} - t_i,\ \ t_{i+1} \le t_{mx} \right\} \tag{2.8a}$$

$$p(t_i, t_{i+1}) = \underset{t \in (t_i, t_{i+1}]}{Max} \left| x(t) - x_{lin}(t) \right| \tag{2.8b}$$

$$p(t_i,\ t_{i+1}) \le p_{mx} \tag{2.8c}$$

where $p(t_i, t_{i+1})$ is the performance index and p_{mx} is an arbitrarily chosen constant (approximation accuracy).

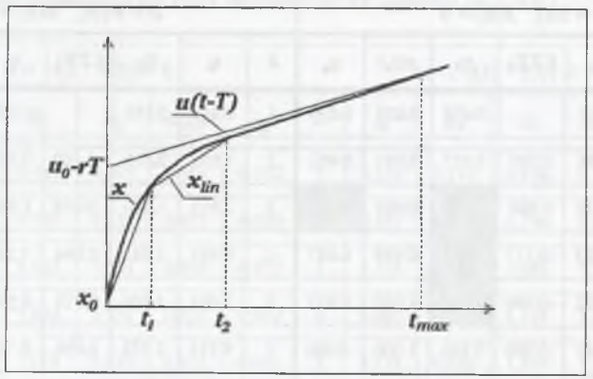

Fig. 2.2. PWL approximation of output signal

The approximation process may be viewed also as a kind of signal conversion presented in Fig.2.3. The first block in this diagram is usually the mentioned above inertial building block (or integrator) defined by (2.1), whereas the other one is the approximator. The latter outputs the PWL segments, which are defined by subsequent breakpoints $[t_k, x(t_k)]$, $[t_{k+1}, x(t_{k+1})]$, ... In this case a formal notation would be $x_{lin} = L(x)$. The structure shown will also be referred to as the *approximation-based PWL model*.

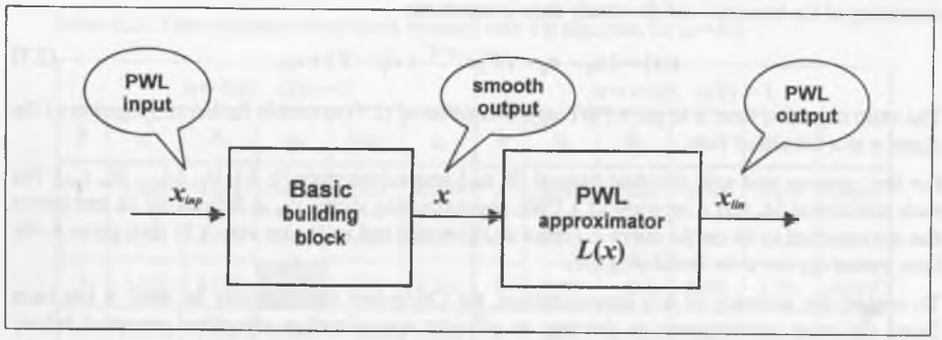

Fig. 2.3. Block structure of approximation-based PWL model

Consequently, for a network modelled with basic building blocks (each provided with operator $L(\cdot)$), all the links between analogue units take the form of PWL signals obtained, in this case, by approximation. The PWL signals propagate from the block outputs to the inputs of their fanout blocks. Each of them is processed separately following the order of signal flow (*selective trace*). Clearly, the resulting PWL output $L(x)$ becomes an input x_{inp} of the relevant fanout blocks.

During simulation it is important to calculate the subsequent points $[t_{k+1}, x(t_{k+1})]$ in an efficient way and possibly avoid iterations that all typical optimisation procedures are based on. Therefore an explicit algorithm to solve this problem could be proposed.

The preliminary approach to resolve (2.8) avoiding transcendental equations was based on quadratic approximation for x [DAB95]. By means of the Taylor series expansion, and using the normalised time $\tau = t/T$ for $\tau_i = t_i/T = 0$ one obtains

$$x_q(\tau) = \frac{x_0 - u_0 + rT}{2}\,\tau^2 + (u_0 - x_0)\,\tau + x_0 \qquad (2.9)$$

The truncation error introduced by omitting in (2.9) the third order Lagrange rest is

$$\Theta(\hat{\tau}^3) = \frac{u_0 - x_0 - rT}{6}\,\hat{\tau}^3, \qquad \hat{\tau} \in (0, \tau) \qquad (2.10a)$$

To control a range of the quadratic approximation (2.9), eqn.(2.10*a*) can be reformulated, like in case of (2.6). The maximum allowed time τ_a for x_q to hold can be found from

$$\tau_a = \sqrt[3]{\frac{6\varepsilon_a}{|x_0 - u_0 + rT|}} \qquad (2.10b)$$

where ε_a is an estimated value of the maximum allowed truncation error. In this approach the constraint condition given by (2.8*a*) has to be modified by putting $t_{i+1} \in (t_i, t_i + t_a]$.

The relevant approximator works in two steps. First, the time τ_a is computed from (2.10*b*) to find the range of an acceptable quadratic approximation for the original response. In the second step an attempt to set the PWL segment is made, so that it possibly covers the full range of τ_a (for the end point of the segment we would have $\tau_1 = \tau_a$). If the Chebyshev distance $p(0, \tau_a)$ between x_{lin} and x exceeds the prescribed approximation accuracy p_{mx}, then the boundary point τ_1 has to be re-evaluated from the simple proportion

$$\frac{\tau_1}{\tau_a} = \sqrt{\frac{p_{mx}}{p(0, \tau_a)}} \qquad (2.11)$$

because x is close to the quadratic waveform x_q. Otherwise the primary evaluation for τ_1 holds. The next PWL segment can be found in the same way after the time shift $\tau \leftarrow (\tau - \tau_1)$ is performed. Since the original smooth waveform is almost quadratic for $\tau \in [0, \tau_a]$, no iterations are necessary to find the distance $p(0, \tau_a)$ and the segment length τ_1 (the other boundary point for the actual segment is defined for $\tau_0 = 0$) [DAB95].

Unfortunately, the PWL waveforms achieved by this approach tend to be sub-optimal according to criterion (2.8). In practice, the lengths of resulting PWL segments are limited by the quadratic approximation range τ_a, rather than by τ_{mx}, as happens mainly in case of flat parts (slowly changing fragments) of the exponential waveforms. As a consequence, those segments are shorter than they could be and fit much better to x than required and could be expected from p_{mx}.

The mentioned above drawback gives rise to search for a more efficient approximation technique adequate to replace the preliminary algorithm in the remaining of this work. One way would be using a higher order polynomial rather than the quadratic approximating function (2.9). In this case, however, the major advantage that comes out from employing the explicit formulas to calculate τ_a and τ_1 would be lost. Instead, CPU intensive algorithms (such as Newton-Raphson) to solve for the arising nonlinear equations would be required.

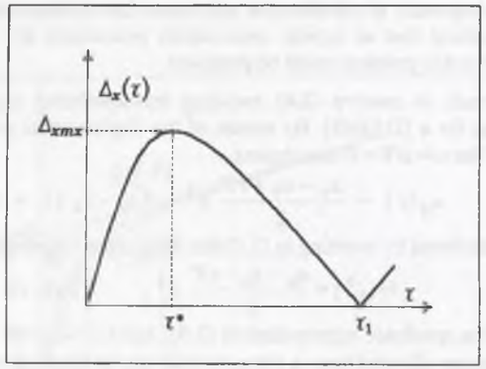

Fig. 2.4. Distance function $\Delta_x = | x - x_{lin} |$ against normalised time τ

An actual approach to solve for the formulated optimisation task (2.8) will be presented below [DAB99F]. For this purpose define the relevant distance function $\Delta_x(\cdot)$ to make use of the pure maximum condition for it

$$\Delta_x(\tau) = | x(\tau) - x_{lin}(\tau) | \tag{2.12}$$

Since x_{lin} crosses through x_0 and $x(t_1)$, i.e. $t_i = 0$, $t_{i+1} = t_1$ (see Fig.2.2), then based on (2.7)

$$\Delta_x(\tau) = \left| (x_0 - u_0 + rT) \left[(e^{-\tau} - 1) - (e^{-\tau_1} - 1)\frac{\tau}{\tau_1} \right] \right| \tag{2.13}$$

Because it is a differentiable function for $\tau \in (0, \tau_1)$, letting $d\Delta_x/d\tau = 0$ obtains

$$\tau^* = \ln \frac{\tau_1}{1 - e^{-\tau_1}} \tag{2.14}$$

for which Δ_x reaches its maximum value (see Fig.2.4), $\Delta_{xmx} = \Delta_x(\tau^*)$

$$\Delta_x(\tau^*) = | x_0 - u_0 + rT | \times \left| \frac{1 - e^{-\tau_1}}{\tau_1} \left(1 - \ln \frac{1 - e^{-\tau_1}}{\tau_1} \right) - 1 \right| \tag{2.15}$$

Note that $\Delta_x(\tau^*) = p(0, \tau_1)$ (see eqn. (2.8b)). Hence, letting $\Delta_x(\tau^*) = p_{mx}$ gives the required relation between the approximation accuracy p_{mx} and the maximum allowed time τ_1 (normalised segment length) with respect to the criteria of (2.8). Now, eqn.(2.15) can be rewritten in the form

$$\frac{p_{mx}}{| x_0 - u_0 + rT |} = 1 - \frac{1 - e^{-\tau_1}}{\tau_1} \left(1 - \ln \frac{1 - e^{-\tau_1}}{\tau_1} \right) \tag{2.16}$$

The left-hand side of (2.16) may be referred to as a relative approximation accuracy p_{mxr}, whereas the right-hand side is a monotonic function defined as $\varphi(\tau_1)$ that asymptotically approaches 1

(Fig.2.5). Using the reciprocal of this function $\Phi = \varphi^{-1}$, the optimisation task (2.8) can be solved directly

$$\tau_1 = \Phi(p_{mxr}), \quad p_{mxr} = \frac{p_{mx}}{| x_0 - u_0 + rT |} \tag{2.17a}$$

and an alternative notation, according to the step count, would be

$$\sigma_i = \Phi(p_{mxr}), \quad p_{mxr} = \frac{p_{mx}}{| x_i - u_i + r_i T |} \tag{2.17b}$$

where $\sigma_i = \tau_{i+1} - \tau_i$ is a length of i-th normalised segment along the time axis. Based on (2.17a) the subsequent approximation breakpoints may be achieved assuming that each segment begins at $t_i = 0$ and ends at $t_{i+1} = \tau_1 T$. In other words, the time shift $\tau \leftarrow (\tau - \tau_1)$ is performed before the next PWL segment is computed. It follows that x_0 and u_0 have to be updated for each PWL segment, i.e. the next x_0 can be found from (2.7) by means of the substitution: $x_0 \leftarrow x(\tau_1 T)$, and the next u_0 in the same way: $u_0 \leftarrow (u_0 + r\tau_1 T)$.

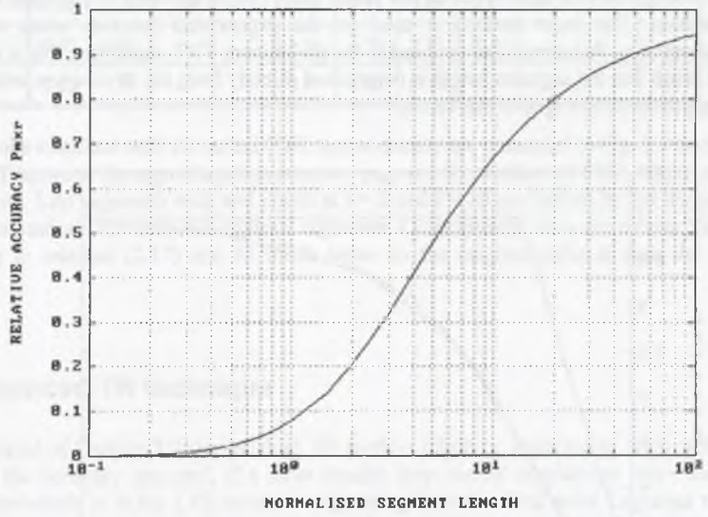

Fig. 2.5. Relative accuracy p_{mxr} against normalised segment length

However, some exceptions exist. From (2.16) and Fig.2.7 one can see that for $p_{mxr} \geq 1$ the Φ function is not defined. In practice it is the case that $\Delta_x < p_{mx}$ for any $\tau > 0$, this usually happens for very slowly changing exponential waveforms. Then a reasonable action in the PWL algorithm is the substitution: $\tau_1 \leftarrow \tau_{mx}$. The other case is for $x_0 - u_0 + rT = 0$, when x changes linearly in time with no transient component (see eqn.(2.7)). Here, $\tau_1 \leftarrow \tau_{mx}$ also. Clearly the same substitution holds for the τ_1 computed from (2.17) if its value exceeds the maximum allowed time τ_{mx}. Remember that τ_{mx} is either a time instant, for which a next PWL u segment is applied to the block input, or it is the end of simulation. This discussion may be summarised by a PWL algorithm depicted in Fig. 2.6.

```
repeat
  if x0 - u0 + rT <> 0 and | x0 - u0 + rT | > pmx
     then  {τ1 ← Φ(pmxr);
              if τ1 > τmx then τ1 ← τmx}
     else τ1 ← τmx;
  xk ← x(τ1T);  x0 ← xk;
  u0 ← ( u0 + rτ1T );
  tk ← ( tk-1 + τ1T );
  k ← ( k + 1 );
  τmx ← ( τmx - τ1 );
until τmx = 0
```

Fig. 2.6. Algorithm for PWL approximator

The result of this algorithm is two sequences $\{t_k\}$ and $\{x_k\}$ that define the PWL output waveform of any inertial building block. A look-up table and linear interpolation are used to calculate the values of the Φ function. The same method is used for the exponential function when computing subsequent values of x_k. Apparently, as opposed to the preliminary PWL algorithm, this is a one-step approach in a sense that the segment length is determined directly from the Φ function with no need of the initial approximation required previously.

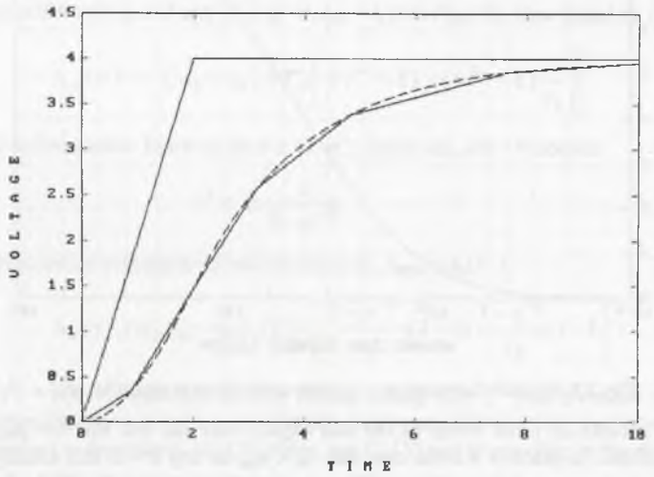

Fig. 2.7. Time response x (dashed line) and its PWL approximation x_{lin} (T=2 and p_{mx}=0.08) against excitation u consisting of two segments for $t\in[0, 2)$ and $t\in[2, 10]$

Now, observe how the length of subsequent PWL segments change. For brevity we define the actual amplitude of the transient component as $s_k = x_k - u_k + r_kT$. Then for the k-th segment the relation (2.17) can be rewritten as follows

$$\sigma_k = \Phi\left[p_{mxr}(k)\right], \quad p_{mxr}(k) = \frac{p_{mx}}{|s_k|} \tag{2.18}$$

Moreover, from (2.7) we have:

$$s_{k+1} = s_k e^{-\sigma_k} \tag{2.19}$$

It is easy to see that the sequence $\{|s_k|\}$ is decreasing, whereas the $\{\sigma_k\}$ is increasing, since $\sigma_k = \Phi(p_{mx}/|s_k|)$ and Φ is a monotonically ascending function. For sufficiently large values of index k, the sequence $\{s_k\}$ approaches zero, so that $\{x_k\}$ approaches $u_k - r_kT$. The latter effect agrees perfectly with the case of the smooth waveform x defined by (2.7) that approaches $[u_0+r(t-T)]$ when t approaches infinity.

Table 2.3. Time points of PWL segments for different approximation accuracy

p_{mx}	$t\in(0,2]$ first segment of u				$t\in(2,10]$ second segment of u							No of segments
0.02	0.42	0.88	1.41	2.00	2.53	3.15	3.88	4.77	5.93	7.56	10.00	4+7
0.04	0.61	1.32	2.00	—	2.78	3.75	5.04	6.95	10.00	—	—	3+5
0.08	0.89	2.00	—	—	3.16	4.79	7.57	10.00	—	—	—	2+4
0.16	1.33	2.00	—	—	3.75	6.92	10.00	—	—	—	—	2+3

Some results obtained with the actual PWL approximator are presented in Fig. 2.7 and in Table 2.3. A tradeoff between the approximation accuracy p_{mx} and the number of PWL output segments may be observed. Last segments with end points at $t = 2$ and $t = 10$ are limited by the length of the input segment u, and are the only non-optimal segments. Consequently, they are shorter than they could be owing to relation (2.17) and fit much better to the original curve x than the former PWL segments.

2.4 Enhanced TR technique

As mentioned in Section 2.2 the standard TR method might be expected to work with larger steps retaining the accuracy imposed, if a more precise step control mechanism were used for it. The existing drawback is in the LTE estimate originating from the third order Lagrange rest (2.6*b*). To cope with this problem, we invoke the definition for the LTE and expand its both components x_k and $x(t_k)$ into the Taylor series around $h_k = 0$. For (2.5*b*) obtains

$$x_k = x_{k-1} + (u_{k-1} - x_{k-1})\frac{h_k}{T} + (r_{k-1}T - u_{k-1} + x_{k-1})\cdot\left(\frac{h_k}{T}\right)^2 + \\ + (r_{k-1}T - u_{k-1} + x_{k-1})\cdot\left[\frac{-1.5}{3!}\left(\frac{h_k}{T}\right)^3 + \frac{3}{4!}\left(\frac{h_k}{T}\right)^4 + \frac{-7.5}{5!}\left(\frac{h_k}{T}\right)^5 + \frac{22.5}{6!}\left(\frac{h_k}{T}\right)^6 + \ldots.\right] \tag{2.20}$$

Similarly, based on (2.7) and the notation of Section 2.3 for $\sigma_k = h_k/T$

$$x(t_k) = (x_{k-1} - u_{k-1} + r_{k-1}T)\,e^{-\frac{h_k}{T}} + r_{k-1}(h_k - T) + u_{k-1} \tag{2.21}$$

so in terms of the Taylor expansion for (2.21) holds

$$x(t_k) = x_{k-1} + (u_{k-1} - x_{k-1})\frac{h_k}{T} + (r_{k-1}T - u_{k-1} + x_{k-1})\cdot\left(\frac{h_k}{T}\right)^2 + \\ + (r_{k-1}T - u_{k-1} + x_{k-1})\cdot\left[\frac{-1}{3!}\left(\frac{h_k}{T}\right)^3 + \frac{1}{4!}\left(\frac{h_k}{T}\right)^4 + \frac{-1}{5!}\left(\frac{h_k}{T}\right)^5 + \frac{1}{6!}\left(\frac{h_k}{T}\right)^6 + \ldots.\right] \tag{2.22}$$

Consequently, from (2.20) and (2.22) the LTE appears to be

$$\varepsilon_k = x_k - x(t_k) = \\ (x_{k-1} - u_{k-1} + r_{k-1}T)\cdot\left[\frac{-0.5}{3!}\left(\frac{h_k}{T}\right)^3 + \frac{2}{4!}\left(\frac{h_k}{T}\right)^4 + \frac{-6.5}{5!}\left(\frac{h_k}{T}\right)^5 + \frac{21.5}{6!}\left(\frac{h_k}{T}\right)^6 + \ldots.\right] \tag{2.23}$$

When dividing both sides of (2.23) by $(x_{k-1} - u_{k-1} + r_{k-1}T)$ the resulting formula resembles perfectly (2.15) in a sense that the normalised step h_k/T is mapped into the relative accuracy, as well. Hence, by putting $\varepsilon_k = \varepsilon_0$, in (2.23) a function like (2.17) might be established

$$\frac{h_k}{T} = \Phi_{TR}(\varepsilon_{0r}), \qquad \varepsilon_{0r} = \frac{\varepsilon_0}{|x_{k-1} - u_{k-1} + r_{k-1}T|} \tag{2.24}$$

Since the right-hand side of (2.23) is an infinite sum, the relation (2.24) can be picked-up based directly on the difference $x_k - x(t_k)$ for any set of parameters: x_{k-1}, u_{k-1}, r_{k-1}, T provided $x_{k-1} = x(t_{k-1})$. Hence, it follows

$$\left(\frac{h_k}{T}\right) \rightarrow \left|\frac{x_k - x(t_k)}{x_{k-1} - u_{k-1} + r_{k-1}T}\right| \tag{2.25}$$

The result of it is shown in Fig. 2.8. Note that the Φ_{TR} resembles the Φ function of Fig. 2.5, but it rises slightly faster than Φ for the relative accuracy below some 0.7 and slower otherwise. Moreover, the PWL algorithm presented in Fig.2.6 can be adopted for the enhanced TR method, provided the Φ function is replaced by Φ_{TR}, and τ_1 represents h_k/T. Consequently, as opposed to standard approaches, now the enhanced TR algorithm is able to also perform very large steps, when flat fragments of $x(t)$ are faced. Like in case of the PWL approximation algorithm, the step h_k/T is limited by the end of the actual input segment $u_i(t)$. In particular, when the transient component amplitude $|x_{k-1} - u_{k-1} + r_{k-1}T|$ becomes smaller than ε_0, i.e. $\varepsilon_{0r} > 1$, the algorithm yields a step corresponding to the end-point of $u_i(t)$. To prove correctness of this (in particular for very large steps), observe that the resulting error ε_k does not exceed the distance δ_{k-1} between x_{k-1} and the asymptotic line that is approached by $x(t)$. Indeed, based on (2.7) the asymptotic line is

$$\lim_{t\to\infty} x(t) = u_0 + r(t - T)$$

Hence, the distance to this asymptote at t_{k-1} is

$$\delta_{k-1} = x_{k-1} - [u_0 + r_{k-1}(t_{k-1} - T)] = x_{k-1} - u_{k-1} + r_{k-1}T$$

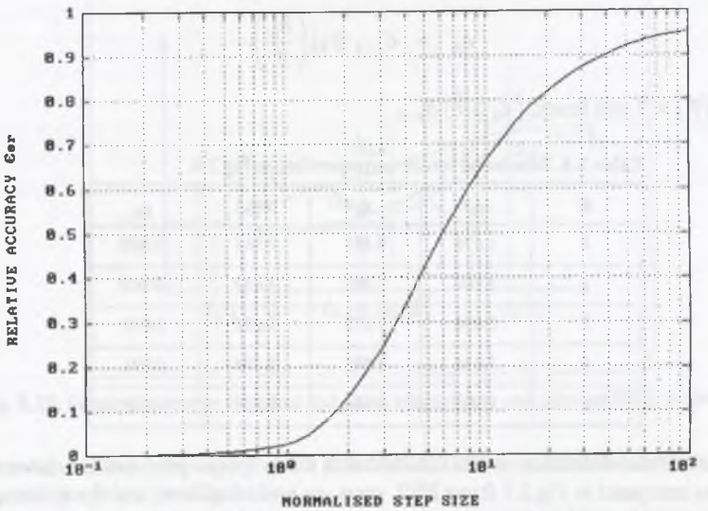

Fig. 2.8. Relative accuracy ε_{0r} against normalised step size

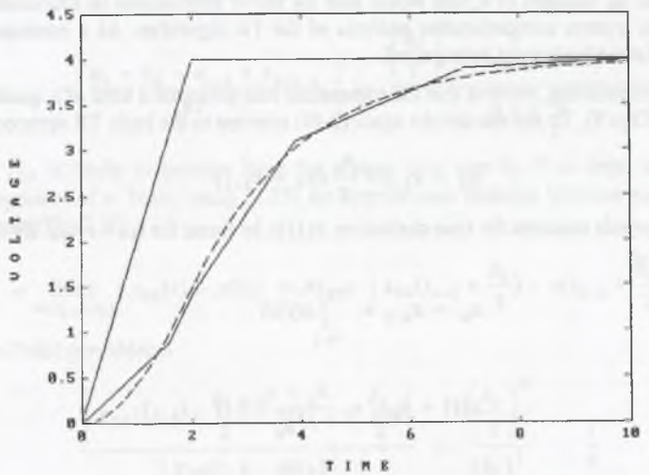

Fig. 2.9. Time response x (dashed line) and PWL response x_{lin} obtained with actual TR approach (T=2 and ε_0=0.08) against excitation u consisting of two segments for $t\in[0, 2)$ and $t\in[2, 10]$

On the other hand ε_k may be evaluated from (2.23), and now it can be rewritten as

$$\varepsilon_k = \delta_{k-1}\,\Phi_{TR}^{-1}\left(\frac{h_k}{T}\right)$$

but $\Phi_{TR}^{-1}(h_k/T) < 1$ and hence, $|\varepsilon_k| < |\delta_{k-1}|$.

Table 2.4. Simulation results corresponding to Fig.2.9.

k	t_k	x_k	$x(t_k)$	ε_k
1	1.576	0.881	0.961	-0.080
2	2.000	1.407	1.472	-0.065
3	3.896	3.075	3.020	0.055
4	7.014	3.885	3.794	0.091
5	10.00	3.983	3.954	0.029

In Fig.2.9 the relevant simulation results obtained with the technique proposed are shown as a PWL waveform. As compared to Fig.2.7 fewer PWL steps are performed here, and the resulting errors $\varepsilon_k = x_k - x(t_k)$, depicted in Table 2.4, are shown to accumulate slightly only in some cases. For k=1 the resulting error $\varepsilon_1 = \varepsilon_0$, since $x_0 = x(t_0)$, so it is a perfect LTE. On the other hand, for k=2 and k=5 the errors ε_k are less than ε_0, since the related steps h_k/T are limited by the time instants of 2 and 10. However, for some in-between points $x_{lin}(t)$, such that $t_k < t < t_{k+1}$, the distance $|x_{lin}(t) - x(t_k)|$ appears much larger than ε_0. Because of it, one would find the above approach to be inconsistent giving, in this way, rise to a more comprehensive analysis of the TR algorithm. As a consequence, further refinement of that method might be expected.

To cope with this problem, observe that the trapezoidal rule performs a kind of a quadratic estimate for x [OGR94, Chpt.8]. To see this invoke again (2.4*b*) relevant to the basic TR approach

$$x_k = x_{k-1} + \frac{h_k}{2}(\dot{x}_k + \dot{x}_{k-1})$$

Note that this formula assumes the time derivative $\dot{x}(t)$ to be linear for $t_{k-1} < t < t_k$. By comparison to the general rule of

$$x_k = x_{k-1} + \int_{t_{k-1}}^{t_{k-1}+h_k} \dot{x}(t)dt \qquad (2.26)$$

one obtains

$$\dot{x}(t) = \dot{x}_{k-1} + \frac{\dot{x}_k - \dot{x}_{k-1}}{h_k}(t - t_{k-1}) \qquad (2.27)$$

and hence

$$x_{TR}(t) = x_{k-1} + \dot{x}_{k-1}(t - t_{k-1}) + \frac{\dot{x}_k - \dot{x}_{k-1}}{2h_k}(t - t_{k-1})^2 \qquad (2.28)$$

where the index TR is used to distinguish from the original waveform x. Apparently, (2.28) is a quadratic estimate with the error rising monotonically from 0 for t_{k-1} up to ε_0 for t_k due to (2.24).

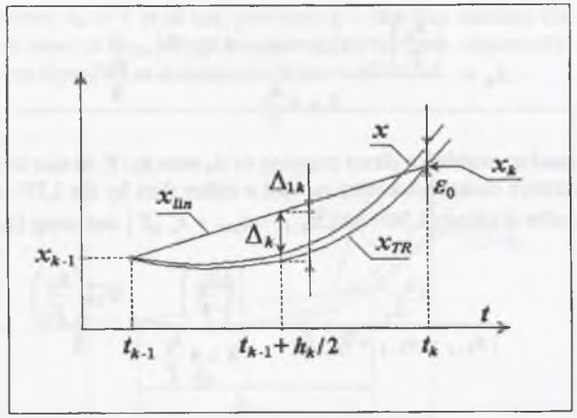

Fig. 2.10. Original response x against quadratic estimate x_{TR} and relevant PWL segment x_{lin}

To evaluate the distance between x_{lin} and x in the Chebyshev sense, first the distance between x_{TR} and the x_{lin} will be found (see Fig.2.10)

$$\Delta_{1k} = \underset{t\in(t_{k-1},\,t_k)}{Max}\left|x_{lin}(t) - x_{TR}(t)\right| = \left|\frac{x_{k-1}+x_k}{2} - x_{TR}\left(t_{k-1}+\frac{h_k}{2}\right)\right|$$

$$= \left|\frac{x_k - x_{k-1}}{2} - \frac{\dot{x}_k + 3\dot{x}_{k-1}}{8}h_k\right| = \left|\frac{\dot{x}_k - \dot{x}_{k-1}}{8}h_k\right| \qquad (2.29)$$

$$= \left|\frac{u_k - x_k - u_{k-1} + x_{k-1}}{8T}h_k\right| = \frac{\left(\frac{h_k}{T}\right)^2 \cdot \left|x_{k-1} - u_{k-1} + r_{k-1}T\right|}{8 + 4\frac{h_k}{T}}$$

Observe that Δ_{1k} is likely to become large for a large time step h_k/T or large amplitude of the transient component of x. Now, using (2.29) the approximate distance between x_{lin} and x may be found as follows (Fig.2.10)

$$\Delta_k = \underset{t\in(t_{k-1},\,t_k)}{Max}\left|x_{lin}(t) - x(t)\right| \approx \Delta_{1k} - \left|x_{TR}\left(t_{k-1}+\frac{h_k}{2}\right) - x\left(t_{k-1}+\frac{h_k}{2}\right)\right| \qquad (2.30a)$$

and based on (2.6*b*) one obtains

$$\frac{\left|x_{TR}\left(t_{k-1}+\frac{h_k}{2}\right) - x\left(t_{k-1}+\frac{h_k}{2}\right)\right|}{\left|x_{TR}(t_k) - x(t_k)\right|} \approx \frac{\left(\frac{h_k}{2}\right)^3}{(h_k)^3} = \frac{1}{8}$$

where $|x_{TR}(t_k) - x(t_k)|$ is the LTE equal ε_0, and hence the relation between Δ_k and ε_0 is

$$\Delta_k \approx \frac{\left(\frac{h_k}{T}\right)^2 \cdot \left| x_{k-1} - u_{k-1} + r_{k-1}T \right|}{8 + 4\frac{h_k}{T}} - \frac{\varepsilon_0}{8} \qquad (2.30b)$$

This result can be used to establish a direct mapping of Δ_k onto h_k/T, so that the time step could be controlled by the relative distance between x_{lin} and x rather than by the LTE (as used in standard approaches). Thus, after dividing (2.30*b*) by $\left| x_{k-1} - u_{k-1} + r_{k-1}T \right|$ and using (2.24) one obtains

$$\frac{\Delta_k}{\left| x_{k-1} - u_{k-1} + r_{k-1}T \right|} \approx \frac{\left(\frac{h_k}{T}\right)^2}{8 + 4\frac{h_k}{T}} - \frac{\Phi_{TR}^{-1}\left(\frac{h_k}{T}\right)}{8} \qquad (2.31)$$

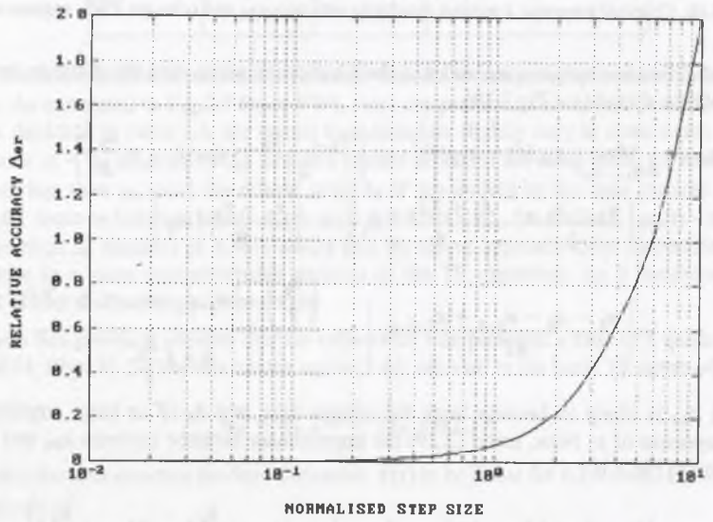

Fig. 2.11. Relative accuracy Δ_{0r} against normalised step size

It is easy to see that (2.31) defines a function of h_k/T, which is depicted in Fig.2.11. The left-hand side of (2.31) (i.e. function values) may be referred to as the relative distance between x_{lin} and x for $t \in (t_{k-1}, t_k)$. In fact, the reciprocal of (2.31) is of interest and it can be denoted in brief as

$$\frac{h_k}{T} = \Phi_{TRe}(\Delta_{0r}), \qquad \Delta_{0r} = \frac{\Delta_0}{\left| x_{k-1} - u_{k-1} + r_{k-1}T \right|} \qquad (2.32)$$

where Δ_0 is the absolute accuracy imposed, and Δ_{0r} is the corresponding relative accuracy. Unlike the functions Φ and Φ_{TR}, here, the relative accuracy exceeds 1 for large time steps. Although the

fragment of Φ_{TRe} where $\Delta_{0r} > 1$ is of use, performing a step that matches the end-point of $u_i(t)$ rather than the actual value of Φ_{TRe} would be preferred in this case. Apparently, with this approach we follow the previous algorithm as it appears to be more effective.

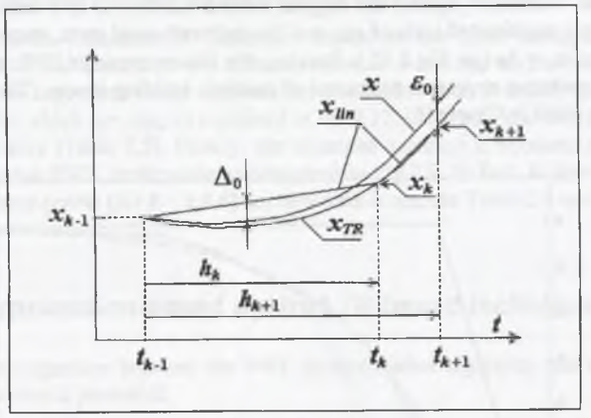

Fig. 2.12. Steps performed with Φ_{TRe} and Φ_{TR} against PWL response for enhanced TR algorithm

```
repeat
  if x0 - u0 + rT <> 0 and | x0 - u0 + rT | > ε0
     then  {σk ← ΦTRe(ε0r);
              if σk > τmx then σk ← τmx}
     else σk ← τmx;
  xk ← x(σkT);  tk ← ( t0 + σkT );                (* first step *)
  if σk < τmx
     then { k ← ( k + 1 ); σk ← ΦTR(ε0r);
            if σk > τmx then σk ← τmx
            xk ← x(σkT); tk ← ( t0 + σkT ) }      (* second step *)
  x0 ← xk;
  u0 ← ( u0 + rσkT );
  t0 ← tk
  k ← ( k + 1 );
  τmx ← ( τmx - σk );
until τmx = 0
```

Fig. 2.13. Enhanced TR algorithm

By using the function Φ_{TRe} we tend to keep the relevant PWL solution closer to x than possible with the Φ_{TR}. In fact, the additional accuracy constraints (2.32) are imposed on the Chebyshev distance Δ_k. The idea standing behind it is that for a given point x_{k-1}, the enhanced TR algorithm would

perform two steps, the first one (shorter) based on the Φ_{TRe}, and the other one based on Φ_{TR}. The breakpoints obtained, i.e. x_k and x_{k+1}, lie on the same curve x_{TR}.

In fact, by combining those two mappings, the resulting PWL waveform (with extra breakpoints) appears to be well "balanced" against the original smooth solution x, in a sense that the distance between x and the overestimated parts of x_{lin}, and the underestimated parts, respectively, are almost the same, provided $\varepsilon_0 = \Delta_0$ (see Fig.2.12). Besides, this feature prevents PWL error accumulation when modelling analogue structures composed of multiple building blocks. This problem will be discussed in more detail in Chapter 6.

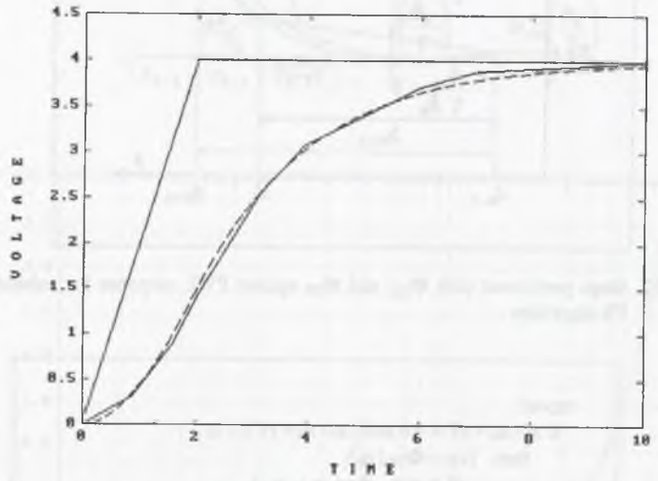

Fig. 2.14. Time response x (dashed line) and PWL response x_{lin} obtained with enhanced TR approach (T=2 and ε_0=0.08) against excitation u consisting of two segments for $t\in[0, 2)$ and $t\in[2, 10]$

Table 2.5. Simulation results corresponding to Fig.2.14.

k	t_k	x_k	$x(t_k)$	ε_k
1	0.896	0.328	0.348	-0.020
2	1.576	0.881	0.961	-0.080
3	2.000	1.407	1.472	-0.065
4	3.146	2.562	2.575	-0.013
5	3.896	3.075	3.020	0.055
6	6.001	3.713	3.658	0.055
7	7.014	3.885	3.794	0.091
8	10.00	3.983	3.954	0.029

In this context the enhanced TR algorithm could be formulated as shown in Fig.2.13. The symbol σ_k stands for the normalised step, and the accuracy equal ε_0 is imposed both on the LTE = ε_k and the distance Δ_k. Observe also that the second step is conditional here, and it is only performed provided the first step does not reach the end-point due to the input segment.

The application of the enhanced TR technique related to the latter example (of Fig.2.9) is presented in Fig.2.14 and Table 2.5. The distance between x and x_{lin} for the breakpoints and the in-between points is kept close to the accuracy assumed. However, it is only guaranteed for the initial segments, for which $x_0 = x(t_0)$ as explained in Fig.2.12. Otherwise it is likely to decrease or to accumulate moderately (Table 2.5). Clearly, the enhanced accuracy is obtained at the expense of increasing the number PWL breakpoints as compared to Fig.2.9. In fact, following the algorithm of Fig.2.13 only three points (for k = 1,4,6) are new here (compare Table 2.4 and 2.5).

2.5 PWL approximation-based against TR-based techniques

In this section a comparison between the PWL approximation algorithm, the standard- and the enhanced TR algorithm is presented.

First, observe that the PWL approximation-based algorithm delivers a waveform that fits perfectly to the original response x at boundary points of the PWL segments, i.e. $x(0) = x_{lin}(0)$ and $x(\tau_1) = x_{lin}(\tau_1)$ (see Fig.2.2 or Fig.2.7). Those segments correspond to the steps performed by the TR techniques. However, the way to control the step/ segment size is completely different for them. In the PWL approximation approach the segment length τ_1 is calculated from the Φ function, and the approximation error $p(0,\tau_1)$ (maximum distance between x and x_{lin} for $\tau = \tau^* < \tau_1$) never exceeds the assumed accuracy p_{mx}. In contrary to this, the both TR techniques use the local truncation error (LTE) step control that plays a role of some approximation accuracy, and is an estimate rather than a real distance between $x(t_k)$ and the computed value x_k. Moreover, this distance is estimated assuming correct value of the starting point. As a consequence, the starting point x_{k-1} (except of the first point) introduces an error that tends to vary from step to step, and in some cases to accumulate resulting in a global error, unlike in the PWL approximation-based algorithm. Besides, an extra step is produced using the Φ_{TRe} function to keep the accuracy for the in-between points. The relevant PWL errors corresponding to the waveforms given in Figs.2.7 and 2.14 are depicted in Fig.2.15. Apparently, in case of the enhanced TR, the errors are balanced better for the convex and concave part of the waveform, although the error amplitudes tend to accumulate temporarily and are larger than those of the PWL approximation technique.

Next, our attention will be focused on step/ segment lengths produced by the relevant algorithms assuming the same values of ε_0 and p_{mx}. Moreover, to assure a kind of compatibility between PWL approximation-based and the TR algorithms, for a given segment we denote $\tau_1 T$ as h_k. In fact, using the notation for the standard integration techniques introduced in Section 2.2, the formula (2.21) could be used directly

$$x_k = (x_{k-1} - u_{k-1} + r_{k-1}T)e^{-\frac{h_k}{T}} + r_{k-1}(h_k - T) + u_{k-1} \qquad (2.33)$$

For completeness the formula (2.5*b*) defining the TR techniques can be invoked too, and rewritten in the form of

$$x_k = \frac{(1 - \frac{h_k}{2T})x_{k-1} + \frac{h_k}{2T}(2u_{k-1} + r_{k-1}h_k)}{1 + \frac{h_k}{2T}} \qquad (2.34)$$

The comparison between the PWL approximation-based and the TR-based techniques could be performed as shown in Tables 2.6*a,b* and 2.7*a,b*. The unit step input and linear input are applied to (2.1) with $T = 1$, like in Section 2.2.

Apparently, in case of slowly changing fragments of x much more steps are required for the standard TR technique than for the two others. When the transient component of x becomes smaller than p_{mx} or ε_0, the PWL algorithm and the enhanced TR can provide a segment (step) of any size. In practice, however, it means that the end-point of the actual input segment $u_k(t)$ is reached, denoted in Tables 2.6*a,b* and 2.7*a,b* as t_{mx} (shaded cells). The errors of the PWL approximation-based algorithm at t_k are not shown, since they are zero at the breakpoints. On the other hand, the effects of overshooting or ringing that feature the TR algorithms are emphasised as shaded cells as well.

Table 2.6*a*. Step responses of test block obtained with PWL approximation-based and TR-based algorithms ($u = 1(t)$ and $x(0) = 0$)

$p_{mx} = \varepsilon_0 = 0.05$											
	PWL			Standard TR				Enhanced TR			
k	t_k	h_k	x_k	t_k	h_k	x_k	ε_k	t_k	h_k	x_k	ε_k
1	0.758	0.758	0.531	0.843	0.843	0.593	0.023	0.759	0.759	0.550	0.018
2	1.984	1.226	0.862	1.981	1.138	0.888	0.026	1.167	1.167	0.737	0.049
3	5.474	3.490	0.996	3.731	1.750	0.993	0.017	2.905	1.738	0.982	0.037
4	t_{mx}	$t_{mx}-t_3$	1.000	8.140	4.409	1.003	0.003	3.645	2.478	1.028	0.054
5	-	-	-	13.99	5.848	0.999	-0.001	t_{mx}	$t_{mx}-t_4$	0.928*)	-0.022
6	-	-	-	22.42	8.434	1.001	0.001	-	-	-	-

*) obtained for $t_{mx} = 20$

Table 2.6*b*. Step responses of test block obtained with PWL approximation-based and TR-based algorithms ($u = 1(t)$ and $x(0) = 0$)

$p_{mx} = \varepsilon_0 = 0.1$											
	PWL			Standard TR				Enhanced TR			
k	t_k	h_k	x_k	t_k	h_k	x_k	ε_k	t_k	h_k	x_k	ε_k
1	1.174	1.174	0.691	1.063	1.063	0.694	0.039	1.147	1.147	0.729	0,047
2	4.442	3.268	0.985	2.640	1.577	0.964	0.035	1.682	1.682	0.914	0.100
3	t_{mx}	$t_{mx}-t_2$	1.000	5.858	3.218	1.008	0.011	t_{mx}	$t_{mx}-t_2$	1.069*)	0.069
4	-	-	-	11.17	5.313	0.996	-0.004	-	-	-	-
5	-	-	-	17.87	6.694	1.002	0.002	-	-	-	-
6	-	-	-	26.30	8.434	0.999	-0.001	-	-	-	-

*) obtained for $t_{mx} = 20$

Table 2.7*a*. Responses to linear input obtained with PWL and standard algorithms ($u = t \cdot 1(t)$ and $x(0) = 1$)

$p_{mx} = \varepsilon_0 = 0.05$											
	PWL			Standard TR				Enhanced TR			
k	t_k	h_k	x_k	t_k	h_k	x_k	ε_k	t_k	h_k	x_k	ε_k
1	0.505	0.505	0.712	0.669	0.669	0.666	-0.027	0.509	0.509	0.697	-0.014
2	1.182	0.677	0.795	1.513	0.844	0.918	-0.035	0.867	0.867	0.657	-0.050
3	2.210	1.028	1.430	2.654	1.140	1.764	-0.030	1.740	0.873	1.050	-0.041
4	4.410	2.200	3.435	4.408	1.755	3.415	-0.017	2.180	1.314	1.344	-0.056
5	t_{mx}	$t_{mx}-t_4$	$t_{mx}-t_4$	8.817	4.409	7.814	-0.003	4.569	2.389	3.555	-0.035
6	-	-	-	14.67	5.848	13.67	0.002	5.723	3.543	4.677	-0.052
7	-	-	-	21.36	6.694	20.36	-0.001	t_{mx}	$t_{mx}-t_6$	19.038*)	0.034

*) obtained for $t_{mx} = 20$

Table 2.7*b*. Responses to linear input obtained with PWL and standard algorithms ($u = t \cdot 1(t)$ and $x(0) = 1$)

$p_{mx} = \varepsilon_0 = 0.1$											
	PWL			Standard TR				Enhanced TR			
k	t_k	h_k	x_k	t_k	h_k	x_k	ε_k	t_k	h_k	x_k	ε_k
1	0.758	0.758	0.695	0.843	0.843	0.657	-0.047	0.759	0.759	0.659	-0.036
2	1.984	0.791	1.259	2.063	1.165	1.263	-0.054	1.167	1.167	0.690	0.099
3	5.474	0.884	4.482	3.880	1.817	2.890	-0.031	2.905	1.738	1.942	-0.073
4	t_{mx}	t_{mx}-t_3	x_{mx}-x_3	8.812	4.932	7.808	-0.015	3.645	2.478	2.589	-0.103
5	-	-	-	15.51	6.694	14.51	0.002	t_{mx}	t_{mx}-t_4	19.044*)	0.044
6	-	-	-	23.94	8.434	22.94	-0.001	-	-	-	-

*) obtained for $t_{mx} = 20$

The even steps performed by the enhanced TR algorithm are much larger than the subsequent steps of the standard TR because of using the Φ_{TR} mapping. The odd steps, produced by the Φ_{TRe}, are shorter but allow to keep the distance (approx. ε_0) between the resulting PWL segments and x. In contrast to this, the standard TR method appears to be inconsistent, in a sense that the in-between points of the PWL segments tend to be out of control as discussed in previous section (Fig.2.10). As the enhanced TR is only slightly inclined to accumulate the local errors, it could compete with the approximation-based approach.

The computational overhead of the methods can be estimated by making a comparison between (2.33) and (2.34), and taking into account the step/ segment length control mechanisms. As many as 9 and 11 simple operations (+,-,*) are required when using (2.33) and (2.34), respectively. Additionally, (2.33) requires one exponential function evaluation that can be performed effectively with table look-up technique. Similarly, to calculate the segment/ step, the PWL approximation-based and the enhanced TR-based technique use accordingly the functions Φ, Φ_{TR}, Φ_{TRe}, all treated with table look-up method. The standard TR exploits the formula (2.6*b*) including the root extraction, for which the table look-up might be used as well.

Apparently, the PWL approximation algorithm saves 2 simple operations, but it requires an extra exponential function evaluation as compared to the TR algorithms. For binary search, with n segments used to define the exponent, at most $\log_2 n$ comparisons are required [RAL71]. Using $16 < n \leq 32$ gives $\log_2 n \leq 5$ (with an average less than 4). As a result, the CPU time that the PWL approximation algorithm needs to calculate a point, takes some 10% longer. In practice, however, as it performs fewer points than the TR algorithms (as shown above), the effective speed of the PWL approximation-based and the enhanced TR-based algorithm is usually the same.

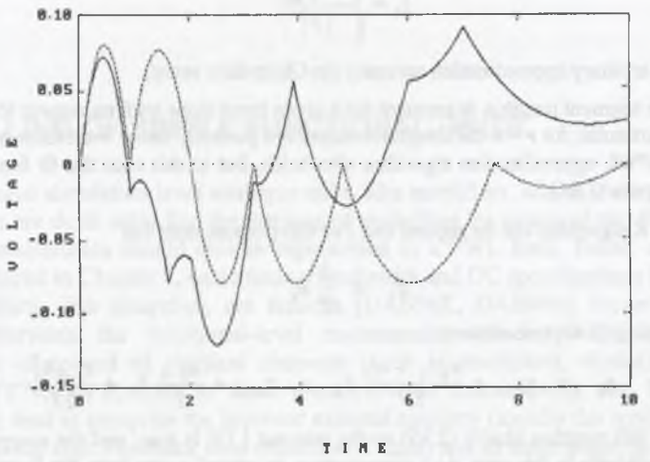

Fig.2.15. PWL errors relevant to waveforms of Fig.2.7 (dashed line) and Fig.2.14 (solid line) obtained with PWL approximation based technique and enhanced TR technique, respectively (p_{mx}=ε_0=0.08)

On the other hand, the advantage of the PWL approximation-based approach is that it follows perfectly the original smooth solution, as opposed to the TR-based technique with unrealistic overshooting or error accumulation appearing in some cases (solid line in Fig.2.15). Ultimately, only the PWL approximation-based and the enhanced TR-based algorithm will be used in the remaining of this work.

2.6 Other macrosimulation issues

Because of their inertial nature most analogue units may be modelled with basic inertial blocks. Besides, the amplifier macromodel presented in the next chapter will be shown to require an extra dynamic element, i.e. the slope limiting module. Also some other effects encountered in analogue circuits need special treatment. In this section, attention is focused on signal multiplication and integration. From the point of view of the PWL approach, both effects may be regarded as special cases.

Assume an effective input signal in form: $u = u_0 + rt$. For the output signal x of an ideal integrator we have the formula

$$x(t) = \frac{1}{T}\left(u_0 t + \frac{r t^2}{2}\right) + x_0 \qquad (2.35)$$

where T is the integrator time constant and x_0 is the initial value of x. If the linear segment x_{lin} crosses x at $(0, x_0)$ and $(t_1, x(t_1))$, then $\Delta_x = |x_{lin} - x|$ as a strictly quadratic function reaches its maximum value $\Delta_{xmx} = |rt_1^2|/(8T)$ at $t^* = t_1/2$. Thus by letting $p_{mx} = \Delta_{xmx}$ the maximum PWL segment length may be obtained

$$t_1 = \sqrt{\frac{8T\,p_{mx}}{|r|}} \tag{2.36}$$

where p_{mx} is an arbitrary approximation accuracy (in Chebyshev sense).

Observe that the segment length t_1 is constant for a given input slope with no respect to actual values of x and u. In particular, for $r = 0$ the integrator output is a perfectly linear waveform, so that $t_1 = t_{mx}$. Thus the basic PWL approximation algorithm also holds, but in this case the Φ function must be replaced by formula (2.36).

The enhanced TR algorithm can be applied too. For this purpose note that

$$\dot{x}_k = \frac{1}{T} u_k \tag{2.37}$$

Hence, by invoking (2.4*b*) one obtains

$$x_k = x_{k-1} + \frac{u_{k-1} + u_k}{2T} h_k = x_{k-1} + \frac{u_{k-1}}{T} h_k + \frac{r_{k-1} h_k^2}{2T} \tag{2.38}$$

Observe that (2.38) matches ideally (2.35) so the relevant LTE is zero, and the mappings Φ_{TR} and Φ_{TRe} are not applicable in this case. As a consequence, to control a step here, the formula (2.36) can be used as well. Because of it when addressing an integrator, no difference between the PWL approximation-based and the enhanced TR-based algorithm can be found.

A similar result may be obtained for an ideal multiplier. When its output satisfies a formula $x = a u_1 u_2$ and the input signals are respectively $u_1 = u_{01} + r_1 t$, $u_2 = u_{02} + r_2 t$, we have

$$x(t) = a\,u_{01}u_{02} + a\,(u_{01}r_2 + u_{02}r_1)\,t + a\,r_1 r_2\,t^2 \tag{2.39}$$

Since the maximum value of the distance function is $\Delta_{xmx} = a\,|r_1 r_2|\,t_1^2/4$, the formula for the PWL segment length takes the form of

$$t_1 = 2\sqrt{\frac{p_{mx}}{a\,|r_1 r_2|}} \tag{2.40}$$

Additionally, a realistic multiplier macromodel should be provided with an inertial block as its output stage.

In the context of this section, it is to be pointed out that another PWL approximation-based algorithm can be used too [KRU96, CHE66]. In that case the obtained points $x_{lin}(t_i)$ are no longer assumed to lie on the original curve x; so x is in some sense interlaced by x_{lin} waveform. However, the corresponding approximation algorithm is efficient only for quadratic curves, and it requires a CPU intensive optimisation procedure otherwise. Moreover, no unique solution is guaranteed [CHU86].

On the other hand, observe that the enhanced TR technique provides a kind of approximation crossing the original waveform x (for inertial blocks). This results in erroneous overshooting in some cases, but the method is less prone to error accumulation, when cascade structures are considered. We will address this problem in Chapter 6.

3. MODELLING OF SIMPLE FUNCTIONAL UNITS

At the functional simulation level analogue units, like amplifiers, adders, voltage comparators or multipliers are dealt with. For the purpose of modelling by means of the PWL technique, their DC characteristics should also be represented in a PWL form. Based on the building blocks introduced in Chapter 2, basic timing/bandwidth and DC specifications including some nonlinear effects, like saturation, are feasible [DAB96K, DAB99F]. However, there is a distinction between the functional-level macromodels (models) and the SPICE-like macromodels composed of physical elements (such as transistors, diodes) or controlled sources [BOY74]. In contrast to those electrical-level macromodels, the functional-level macromodels tend to comprise the involved external circuitry (usually the feedback elements) within a common unit. Feedback loop elements (if exist) are, in some sense, hidden inside the model. From electrical point of view it also means that the PWL signals represent voltages and the blocks used for synthesis are assumed to be unidirectional so that currents are usually not accounted for. Electrical effects that require bi-directional signal flow (such as for transmission gates or coupling capacitors) should be avoided by means of incorporating them into the building blocks, like in case of the tight feedback loops.

In principle, a functional-level analogue macromodel might be thought as a structure composed of basic building blocks such as the inertial block, integrator or the purely static block. Typically, two building blocks connected in cascade might constitute a simple model. Those blocks represent usually the front-end and the output stage of a real analogue unit. When possible, the relevant specifications are assigned to individual blocks separately, e.g. large signal- and small signal behaviour or delay- (involved with saturation) and rise/fall effect.

On the other hand, it has been shown [RUA91, KRU96] that basic logic functions, performed typically by logic gates, can also be implemented by simple analogue operations along with some delay mechanism when PWL signals are assumed to be their arguments. However, this approach cannot be easily used for behavioural models of complex digital units, which usually make use of the Boolean- or a multiple-value algebra (e.g. 1,0,X,Z). Hence, for complex mixed A/D networks the mixed signal simulation is preferred [SAL94]. Clearly, the PWL- and the logic models (used in the presented approach) require signal conversion, when signals have to propagate between them. As a consequence, the effective outputs of logic models driving analogue units are of the PWL type as well.

In this chapter the PWL macromodels of voltage comparator, amplifier and logic gate are addressed. Their detailed time/frequency specifications are taken into account and the introduced previously basic inertial building blocks are used for synthesis. Finally, a performance of the macromodels derived is shown by a comparison to the respective SPICE estimates in the time domain.

3.1 Voltage comparator macromodel

For a comparator that switches its output between two voltage levels the delay effects are essential. A few factors influence the comparator time delay: input initial polarisation, input overdrive and loading effects at the output. Also an intrinsic delay must be accounted for. To represent the delay effects caused by the input signals a cascade of two building blocks and a delay mechanism can be used

$$\begin{aligned} T_0\,\dot{x} \;+\; x \;&=\; f_0(u_{inp}) \\ T_1\,\dot{y} \;+\; y \;&=\; f_1(x) \end{aligned} \tag{3.1}$$

where u_{inp} denotes the differential input signal. The nonlinear function $f_0(\cdot)$ enables one to represent the initial polarisation and the overdrive effect, whereas $f_1(\cdot)$ is a PWL approximation of the comparator DC characteristics (Fig. 3.1). Within a short range $x \in [V_-, V_+]$ it rises with a constant gain of $(U_{OH} - U_{OL})/(V_+ - V_-)$. For $x > V_+$: $f_1(x) = U_{OH}$ and for $x < V_-$: $f_1(x) = U_{OL}$. The function $f_0(\cdot)$ does not influence the DC characteristics, since it rises with the gain k_0 equal unity for $u_{inp} \in [U_-, U_+]$, and $U_- < V_-$, $U_+ > V_+$. In fact, the bigger the initial polarisation and the less the input overdrive, the longer the time required to leave the saturation zone in the second block. However, in order to limit an impact of the large initial polarisation on the delay time, the gain k_0 outside the range $[U_-, U_+]$ must be substantially reduced.

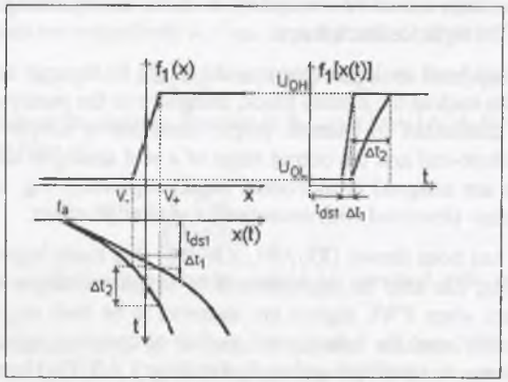

Fig. 3.1. Mechanism of saturation delay

To match the typical timing specifications for a voltage comparator we assume a step function for u_{inp}, so that it changes from U_a to U_b. In the steady state for $u_{inp} = U_a$: $u_{out} = U_{OL}$ and for $u_{inp} = U_b$: $u_{out} = U_{OH}$. By putting $f_0(U_a) = f_a$ and $f_0(U_b) = f_b$, the step response of the first block takes the form

$$x(t) = f_a + (f_a - f_b)\, e^{-t/T_0} \tag{3.2}$$

Hence, the time required for the second block to leave the saturation zone (i.e. for x to rise in time from f_a to V_-) can be found from: $t_{ds} = T_0 \ln[(f_b - f_a)/(f_b - V_-)]$.
Moreover, for large overdrives at the input the signal x rapidly crosses the active zone $[V_-, V_+]$ (see Δt_1 in Fig.3.1), so that

$$f_1[x(t)] \;\approx\; U_{OL} \;+\; (U_{OH} - U_{OL})\,1(t - t_{ds}) \tag{3.3a}$$

$$y(t) \;\approx\; U_{OL} \;+\; [(U_{OH} - U_{OL})(1 - e^{-(t - t_{ds})/T_1})]\; 1(t - t_{ds}) \tag{3.3b}$$

The time constant T_1 can also model the capacitive loading effect at the output similarly to (2.2). The rise delay time for y to change from U_{OL} to the threshold voltage level U_{TH} can be calculated from $t_{dr} = T_1 \ln[(U_{OH} - U_{OL})/(U_{OH} - U_{TH})]$. In particular, for CMOS circuits usually $U_{TH} = (U_{OH} + U_{OL})/2$ and we have $t_{dr} = T_1 \ln 2$. Finally, the output signal u_{out} can be obtained by using the delay mechanism

$$u_{out}(t) \;=\; y(t - t_{di}) \tag{3.4}$$

where t_{di} is the comparator intrinsic delay time.

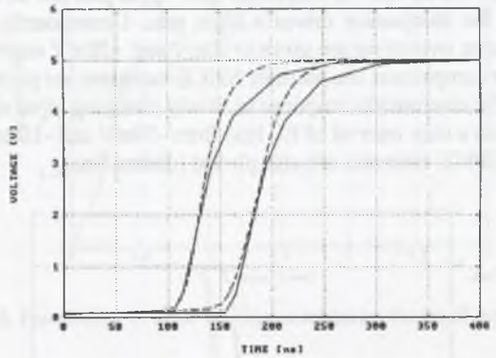

Fig. 3.2. Comparator step responses to different initial polarisation

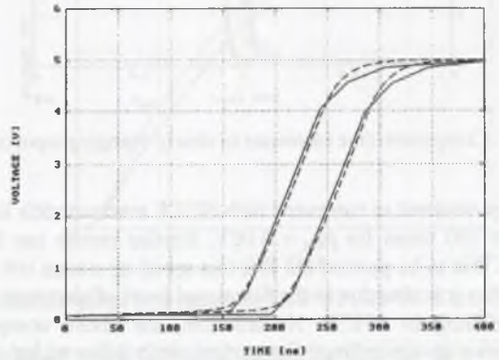

Fig. 3.3. Comparator step responses to different input overdrive

The total delay time of this macromodel is the sum of saturation delay, rise/fall delay and intrinsic delay: $t_d = t_{ds} + t_{dr} + t_{di}$. The simple synthesis concentrates on a limited class of step-input signals. On the other hand, the macromodel behaviour derived represents in some sense the physical phenomena in a real comparator circuit. Hence, it is said to possess the predictive ability feature [CHU80] and is expected to also work well for slowly changing input signals.

Based on the formulas derived the estimates of the macromodel parameters (T_0, T_1, k_0, U_-, U_+, t_{di}) can be found. Preferably a careful scaling procedure should be used for particular comparator specifications to obtain the optimal parameter set [CAS91]. Slowly changing

input signals can be also considered. However, no claim is made regarding macromodel suitability to ideally mimic the real comparator behaviour for all situations.

Here, the macromodel performance for the LM311 comparator is illustrated loaded at its output by a 10pF capacitance and a pull-up resistance of 2kΩ. The optimal parameter set, obtained by the scaling procedure to match SPICE macromodel estimates, is (105ns, 33ns, 0.2, -20mV, +20mV, 5ns). In Figs. 3.2 and 3.3 the time responses to the step inputs obtained with the PWL approximation-based technique are presented. The impact of the input initial polarisation may be seen in Fig.3.2 (-30mV and -100mV respectively with +5mV overdrive). The discrepancy observed between PWL- and SPICE waveforms in their upper part can be neglected in a typical application, e.g. when the comparator drives a logic gate. Consequently, in Fig.3.3 the time responses to different input overdrives are given (+10mV and +20mV respectively with -100mV initial polarisation). For comparison, the accurate SPICE estimates are plotted with dashed lines. In addition, in Fig.3.4 the macromodel response to slowly changing input signals is shown. Both inputs rise linearly within a time interval of (0, 1μs) from -30mV and -100mV respectively up to +5mV. The respective SPICE estimates are also plotted (dashed line).

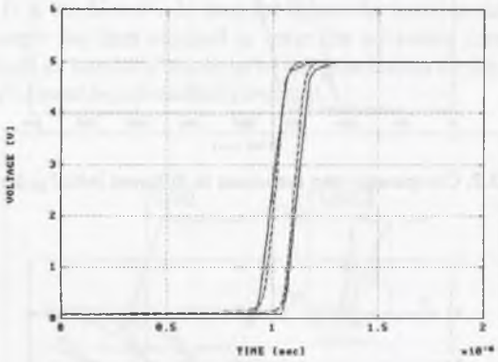

Fig. 3.4. Comparator time responses to slowly changing input signals

The simulation speed-up obtained as compared with SPICE macromodels is up to 1000 times for p_{mx} = 0.08V and up to 500 times for p_{mx} = 0.02V. Similar results can be obtained with the enhanced TR approach. It is to be pointed out that this speed-up comes out not only of using the explicit PWL analysis, but it is also due to the functional level of abstraction used as opposed to the detailed macromodelling in SPICE. Nevertheless, the above comparison seems to be reasonable and may serve as an indirect comparison with other techniques, since SPICE is usually viewed as a kind of standard in analogue modelling.

3.2 Amplifier macromodel

For an amplifier the macromodel synthesis procedure is much simpler. Usually the following specifications must be accounted for: gain, dominant pole, saturation, output resistance and slew rate. For small input/output signals a fully linear macromodel is sufficient. However, to cover a full range of input amplitudes the nonlinear function $f(\cdot)$ and a special slope-limiting mechanism (SLM) [DAB99F] must be used. The SLM does not begin to act until $|\Delta u_{in}| > u_{th}$, where Δu_{in} is an initial increment of the input signal (when starting from the quiescent point) and u_{th} denotes

the threshold voltage that puts the amplifier input stage into saturation [SOL74]. Consequently, each linear segment of u_{in} with amplitude above this threshold is checked for the slew-rate (SR) parameter. If the segment slope $|du_{in}/dt| > SR/k$, its value is reduced to the limit SR/k (k is the amplifier gain) and the next PWL segments are shifted appropriately along the time axis to avoid time discontinuities (see Fig.3.5).

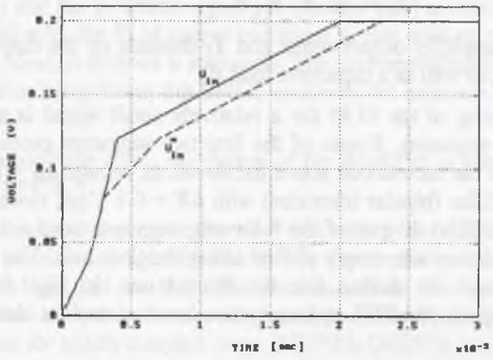

Fig. 3.5. Functioning of slope limiting mechanism for small input (k=50)

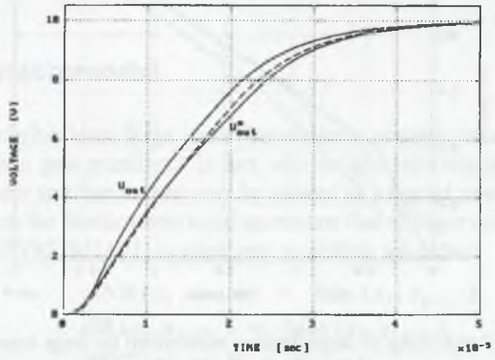

Fig. 3.6. Amplifier time responses for k=50 (SR=0.5 V/μs, f_1=1MHz)

The segment shifting holds well for relatively small input amplitudes. However, when the input signals are larger, a different procedure is preferred for the SLM. In this case the amplifier input stage is deeply saturated, so the output continues to slew although the actual input slope does not exceed SR/k any more. To follow this effect, the SLM has to decide how to proceed with the slowly changing segments (which follow the fast one). For this purpose, at the time instant for which the slope limiting is likely to stop (for smaller amplitudes), the difference u-y/k is checked. If it is smaller than u_{th}, the next segments are shifted (as mentioned earlier). Otherwise, the SLM does not change the slope SR/k at its output until the original signal u is crossed (see Fig.3.6). As a consequence, the remaining u need not be shifted.

As well as the SLM, two cascaded inertial blocks are required. The first gives gain and the dominant pole, whereas the other one serves as an output stage. Denoting the signal obtained from SLM by u^*_{in}, for the first block we have

$$T_0\dot{x} + x = k u_{in}^{*}, \quad when\ |k u_{in}^{*}| \le U_{os} \tag{3.5a}$$

$$T_0\dot{x} + x = U_{os}, \quad when\ |k u_{in}^{*}| > U_{os} \tag{3.5b}$$

where T_0 is the inverse of a dominant pole frequency ω_0 ($T_0 = 1/\omega_0$) and U_{os} is the output saturation voltage. For the second building block

$$T_1\dot{y} + y = x \tag{3.6}$$

where $y = u_{out}$ is the amplifier output signal and T_1 depends on the output resistance R_{out} and output capacitance C_{out} as well as a capacitive load C_l.

In Fig.3.5 the functioning of the SLM for a relatively small signal is shown. The u_{in} signal consists of four PWL segments. Slopes of the first two segments exceed SR/k, but the SLM limits only the slope of the second one above the threshold voltage u_{th} = 80 mV. It corresponds to a noninverting amplifier (bipolar transistor) with SR = 0.5 V/μs, closed loop gain k=50 and ω_0=125600 rad/s (f_1=1MHz). Slopes of the following segments need not be limited since they are less than SR/k, and they are simply shifted along the time axis. The solid line denotes the original input signal and the dashed line the limited one. In Fig.3.6 the amplifier output waveforms for u_{in} are given (the PWL approximation is not plotted for clarity of the diagram).

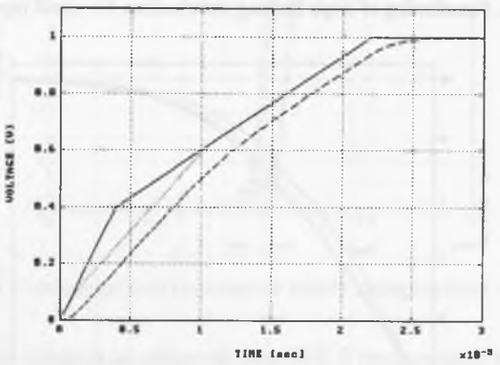

Fig. 3.7. Functioning of slope limiting mechanism for large input (k=10)

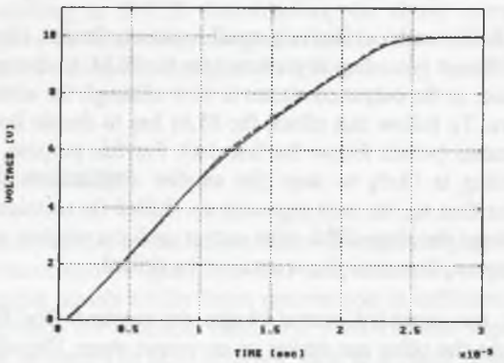

Fig. 3.8. Amplifier time responses for k=10. PWL waveform is not plotted for clarity of diagram

The u^{*}_{out} solid line has been obtained with the SLM, whereas the u_{out} solid line without it. For comparison the SPICE estimate of the u^{*}_{out} is also plotted (as the dashed line).

For other values of k ($k\omega_0$=const.) the time responses for various input signals also give a good approximation to SPICE estimates. In Figs.3.7 and 3.8 another example of the input and output waveforms for k=10 is given. A slope of the first input segment is limited after crossing u_{th}. When the output of SLM (dotted line in Fig.3.7) reaches 0.4V, the difference between u (solid line) and the feedback signal y/k (dashed line) is checked. Since in this case it is approximately 160 mV, the SLM output continues to rise with no change of slope until u waveform is crossed. Next, it follows u segments. The corresponding amplifier time response is shown in Fig.3.8. For comparison the SPICE macromodel estimate is plotted with dashed line.

By means of this approach the PWL simulation of the amplifier is also up to three orders of magnitude faster than SPICE.

Using a more natural way to represent the slew-rate effect, as with a closed loop OpAmp [SOL74], leads to a problem of tight feedback. The input of the nonlinear SLM no longer depends on the initial Δu_{in} but on the difference $u_{in}-y/k$. In this case the presented PWL approximation algorithm requires an iterative approach to maintain accuracy. However, because of its slow convergence for tightly coupled loops [NEW84, DEB87], it is undesirable to give up the simple approach used here. On the other hand, this will result in some loss of accuracy, particularly when narrow pulses are applied to the amplifier input.

3.3 Logic gate macromodel

At the functional simulation level, logic units like registers, counters, multiplexers or memories are dealt with rather than gate primitives. In fact, also simple gates can play important role in a digital (or mixed) system and hence, they may be viewed as a special case of the functional unit. Moreover, they perform the fundamental logic operations that all logic units are based on. Using the PWL approach [DRY85, RUA91] to logic gate modelling we define:

$$AND(x_1, x_2, \dots) = Min(x_1, x_2, \dots) \tag{3.7a}$$

$$OR(x_1, x_2, \dots) = Max(x_1, x_2, \dots) \tag{3.7b}$$

$$NOT(x) = V_L + V_H - x \tag{3.7c}$$

where V_L, V_H, are respectively the low- and high-level logic voltage. Apparently, the right-hand side operations in (3.7) are well suited to proceed with PWL signals, which are in this way equivalent to multiple-value logic signals. In addition to Boolean L and H, at least two extra states R (rise) and F (fall) are required (accomplished by X as uninitialised state). Consequently, any logic signal may be represented as an ordered set ...,(t_i, s_k), (t_{i+1}, s_j),... where $t_i < t_{i+1}$ and $s_k \neq s_j$ are different logic states. In particular, the R- comes usually after the L-state and the F- after the H-state. Some changes are forbidden, e.g. $H \rightarrow R$ or $H \rightarrow L$.

An equivalent PWL signal can be obtained by using constant voltages for the Boolean H- and L-state, and linearly changing voltages for R- and F-state. For some signal (see Fig.3.9) defined as the sequence: (t_0, L), (t_1, R), (t_2, H), (t_3, F), (t_4, L) the voltage rate of its rising edge that comes at t_1 is equal $r = (V_H - V_L)/(t_2 - t_1) > 0$. Consequently, for the falling edge coming at t_3, we have $r = (V_L - V_H)/(t_4 - t_3) < 0$. Moreover, to handle some unknown states we assume the rising and falling edges to start or stop at any intermediate value V_i, so that $V_L \le V_i \le V_H$.

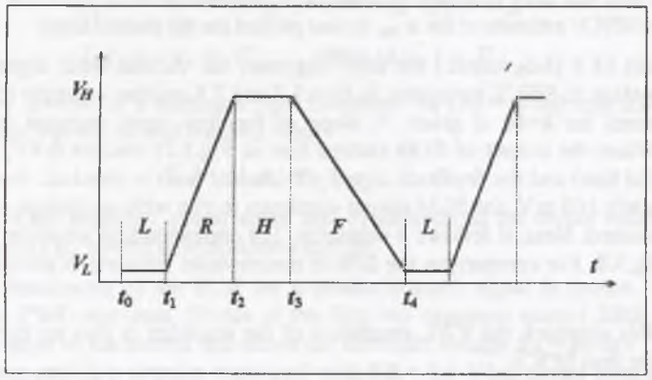

Fig. 3.9. Representation of PWL signal fo logic gate modelling

In addition to the formulas (3.7), the PWL logic gate macromodel needs a slope acceleration mechanism [RUA91], so that the output rate is

$$r_a = \beta r_{inp}, \qquad r_a \in [F_{mx}, R_{mx}] \tag{3.8}$$

This rate is limited by F_{mx} for the falling- and by R_{mx} for the rising edge as stated. The coefficient β is chosen individually for each type of a gate. If C_s is a self-capacitance of a gate and C_o is its fanout capacitance, the output rate r_a is reduced

$$r_{out} = \frac{C_s}{C_s + C_o} \cdot r_a \tag{3.9}$$

To propagate u_{inp} with a new slope r_a, the corresponding primary PWL events are replaced with the new ones as shown in Fig. 3.10.

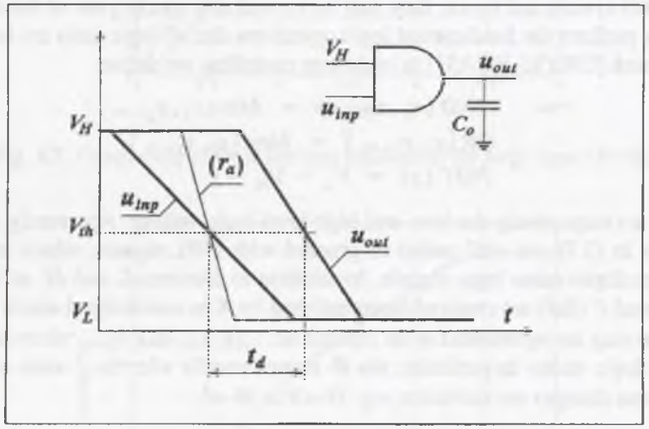

Fig. 3.10. PWL signal conversion in logic gate

The delay time depends on the input rate r_{inp} and also on C_o (not only on r_{inp} as assumed in [RUA91]). To calculate the delay time t_d for MOS gates we use the formula with a product of the mentioned above parameters considered as its argument [CHN88]

$$t_d = C_o \cdot f(C_o \cdot r_{inp}) \tag{3.10}$$

The functions $f(\cdot)$ are obtained individually for different gates based on SPICE estimates [ZAJ98] and are implemented in a form of tables to support the look-up technique.

This model may be referred as fully behavioural with a logic function defined by eqns.(3.7), and timing specifications defined by eqns.(3.8) through (3.10). In contrary to this, using a more natural approach one comes to structurally oriented modelling. As a result the timing part of the behavioural model can be represented with the inertial building block, as shown in Fig.3.11. Clearly, this model needs careful parameter adjustment to meet the specifications, and should be provided with the PWL approximator at its output. Alternatively, the inertial block can be discretised due to the enhanced TR algorithm.

On the other hand, the enhanced structural model of a logic gate, in which the delay time and output slope time may be distinguished, can be patterned after the comparator macromodel derived in Section 3.1.

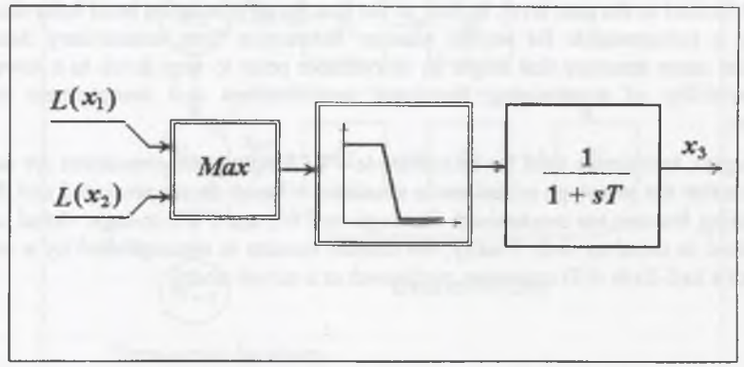

Fig. 3.11. Structural PWL macromodel of NOR gate

Basically, those macromodels are limited to simple gates. For more complicated units different approach is used to describe their logic functions and delays as well. In this case the input- and output stage of a unit are separated from each other, so that the output rate depends only on C_o (eqn.(3.9)), and r_a is assumed to be a constant. Consequently, the delay time can be represented as a sum of: the input-stage delay t_{di}, intermediate-stage delay t_{dt} and output-stage delay t_{do}.

$$t_d = t_{di} + t_{dt} + t_{do} \tag{3.11}$$

where $t_{di} = f_1(r_{inp})$, $t_{dt} = const.$ and $t_{do} = f_2(C_o)$.

Also the formulas (3.7) alone are insufficient. In fact, they can be accepted in modelling the input and output gate stages of a complex unit. In this case they usually play a role of virtual converters between the logic- and PWL domain, whereas for the internal part of the unit a behavioural description is preferred. This problem is discussed in more detail in Chapter 4.

4. MIXED-MODE PWL/ LOGIC MACROSIMULATION

Since it is difficult to propagate non-Boolean states through complex logic blocks, using a mixed-mode technique to macrosimulation of mixed A/D networks seems to be a reasonable approach. In particular, in this context we address the intermediate values $V_i \in (V_L, V_H)$ performed by the PWL approach. Because of it, for complex logic blocks logical modelling would be preferred. Hence, the relevant mixed models would require logic-to-PWL and PWL-to-logic signal converters [SAL94]. As compared to previous work, the fully unified PWL treatment of A/D networks, introduced in [RUA91], is no longer applied here. In contrast to that unified approach, using the mixed-mode technique enables behavioural logic modelling that is not limited to the gate level. In fact, at the functional simulation level behavioural logic modelling is indispensable for several reasons: abstraction from unnecessary details (also from model inner structure that might be unavailable prior to step down to a lower design level), capability of emphasising functional specifications and compactness of model description.

In this chapter, techniques used for mixed-mode PWL/ logic macrosimulation are addressed. Algorithms that the prototype mixed-mode simulator is based on are presented and their most distinguishing features are emphasised. The logic-to-PWL and PWL-to-logic virtual converters are discussed in detail as well. Finally, the chapter content is accomplished by a simulation example of a half-flash A/D converter, performed as a mixed model.

4.1 Simulation algorithm

As mentioned in Section 1.1, modern simulation techniques, in particular the mixed-mode simulation, tend to adopt mechanisms typical of logic simulation [BRE75]. The common use of *event-driven* and *selective-trace* modes proved to be a unifying mechanism in mixed-mode simulation [SAL94]. Following that experience, in the presented mixed PWL/logic approach the same principles are exploited.

To establish this kind of simulation, a *time-queue* and an *event-scheduler* are introduced. As opposed to logic simulation, special attention must be paid to the PWL simulation when defining an *event*. Observe that the PWL-events can be generated by subsequent breakpoints (t_k, V_k) in a natural way. In fact, two subsequent points (t_k, V_k), (t_{k+1}, V_{k+1}) are required for an event to be determined at the time instant t_k. In practice, this event can also be defined as a change of the corresponding rate $r_k = (V_{k+1} - V_k)/(t_{k+1} - t_k)$ assigned to (t_k, V_k).

Whenever the event at a block output occurs, it is possible to *schedule* all of its fanouts to be processed (blocks directly controlled from that node). Since the only blocks that are processed are those which are affected directly by the event, this technique is referred to as *selective-trace*. Following the selective-trace principle, only the active parts of the simulated network are analysed. Active blocks generate events (accordingly PWL- or logic ones) each time they produce a new PWL breakpoint or make a transition to a new logic state, respectively. Thus, processing an event means analysing the fanout blocks adjacent to their input node, activated recently. The fanout blocks are found from the fanout table that is available after compiling a network netlist. Consequently, the analysis follows (traces) the signal propagation, whereas the remaining (latent) parts of the network are skipped (and if inactive at all, need only to be initialised). As a result the relevant blocks are scheduled and next processed due to the signal flow in a network. Also self-scheduling of a PWL block is required when the output produces a new event. Clearly, in this case the block has to be processed further to follow its transient behaviour, regardless any new events at the block input occur.

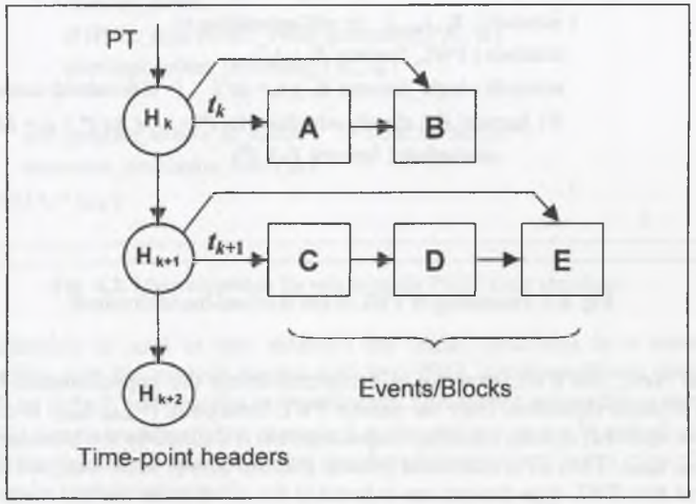

Fig. 4.1. Time queue as indexed list

The event-scheduler controls the order of simulation based on a *time-queue*, which is organised as an indexed list (see Fig.4.1). The time-queue contains time-point headers that point to queue-entries grouped into true event-lists (including names of blocks to be processed), each assigned to a given time-point. PT denotes the present time pointer. If a new event is encountered, the corresponding fanout blocks can be added to the queue thanks to the list of time-point headers and an extra pointer that points to the end of an event-list. This structure differs from the standard time-queue indexed list in that the time-point headers are arranged as a list rather than a vector (array) where time instants are integer multiples of some prescribed time-step Δt [SAL94]. Using the list seems to be a more reasonable approach. Otherwise, a very fine Δt would be required to schedule PWL events, resulting in a large number of time-point headers, with most of them pointing to no events.

An algorithm for processing the PWL-event is depicted in Fig.4.2, where E_i is a block to be analysed and t_k is a time of activation (current event). First, the current input x_{inp} and previously predicted output state of E_i are identified. The actual segment of x_{inp} is converted into the u segment (as defined in Section 2.1), and the amplitude V_k is updated due to the relation between t_k and t_{next}, which represents the output breakpoint predicted previously. If $t_k = t_{next}$, the predicted output V_{next} is taken as the actual output V_k. Otherwise, V_k has to be

```
PWL_event_processing ( E_i , t_k )
    { get_input&state ( E_i , t_k );      (* input and predicted output segment *)
      convert_input_segment( E_i , t_k);          (*  u = f(x_inp)  *)
      if (t_next > t_k)  V_k ← [V_next – (V_next - V_k-1)·(t_next - t_k)/(t_next - t_k-1)]
        else V_k ← V_next;                        (*  t_next = t_k  *)
      get_next_point;              (* find next breakpoint (V_next , t_next) *)
      out ← (V_k, t_k)             (* update output *)
      out_next ← (V_next , t_next);   (* predict next output *)
      if ( E_i active )            (* output of E_i changes for t > t_k  *)
          { schedule ( E_i , t_next );  (* self-scheduling *)
            schedule ( PWL_fanouts( E_i ), t_k );
            schedule ( logic_fanouts( E_i ), t_k + Δt );   (* if threshold crossed *)
            if ( fanouts( E_i ) already scheduled for t* > t_k or for t* > t_k + Δt )
                  unschedule ( fanouts( E_i ), t*)
          }
    }
```

Fig. 4.2. Processing of PWL event in mixed-mode simulator

recomputed. Next, the PWL analysis is performed using the approximation-based or the enhanced TR-based algorithm. Only the nearest PWL breakpoint (V_{next}, t_{next}) is computed (i.e. V_{next}, t_{next} are updated) and the resulting output segment is defined by the boundary points (V_k, t_k) and (V_{next}, t_{next}). Then E_i is scheduled (placed into the queue) to be analysed again for $t_{next} > t_k$. Observe that PWL-type fanouts are scheduled for t_k, whereas the logic-type fanouts not until the logic threshold is crossed, i.e. for $t_k + \Delta t$.

To avoid undue backtracking, the event-scheduler controls carefully the incoming PWL events (breakpoints). In particular, if an event appears at t_k, and $t_k < t^*$, then the previously scheduled event at t^* for the fanout block is discarded, the new event is scheduled for the instant t_k, so the block output will be updated for t_k.

A slightly different algorithm is required for logic events associated with behavioural logic models. It is because logic units differ substantially in their behavioural functionality. The other reason is that the particular logic inputs of a model feature usually different specification, and different violations may occur for them (such as spikes, setup or hold time violations). Hence, reporting of violations are incorporated into a logic model. Besides, the source of activation must be recognised to proceed an event effectively (see Section 4.3).

The overall simulation flow is presented in Fig.4.3. The blocks are analysed subsequently within the main simulation loop for the actual simulation time t_k. After all events (blocks) from the relevant list have been processed or cancelled (discarded) the list becomes empty, and it should be removed from the queue. Since a size of the time queue is limited, some later events are placed on an extra list, called *remote list* to avoid overflow. The remote list plays a role of a buffer, and if any list is removed from the time-queue, a new list of pending events (if exist) from the remote list can be put into the queue. Finally, the time is advanced to continue simulation.


```
initialise_simulation
    { compile_netlist; compile_input_stimulus;
      arrange_time_queue; arrange_remote_list
    }
t_k ← 0;                    (* reset simulation time *)
repeat
    enter_time_queue (t_k);
    while ( event_list not empty )
       { identify_block;
         if (PWL_type ) PWL_event_processing ( E_i , t_k )
         else logic_event_processing ( E_i , t_k )
       }
    add_pending_events_to_queue;    (* from remote list *)
    increment_simulation_time ( t_k )
until ( t_k = t_stop )
```

Fig. 4.3. Main algorithm for mixed-mode PWL/ logic simulator

The main algorithm is used to also establish the initial conditions in a network to be simulated. In this case the analysis begins with zero-PWL initial conditions, logic X-states (uninitialised) by default and constant external inputs, all together assumed as a starting point. Once the PWL signals stabilise (stop changing) at the rate $r < r_{min} \approx 0$, and all X-states are replaced by determined logic states, the pure simulation may begin. Clearly, predefined initial conditions may be introduced as well.

4.2 Mixed-mode interfacing and synchronisation

An interface between the PWL- and logic sub-simulator plays an important role. After compiling a netlist, all PWL-units with fanout of logic type are equipped with PWL-to-logic converters, and vice versa, logic units that control PWL-type blocks, with logic-to-PWL converters. The relevant converters that provide this interfacing are, in fact, virtual objects since they do not represent directly any physical elements in a network. Basically, at lower levels of abstraction the virtual converters are dependent on technology (e.g. TTL, ECL, MOS), and when defining, one should pay special attention to the loading or bi-directional coupling effects [SAL94, DAB96K].

Here, at the functional level, this task appears much simpler unless detailed timing is required. On the other hand, the nowadays designs are oriented mainly towards MOS circuits, for which bi-directional coupling might usually be neglected, and capacitive loading effects are crucial. These, in turn, can be effectively modelled with the basic building blocks as stated in Section 2.1. Floating elements such as transmission gates cannot be represented separately and should be incorporated into the blocks (as mentioned earlier). As a consequence, the

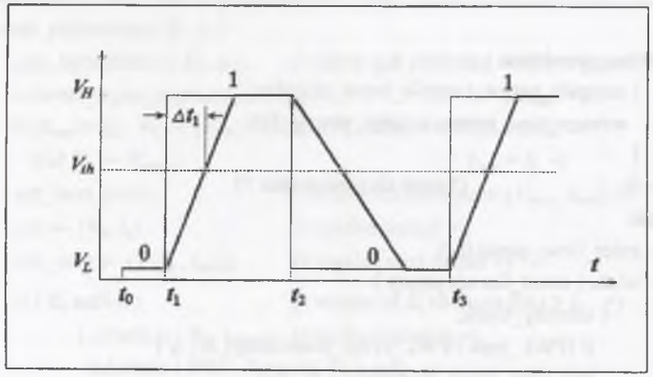

Fig. 4.4. Principle of logic-to-PWL conversion

required logic-to-PWL converter can be based on the PWL-model of a gate (Section 3.3), where some standard value for the slope r_a may be assigned to a corresponding logic event. However, this kind of conversion is somewhat cumbersome. That is, a logic block should output the respective logic event at the time instant, for which the resulting PWL segment should start (rather than matching directly the logic delay due to voltage threshold). The principle of logic-to-PWL conversion is shown in Fig. 4.4. The thin line represents logic input to the converter, whereas the thick line its PWL output (with slopes r_{out} defined as in Section 3.3). The logic events occur at the time instants t_1 through t_3. Apparently, they come earlier than they would come unless the conversion were required for the logic block (see Δt_1 for the event at t_1).

To process an event, the logic-to-PWL converter should recognise its logic input and actual state. In particular for t_1, the converter updates the output as (V_L, t_1), and computes the next breakpoint to appear Δt_R later with the amplitude V_H. Then, the scheduler schedules the relevant fanouts to be processed for t_1, and the converter itself to be processed for (V_H, $t_1+\Delta t_R$).

The converter may be treated as a kind of the building block. An algorithm for processing a logic event in the logic-to-PWL converter is depicted in Fig.4.5. First, an actual output state is recognised using the *state* variable. For example, the *state* variable is reset from '**0**' to '*R*' to indicate that the output begins rising when the input changed from **0** to **1**. Next, suppose that the logic input changes from **1** to **0** before the amplitude V_H is reached. Consequently, the converter is scheduled and processed for that time. In this case, t_{next} (computed previously) is found to be bigger than the actual event time t_k, so the output is updated as (V_k, t_k) and the next breakpoint with the amplitude V_L is calculated (state is reset to '*F*'). Unless the logic input is too short (such as in Fig.4.4), the amplitude V_H is reached as the nearest event to be processed, so for this event we would find $t_k = t_{next}$ (state is reset from '*R*'to '**1**'). To define the PWL output segment, the variables *out* and *out_next* are updated.

Clearly, the PWL-to-logic conversion is simpler since it is a conversion from lower- to higher level of abstraction. This involves removing unnecessary details from the PWL waveform. The resulting logic signal switches between 0 and 1 after crossing the prescribed logic threshold V_{th}. In Fig.4.6 the input and output signals of the converter are plotted respectively


```
case state of
    '0': { V_k ← V_L ; state ← 'R';
           t_next ← t_k + Δt_R ; V_next = V_H };
    '1': { V_k ← V_H ; state ← 'F';
           t_next ← t_k + Δt_F ; V_next = V_L };
    'R': if (t_next > t_k ) { V_k ← [ V_H − r_R (t_next - t_k) ]; state ← 'F';
                              t_next ← [ t_k + (V_k − V_L)/r_F ]; V_next = V_L }
         else { V_k ← V_H ; state ← '1';
                t_next ← t_end ; V_next = V_H }
    'F': if (t_next > t_k ) { V_k ← [ V_L + r_F (t_next - t_k) ]; state ← 'R';
                              t_next ← [ t_k + (V_H − V_k)/r_R ]; V_next = V_H }
         else { V_k ← V_L ; state ← '0';
                t_next ← t_end ; V_next = V_L }
    end case;
out ← (V_k, t_k)                          (* update output *)
out_next ← (V_next , t_next);             (* predict next output *)
```

Fig. 4.5. Processing of event in logic-to-PWL converter

with the thick and thin line. However, when precise timing is required, the PWL-waveform slope should be taken into account to compute the delay t_{di} of the logic front-end stage (see eqn.(3.11)). The PWL-to-logic converter has a built-in mechanism to check for crossing the logic threshold. If a PWL segment crosses the level V_{th} the corresponding time instant is calculated (e.g. t_1 in Fig.4.6) and a logic event for the fanout block(s) is scheduled at that time.

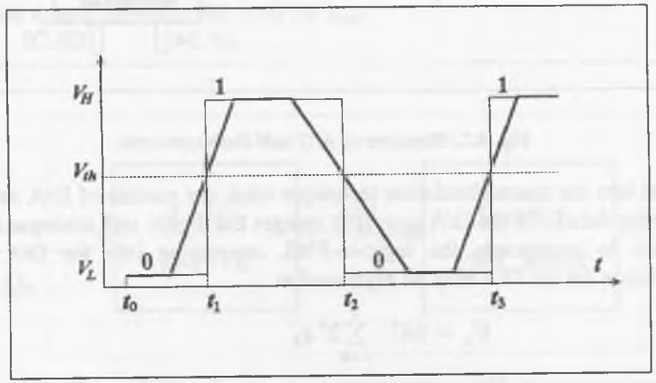

Fig. 4.6. Principle of PWL-to-logic conversion

4.3 Macrosimulation example

For illustration a simulation example of an A/D converter is presented. The results have been obtained by means of the prototype mixed-mode functional-level simulator, presented in this chapter. As stated, the simulator is event-driven, where the subsequent PWL breakpoints (V_i, t_i), like the logical (S_k, t_k), are defined to be events. Scheduling of the events is based on the time-queue mechanism.

In Fig.4.7 a functional structure of the eight-bit half-flash A/D converter is given. Here, the behavioural specification is also necessary to define the macromodels, particularly for the digital units. Three-valued logic (**1**, **0**, ***Z***) plus uninitialised ***X***-state are used to cope with their timing specifications, e.g. the hold- or set-up time [BRE75, SAL94]. For the T/H, the amplifier macromodel is used together with a memory mechanism and a switch that performs multiplication of two PWL signals (the input and the control one). Also the MUX macromodel is based on switches and the inertial block at its output. For the A/D flash a signal divider and comparator macromodels are used to provide a 16-state PWL signal. The resulting 4-bit data N3..N0 is obtained with a digital decoder modelled by behavioural logic description. Its front-end is provided with a PWL-to-logic virtual converter. The Ctrl unit is modelled behaviourally as well. For the D/A a mixed model is used so that logic-to-PWL conversion is required for it.

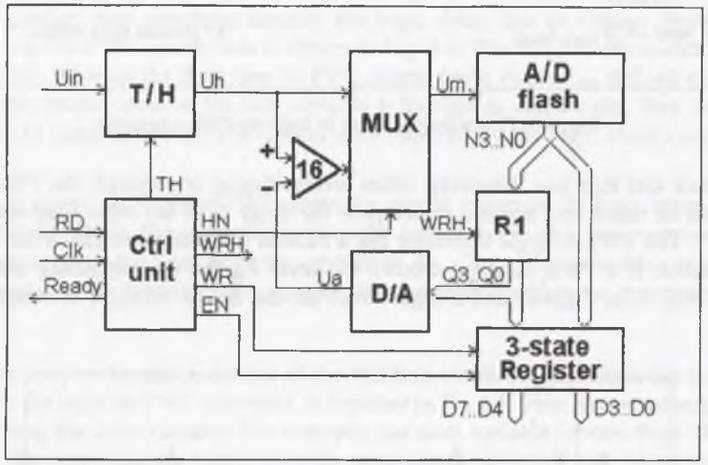

Fig. 4.7. Structure of A/D half-flash converter

To give insight into the macrosimulation technique used, the models of D/A and R1 will be presented in more detail. As the D/A converter merges the digital and analogue functionality, it is reasonable to incorporate the logic-to-PWL conversion into the D/A model. The constitutive relation for the D/A may be expressed as

$$U_a = \Delta U \cdot \sum_{k=0}^{3} 2^k q_k \qquad (4.1)$$

where ΔU is the resolution. The q_k parameters are set either to 0 or to1 with respect to the digital input $\mathbf{Q_3..Q_0}$. Note that each component of (4.1), i.e. $U_k = \Delta U \cdot 2^k q_k$, can be represented as the logic-to-PWL converter driven by Q_k, and the output changing between $V_H = \Delta U \cdot 2^k$


```
if ( state = 'steady' ) { V_k ← V_next ; state ← 'R/F';
                          V_next ← ΔU·decimal(Q3..Q0);
                          t_next ← [ t_k + |V_next - V_k| / r_a ]}
else                                                    (* state = 'R/F' *)
    if (t_next > t_k )  { V_next1 ← V_next ;
                          V_next ← ΔU·decimal(Q3..Q0);
                          V_k ← [ V_next1 + r_a (t_next - t_k) sgn(V_next - V_next1) ];
                          t_next ← [ t_k + |V_next - V_k| / r_a] }
        else { V_k ← V_next ; state ← 'steady';         (* t_next = t_k *)
               t_next ← t_end }
out ← (V_k, t_k);                                       (* update output *)
out_next ← (V_next , t_next);                           (* next output *)
```

Fig. 4.8. Processing of event in 4-bit logic-to-PWL converter

and $V_L = 0$. However, as logical events for the bits $\mathbf{Q_0}$ through $\mathbf{Q_3}$ are assumed to occur simultaneously (are synchronised by the WRH signal), a single 4-bit logic-to-PWL converter would be sufficient for them. As compared to the model shown in Figs. 4.4 and 4.5, here the fixed levels V_H and V_L should be replaced with the actual voltage levels calculated from (4.1). When a logic event occurs on $\mathbf{Q_3..Q_0}$ at t_j, the converter is scheduled (based on the fanout table) to be processed Δt_d later, i.e. after the prescribed delay of the input stage. Once the simulation time is advanced to $t_k = (t_j + \Delta t_d)$, the converter is popped from the time queue to perform the analysis. An algorithm for the relevant logic-to-PWL conversion is presented in Fig.4.8. Here, the *state* variable can be set either to '*R/F*' (when the converter output is changing) or to '*steady*' (when the actual voltage level has been reached). The updated variable V_{next} provides the output amplitude of the next breakpoint to appear for t_{next}. If the output is changing and the current event precedes the end of the current output segment (i.e., $t_{next} > t_k$) then the actual output V_k is recomputed. Next, the output segment is defined using *out* and *out_next* variables. Finally the scheduler will schedule the relevant fanout block for the current time t_k, and the converter itself for t_{next}.

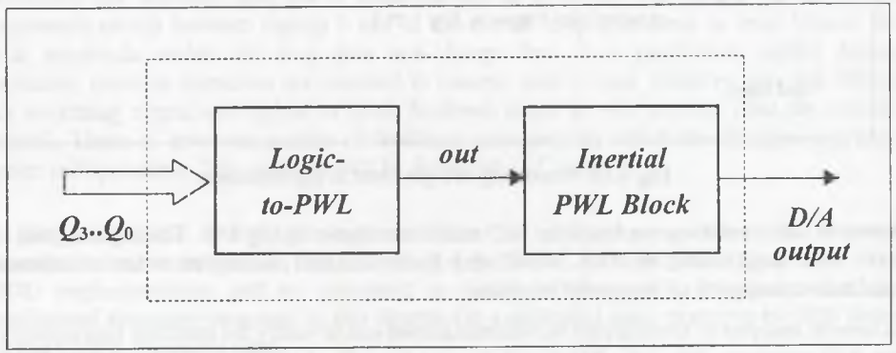

Fig. 4.9. Structure of D/A model

A complete model of the D/A unit composed of the 4-bit logic-to-PWL converter and the PWL inertial block is presented in Fig.4.9. A model of the inertial block follows the algorithm presented in Fig.4.2.

The other model, concerning the R1 register is depicted in Fig.4.10. When an event occurs at the register data input (N3..N0) or strobe input (WRH), the involved *active* variable is set to '***input***' or '***strobe***', respectively. Once the R1 model is popped from queue, the value of *active* indicates the reason for activity. Apparently, *active* = '***output***' is preceded by the active strobe. The model checks for timing specifications, such as setup or hold time, and it reports the relevant violations when necessary. However, no ambiguity for the register output is introduced in that case. To follow this idea, some indeterminate state ***U*** would be required that, on the other hand, tends to spread in a network. As a consequence, the timing obtained is likely to be even more indeterminate in some cases [ARM88, BRE76]. The scheduling of logic events used here resembles that of the PWL model (Fig.4.2)

```
case active of
  'strobe':  if ( strobe = 1 ) and ( data <> data_in )          (* rising edge *)
               { data ← data_in;  t_next ← ( t_k + t_strobe );
                 schedule( Reg, t_next ); active ← 'output';    (* self scheduling *)
                 get_last_input; if (t_k - t_last_input < Δt_setup )
                                   report( 'setup time violated for', Reg, 'at', t_k )
               }
             else if ( strobe = 0 ) { get_last_strobe;          (* falling edge *)
                                      if (t_k - t_last_strobe < Δt_strobe )
                                        report( 'strobe width violated for', Reg, 'at', t_k )
                                      }
  'input':   { get_last_strobe; if (t_k - t_last_strobe < Δt_hold )
                                  report( 'hold time violated for', Reg, 'at', t_k )
             }
  'output':  { data_out ← ( data, t_k );
               schedule( Fanouts( Reg ), t_k );
               if ( Fanouts( Reg ) already scheduled for t* > t_k )
                 unschedule( Fanouts( Reg ), t* )
             }
  end case;
```

Fig. 4.10. Processing of logic event in register model

Some of the simulation results of the A/D model are shown in Fig.4.11. The logic signals also have been represented as PWL waveforms (with standard slopes) in order to enhance a qualitative similarity of the model behaviour.

A careful analysis of the relevant waveforms allows one to verify the assumed functionality of the A/D converter. When the RD signal changes from low to high, the T/H unit switches into the hold mode and the 3-state register outputs the old data D07. Next, on the first falling clock

edge the high nibble is written into the R1 register (the MUX transmits so far the signal Uh). The high nibble Q03 is fed back via the D/A, amplifier and MUX to the A/D flash input. This loop closes after the MUX switches to the amplifier output and the A/D provides the low nibble. On the next falling clock edge, first the output register turns into the high impedance state, next the high and low nibbles are written together into this register, and the T/H starts to follow the input signal. If RD remains high the output register is transparent and a new data is available at its output D07 (see change 92H → F1H). In this case the Rdy signal turns to high.

The input signal Uin rises and falls linearly between 0V and 5V and is sampled 4 times. The Um is the most complicated waveform shown, since for some time interval it follows the Uh and otherwise the amplifier output. After switching the MUX, first it changes rapidly and next it slews due to the amplifier slew rate equal 15V/μs. For the accuracy of the PWL waveforms p_{mx} = 50 mV has been chosen. The PWL approximation-based algorithm has been used.

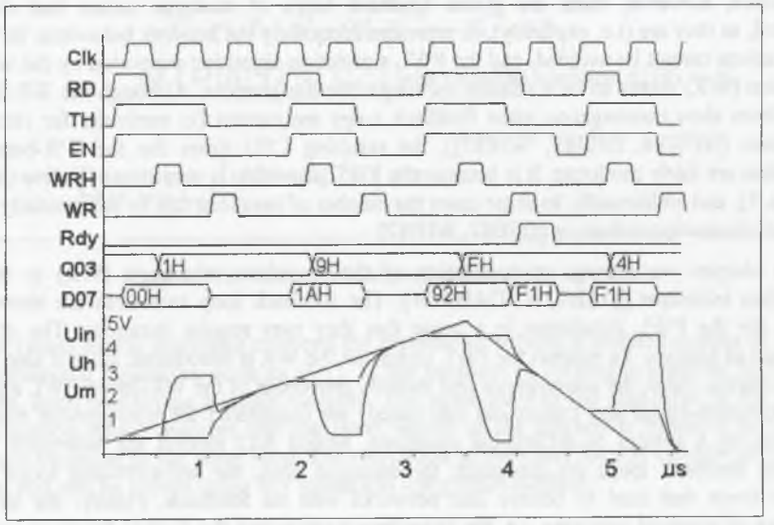

Fig. 4.11. PWL waveforms of A/D half-flash converter

Observe that a kind of feedback structure is faced in the above example. The blocks constituting the feedback loop are of analogue, digital or mixed A/D nature. As the digital components switch between logical 0 and 1, the overall loop gain turns to zero. Hence, the signal amplitude within the loop does not change free. It is particularly useful during simulation, since no iterations are required to process such a loop. Observe also that blocks with switching output are typical of most feedback loops in A/D circuits (that are usually clocked). There is, however, a class of feedback structures for which the iterative approach appears indispensable. This problem will be discussed in Chapter 5.

To put the mixed PWL/logic approach into a broader perspective, the relevant macrosimulation technique has been developed in VHDL [ASH90, LIP91] as well. The VHDL implementation will be presented in detail in Chapter 7. It appears that the experimental simulator presented in this chapter (as a dedicated one), performs by 30% faster than the VHDL simulator used.

5. SUPPORT BY RELAXATION TECHNIQUE

The techniques outlined in Chapter 4 proved to be effective for a typical A/D simulation task performed at the functional level. The system to be modelled should consist of unidirectional blocks. These should embody the existing local couplings, which here are not allowed to be represented separately. As observed in Section 4.3, the presented techniques are capable also of simulating some feedback loop structures with no need of iterations.

In practice, however, there are global feedback loops of analogue nature that should be modelled, as they are (i.e. explicitly), to represent adequately the network behaviour. In this case the iterations cannot be avoided, and the PWL simulation algorithm supported by the waveform relaxation (WR) seems to be a suitable technique for this purpose. Although the WR is said to suffer from slow convergence when feedback loops are present (in particular for circuit-level simulation [NEW84, DEB87, WHI87]), the resulting CPU times for the WR-based PWL simulation are fairly moderate. It is because the PWL algorithm is very time effective (as shown in Chpt. 3), and additionally, in some cases the number of iterations can be substantially reduced with the windowing technique [DEB87, WHI87].

In this chapter we discuss an application of the waveform relaxation (WR) to the PWL simulation technique [DAB97W, DAB99W]. The feedback loop structures are shown to be critical for the PWL simulation in a sense that they may require iterations. The chapter is organised as follows. To support the PWL technique the WR is introduced, since it also operates on waveforms. Next, the convergence and stability properties of the WR-based PWL algorithms (approximation-based and trapezoidal rule-based) are considered for homogenous models, all described by a system of differential equations. Mixed A/D models are addressed as well. Different feedback loops are discussed: the analogue ones, the self-switching loops and the clocked loops that tend to behave like networks with no feedback. Finally, the simulation examples of practical networks, i.e. the telemetric receiver and the digit-to-frequency converter (D/f) are presented.

5.1 Waveform relaxation PWL algorithm

If a precise value of a signal to be fed back in a loop structure is unavailable, the simulation of such a loop must be organised as an iterative process. This happens usually in case of analogue loops, for which all the involved loop units are active simultaneously. Following the selective trace technique the Gauss-Seidel relaxation algorithm seems to be well suited for this purpose. To discuss this problem consider a system composed of n basic building blocks (generalised blocks) with multiple inputs, all described by uniform state-space equations:

$$\dot{x}_k = f_k(x_1, x_2, \ldots x_n, u), \qquad k = 1, 2, \ldots n \tag{5.1}$$

where $f_k : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ for $t \in \langle t_0, t_{stop} \rangle$ are PWL functions, and $u(t) \in \mathbb{R}^m$ denotes here a vector of external input PWL signals. The standard Gauss-Seidel scheme applied directly to (5.1) (also referred to as the waveform relaxation Gauss-Seidel) proceeds as shown in Fig.5.1 [WHI87]. The upper index j stands for the iteration count.

$j \leftarrow 0$;
guess waveforms $x^0(t)$, $t \in [t_0, t_{end}]$ such that $x^0(t_0) = x_0$;
repeat { $j \leftarrow (j+1)$;
  foreach ($k \in [1, .. n]$)
    solve
      $\dot{x}_k^j = f_k(x_1^j, \ldots x_k^j, x_{k+1}^{j-1}, \ldots x_n^{j-1}, u)$
    for ($x_k^j(t)$; $t \in [t_0, t_{end}]$) with the initial condition $x_k^j(0) = x_{i0}$;
  }
until ($\max_{1 \le k \le n} \max_{t \in [t_0, t_{stop}]} \left| x_k^j(t) - x_k^{j-1}(t) \right| \le \varepsilon$)

Fig. 5.1. WR Gauss-Seidel algorithm for solving eqns.(5.1)

The WR Gauss Seidel algorithm converts the problem of solving a coupled system of n first-order differential equations, such as (5.1), to the problem of solving n separate differential equations, each containing a single variable. The outer loop of the algorithm is the Gauss-Seidel iteration which requires that the latest values of the relaxation variables be used to solve each equation in the inner loop. Each equation in the inner loop is a single differential equation that can be solved using any numerical integration method.

The standard WR Gauss Seidel algorithm can be enhanced by rearranging the order of equations in the inner loop according to the signal flow. It is, in fact, an event-driven analysis where only active variables need to be updated. This is referred to as the selective-trace as mentioned earlier.

Similarly, the Gauss-Seidel technique can be superimposed on the main simulation procedure shown in Fig.4.3 to arrange a relaxation loop for the mixed PWL/logic approach. As a consequence, the local event-driven simulation process has to be repeated until the related waveforms converge to some limit. In this case the waveforms would consist of a single segment or of a few PWL segments due to the *windowing* approach used, which is a technique where the analysis time is divided into intervals [DEB87, WHI87]. In this way the convergence can be enhanced, since the accumulation of waveform errors is limited in time.

An algorithm for WR-based mixed PWL/logic simulation is shown in Fig. 5.2. The windowing technique is applied. Within each window the WR Gauss-Seidel iterations are organised as the while-loop. As compared to the algorithm of Fig.4.3, here, each analysis (iteration) provides PWL/logic waveforms matching the actual window rather than a single PWL segment. Once the analysis of a block for the actual window is completed, the next block is popped from queue for that window. The analysed blocks need extra scheduling due to the outer simulation loop unless convergence of the iterated waveforms is reached. Besides,

active blocks should be scheduled for the next window before the analysis for the actual window is completed.

```
initialise_simulation
    { compile_netlist; compile_input_stimulus;
      arrange_time_queue; arrange_remote_list
    }
repeat
    while (queue_not_empty)          (* outer simulation loop *)
      { reset_time_for_window(k);
        repeat                       (* inner simulation loop *)
          enter_time_queue (t_k);
          .          (* Process a block *)
          .          (* within actual time window *)
          .
        until end_of_window(k);
        check_for_convergence;
        if (no_convergence) schedule block
      }
    inc(k)
until last_window_processed
```

Fig. 5.2. WR Gauss-Seidel algorithm for mixed-mode PWL/ logic simulator

When processing within the outer simulation while-loop, only selected waveforms need checking for convergence, and usually checking for one waveform per network loop is sufficient (other waveforms follow the loop constraints and converge as well). Following this also a single block per network loop needs to be scheduled at the beginning of the actual window (when not converged), whereas the other loop elements are scheduled in the inner simulation repeat-loop, once that block is processed.

In order to discuss the convergence of the WR-based PWL algorithm, consider again the system of basic building blocks described by equations (5.1). Using the PWL approximation-based approach each block is provided, in some sense, with the PWL approximator as shown in Fig. 2.3. Hence, denoting by $L(x_k)$ the PWL approximation of x_k (PWL output of k-th block), eqn.(5.1) can be rewritten in form

$$\dot{x}_k = f_k[L(x_1), L(x_2), \ldots L(x_n), u], \qquad k = 1,2, \ldots n \tag{5.2}$$

Observe that this notation only makes sense when the operators $L(\cdot)$ are provided with explicit arguments, so solving for (5.2) should be performed in terms of relaxation. To cope with this problem the algorithm of Fig.5.1 can be adopted, referred to as the *WR Gauss-Seidel PWL algorithm* (approximation based). Basically, the WR algorithms converge under mild conditions [DEB87, WHI87]. Based on it, a formulation of a similar convergence theorem for the relevant WR-PWL algorithm is feasible.

For this purpose denote the vector $[x_1, x_2, \ldots x_n]^T$ by x, and introduce an exponentially weighted norm on $C(\langle t_0, t_{stop} \rangle, \mathbb{R}^n)$

$$\| x \|_\lambda = \underset{t \in \langle t_0, t_{stop} \rangle}{Max} e^{-\lambda t} \| x(t) \|, \qquad \lambda > 0 \tag{5.3}$$

Theorem 5.1 (1st convergence theorem): For a system defined by eqn.(5.2) the WR Gauss-Seidel PWL approximation-based algorithm converges uniformly in the norm (5.3) with some $\lambda > \lambda_0$, if the functions $\{f_k\}$ are continuous and Lipschitz with respect to their arguments on $t \in \langle t_0, t_{stop} \rangle$.

Proof: Applying the Gauss-Seidel scheme to eqn.(5.2) we obtain

$$\dot{x}_k^{j+1} = f_k[L(x_1^{j+1}), L(x_2^{j+1}), \ldots L(x_k^{j+1}), L(x_{k+1}^{j}), \ldots L(x_n^{j}), u], \quad k = 1,2, \ldots n \tag{5.4}$$

where j is the actual iteration number.

For brevity denote the time derivative of x by z, and introduce an integral operator $I(z) = x$. Now, the iterative eqn.(5.4) may rearranged to a general form

$$z^{j+1} = \varphi[LI(z^{j+1}), LI(z^j), u] \tag{5.5}$$

In order to show that (5.5) is a contraction, observe that since the functions $\{f_k\}$ are Lipschitz, ν is Lipschitz as well. Hence, for the norm (5.3)

$$\| z^{j+1} - z^{k+1} \|_\lambda \le K_1 \| LI(z^{j+1}) - LI(z^{k+1}) \|_\lambda + K_2 \| LI(z^j) - LI(z^k) \|_\lambda \tag{5.6}$$

where K_1, K_2 are positive constants and j, k stand for iteration indexes.

The operator $I(\cdot)$ may be shown to be Lipschitz too [WHI87, ch.4]:

$$\begin{aligned} \| I(z^j) - I(z^k) \|_\lambda &= \underset{t \in \langle t_0, t_{stop} \rangle}{Max} e^{-\lambda t} \left\| \int_{t_0}^{t} [z^j(\tau) - z^k(\tau)]\, d\tau \right\| \\ &\le \underset{t \in \langle t_0, t_{stop} \rangle}{Max} e^{-\lambda t} \int_{t_0}^{t} e^{\lambda \tau} \left[e^{-\lambda \tau} \| z^j(\tau) - z^k(\tau) \| \right] d\tau \\ &\le \frac{1 - e^{\lambda(t_0 - t)}}{\lambda} \cdot \| z^j - z^k \|_\lambda \le \frac{1}{\lambda} \cdot \| z^j - z^k \|_\lambda \end{aligned} \tag{5.7}$$

The same holds for $\{x_k\}$, since $\{\dot{x}_k\} = \{f_k\}$ are continuous. Consequently, as $x_k = L(x_k)$ for the PWL breakpoints (see (2.8)), the operator $L(\cdot)$ satisfies also the Lipschitz condition with some positive constant M, i.e.:

$$\| L(x^j) - L(x^k) \|_\lambda \le M \| x^j - x^k \|_\lambda \tag{5.8}$$

Next, substituting (5.7) and (5.8) (both for j, k and $(j+1)$, $(k+1)$) to (5.6) one obtains

$$\begin{aligned} \| z^{j+1} - z^{k+1} \|_\lambda &\le K_1 M \| I(z^{j+1}) - I(z^{k+1}) \|_\lambda + K_2 M \| I(z^j) - I(z^k) \|_\lambda \\ &\le K_1 M \frac{1}{\lambda} \cdot \| z^{j+1} - z^{k+1} \|_\lambda + K_2 M \frac{1}{\lambda} \cdot \| z^j - z^k \|_\lambda \end{aligned}$$

and after rearranging

$$\| z^{j+1} - z^{k+1} \|_\lambda \le \frac{K_2 M}{\lambda - K_1 M} \cdot \| z^j - z^k \|_\lambda \tag{5.9}$$

For sufficiently large λ ($\lambda_0 = (K_1 + K_2) \cdot M$) the global constant in (5.9) is less than 1 so that (5.5) is a contraction for the weighted norm $\| \cdot \|_\lambda$, and $\{z^j\}$ converges uniformly to a unique

fixed point by virtue of the contraction mapping theorem [WHI87]. Obviously, since for each iteration $x^j(t_0) = x(t_0)$, the $\{x^j\}$ sequence converges as well.□

A similar discussion can be performed also for the Gauss-Jacobi scheme, particularly well suited to parallel computing, which is not addressed here.

To summarise, the WR Gauss-Seidel PWL algorithm is guaranteed to converge under mild conditions imposed on the functions $\{f_k\}$ in Theorem 5.1. However, only equation-based models defined by (5.1-5.2) have been considered. Besides, the relation $\lambda > \lambda_0$ is of little practical use, since the constants K_1, K_2 and M are hardly available. Hence, this relation may be viewed as an implicit condition, which only reflects some other convergence condition of practical use that would be of interest. A detailed discussion regarding some more stringent convergence condition as well as the application of Theorem 5.1 to mixed-mode PWL/logic modelling will be presented in the following two sections.

5.2 Equation-based against behavioural models

Apparently, for mixed A/D networks represented at the functional level we deal with mixed models, which do not obey the system of homogenous equations, such as (5.2). However, the equation-based model may be shown to be equivalent to a mixed model, i.e. PWL-equation-based for the analogue part and behavioural-logic for the digital part of a system.

For example, consider a gate model shown in Fig.3.11. The two-argument maximum value function $Max(\cdot, \cdot)$ followed by the PWL DC-transfer function $f(\cdot)$ driving a unity gain linear inertial block is a well posed PWL macromodel of an OR or NOR logic gate, which can be described by a single equation

$$T\dot{x}_3 + x_3 = f\left[Max\left(L(x_1), L(x_2)\right)\right] \qquad (5.10)$$

and $L(x_3)$ provides the PWL output. Such a model can mimic the basic logic and timing behaviour of a gate; in particular, the delay time and the inertial effects caused by spikes or glitches at the input. More accurate macromodel, in which the delay time and output slope time may be distinguished, requires one inertial block more (with saturation). In this case the first one is responsible mainly for the delay time, whereas the next one in the cascade, for the slope. The cascade structure needs careful parameter matching as mentioned in Section 3.1 regarding the voltage comparator.

On the other hand, one can produce the correct time response of the one- or two-equation-based gate model (mentioned above) that is a subsystem of (5.2) by means of an equivalent behavioural model with PWL inputs, such as defined by eqns.(3.7) through (3.10). Clearly, all situations pertaining to the input/output signals that are involved with the equation-based model must be foreseen and built into the behavioural model to assure its consistency.

Similarly, since a complex digital unit consists of simple gates, the respective subsystem of (5.2) is able to represent its model as well. As a consequence, rather than to solve for that subsystem, equation by equation, one prefers an equivalent behavioural model (of a unit) with the same time responses as the origin (or very close to it).

By virtue of the above discussion and the results of Section 5.1, we conclude that the waveform relaxation algorithm applied to structures with different models (PWL- for analogue parts and behavioural models for their digital counterparts) also converges with respect to Theorem 5.1, if the behavioural models and the original ones are equivalent.

Observe that the behavioural models are viewed in this context only as a tool to solve for the subsystems of the differential eqns.-based models. Clearly, in practice, the adequate behavioural models (at the functional level) are developed directly from the specifications, and usually there is no need to account for the original block structure described as the subsystems of (5.2).

5.3 Waveform relaxation and one-segment relaxation for analogue sub-systems

In order to address the distinctive features of the WR-based PWL algorithm, first consider a second order low-pass filter shown in Fig.5.3. As it is an analogue feedback structure, its simulation should be organised as an iterative process, for which the algorithm depicted in Fig.5.2 can be used. Observe that for $Q > 0.5$ this structure cannot be replaced with two inertial blocks in cascade.

The WR Gauss-Seidel PWL algorithm (approximation-based) starts with the actual linear segment, given by the input stimulus. When propagating through the both blocks, it is usually partitioned into smaller pieces. The resulting segment lengths are at most equal to the input ones. After a few iterations the length of the actual segment (time-step size) stabilises in a loop at some minimum. In fact, for each block its input segment length and the corresponding output segment length are the same. Usually, the PWL segments approach their final lengths during the early iterations, when large WR errors propagate through the loop. The obtained segments are relatively short, so that the final PWL accuracy of the waveforms is usually better than the assumed one.

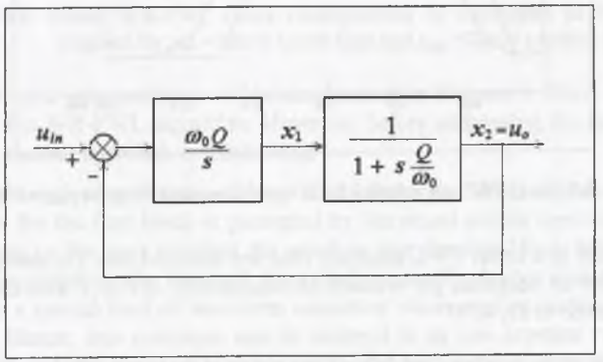

Fig. 5.3. Model of bi-quad low pass filter

Some of the simulation results for the Bessel filter are given in Figs.5.4 and 5.5. The filter specification is as follows: $Q = 0.577, f_n = 1.274, f_c = 1\text{kHz}, p_{mx} = 50\text{mV}, \omega_0 = 2\pi f_n f_c = 8004$ rd/s. To enhance the convergence speed of the WR, the windowing technique is used. The windows: [0, 0.2ms], [0.2ms, 0.7ms], [0.7ms, 1.7ms], [1.7ms, 2.2ms] and [2.2ms, 3ms] are generated by the respective input events. The first window is not of interest since the network is latent in it. For the WR convergence accuracy of 10mV and PWL accuracy $p_{mx} = 50\text{mV}$, the obtained number of iterations for the second- through the fifth window is respectively: 7, 13, 7, 11. Some of the iterative waveforms are depicted in Fig. 5.4, where the numbers shown correspond to the subsequent iterations. Observe also that in some windows (Fig.5.5) a relatively large number of the PWL segments occur (i.e., 9 in the third window and 6 in the

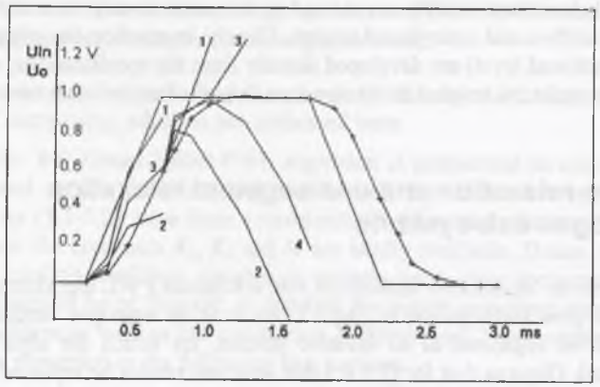

Fig. 5.4. Some of PWL iterative waveforms for bi-quad low-pass filter, p_{mx} = 50mV

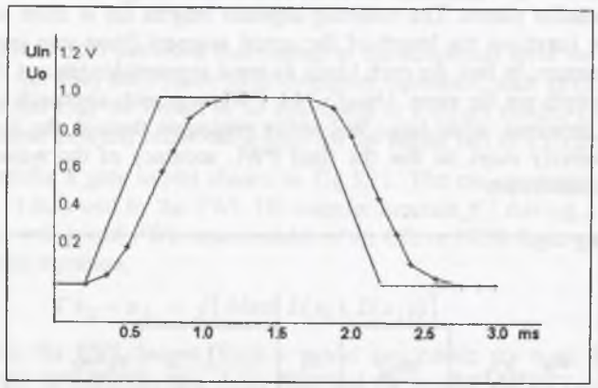

Fig. 5.5. Final PWL waveforms for bi-quad low-pass filter for p_{mx} = 50mV

last one) that result in a better PWL accuracy than the assumed one. For comparison, if p_{mx} = 20mV, the number of iterations per window is respectively: 6,11,6,9 with the corresponding number of segments: 4, 15, 4, 9.

On the other hand, if the windows are halved, the iteration count per window for p_{mx} = 50mV is as follows: 6+6, 8+100, 6+6, 7+6. Apparently, with the number of 100 iterations for the window [1.2ms,1.7ms] the process becomes almost unstable. In this case the algorithm attempts to yield a large segment length (since $| x_0 - u_0 + rT |$ is very small), which then is reduced to match the window size. Indeed, for very small values of the expression $| x_0 - u_0 + rT |$ for the inertial block, or $| r/T |$ for the integrator, the PWL algorithm yields relatively large time steps (due to the approximation formulas (2.17) and (2.36)), which are likely to make the relaxation process unstable.

Moreover, it should be observed that even well posed iterations give rise to accumulation of the PWL approximation errors. In fact, any smooth output waveform is either under- or over-estimated by the PWL approximator for a given time segment. Thus, multiple propagation of a waveform in a loop causes the effect of error accumulation (global errors arise). When the

PWL breakpoints are taken into account, those global errors are usually different from zero, unlike the local PWL errors (defined in Sec.2.1).

For the low-pass filter presented above, the resulting global errors are depicted in Fig.5.6. The error function is defined to be a difference between the PWL output u_{olin} (when brought to convergence) and the original response u_o (obtained without PWL approximation). Fortunately, those errors may be kept low using p_{mx} of reduced value, as shown, and they are usually less than $p_{mx}/2$ at the breakpoints, and less than p_{mx} otherwise. The results obtained with the TR-based technique are very close to this.

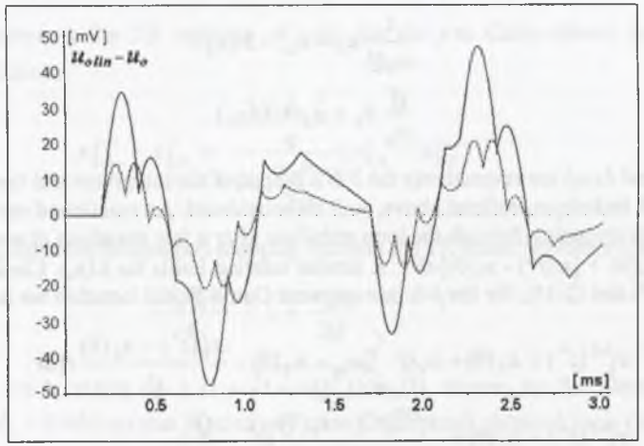

Fig. 5.6. Global WR-PWL errors corresponding to waveforms of Fig.5.5 obtained for p_{mx} =50mV (solid line) and p_{mx} =20mV (dashed line)

The problem of error accumulation will be emphasised in Chapter 6. Here, we focus on the convergence of the WR-PWL algorithm. However, before addressing the latter problem, we would consider another approach to windowing.

It is the *dynamic windowing* where a window size follows the actual segment length. That is, the first window for the first block is generated by the actual events involved with the input stimulus, whereas in the next iteration the window size for that block follows the iterated PWL segment provided by the feedback loop (and is usually shorter than the former one).. Apparently, it is a special kind of waveform relaxation where each waveform consists of one PWL segment. Hence, this technique can be referred to as *one-segment relaxation* (*OSR*). This approach is, in some sense, equivalent to the *one-step relaxation* mentioned already in Section 1.2. Applying OSR to the filter of Fig.5.3 results in reducing the number of iterations (unless instability is faced). The number of iterations per one-segment window for p_{mx} = 50mV is respectively: 4,4,3,7,6,7,7,6, ... resulting in the average as much as 5.7 (iterations per window). However, more CPU time might be required to perform the whole simulation as compared to the case of windows, each comprising a few PWL segments.

For example, using the windows: [0, 0.2ms], [0.2ms, 0.7ms], [0.7ms, 1ms], [1ms, 1.4ms], [1.4ms, 1.7ms], [1.7ms, 2.2ms], [2.2ms, 2.6ms] [2.6ms, 3ms] with the average number of segments per window equal 3.2, results in the average number of iterations per window as much as 7.5. The total CPU time for this windowing is by 20% less than that of OSR technique. Despite this, OSR might be viewed as a uniform approach to iterative PWL simulation, which is

particularly useful in implementations that are not well suited to the WR, such as VHDL (see Chpt.7).

To summarise, a balance exists between the potential gain due to the windowing approach and the additional computations and storage required in managing the windowing solution. This problem has been investigated e.g. in [DAB91E]. However, no technique has been reported so far to provide optimal windowing in general case.

To discuss the convergence of the relaxation iterations, express formally the PWL approximation procedure by the operator $L(\cdot)$, so that the filter structure may be described by

$$\begin{aligned} \frac{1}{\omega_0 Q}\dot{x}_1 &= u_{in} - L(x_2) \\ \frac{Q}{\omega_0}\dot{x}_2 + x_2 &= L(x_1) \end{aligned} \tag{5.11}$$

where $L(x_1)$ and $L(x_2)$ are respectively the PWL output of the integrator and the inertial block. First, the OSR technique, defined above, will be considered. As mentioned earlier, the length of a segment propagating through the loop stabilises after a few iterations at some value t^*, so that $L(x_1) = x_1(0) + [x_1(t^*) - x_1(0)]\cdot t/t^*$. A similar relation holds for $L(x_2)$. Consequently, with respect to (2.7) and (2.35), for the j-th one-segment Gauss-Seidel iteration we have

$$\begin{aligned} x_1^{j+1}(t^*) &= x_1(0) + \omega_0 Q \cdot \int_0^{t^*} [u_{in} - x_2(0) - \frac{x_2^j(t^*) - x_2(0)}{t^*} t]\,dt \\ x_2^{j+1}(t^*) &= x_1(0) + \frac{x_1^{j+1}(t^*) - x_1(0)}{t^*}(t^* - \frac{Q}{\omega_0}) \\ &\quad + [x_2(0) - x_1(0) + \frac{x_1^{j+1}(t^*) - x_1(0)}{t^*}\frac{Q}{\omega_0}]\, e^{-\frac{t^*\omega_0}{Q}} \end{aligned} \tag{5.12}$$

and after simple manipulations

$$x_2^{j+1}(t^*) - x_2^j(t^*) = -[\frac{\omega_0 Q}{2}t^* - \frac{Q^2}{2}(1 - e^{-\frac{t^*\omega_0}{Q}})]\cdot[x_2^j(t^*) - x_2^{j-1}(t^*)] \tag{5.13}$$

Now, by virtue of the contraction-mapping theorem the Gauss-Seidel iterative process performed on a single segment converges when

$$\left| \frac{\omega_0 Q}{2}t^* - \frac{Q^2}{2}(1 - e^{-\frac{t^*\omega_0}{Q}}) \right| < 1 \tag{5.14}$$

This inequality (solved numerically) holds for $t^* < 0.505$ ms, which is the maximum allowed segment length that assures the WR-PWL approximation-based algorithm to be convergent.

Similarly, we would consider the application of PWL TR-based technique. Clearly, the $L(\cdot)$ operator in (5.11) should be replaced with the trapezoidal rule:

$$x_k = \frac{\dot{x}_k + \dot{x}_{k-1}}{2} h_k + x_{k-1}$$

where x_k is an estimate of $x(t_k)$, and h_k stands for the actual time step, common for both blocks (as already explained). Hence, one obtains

$$\begin{aligned} x_{1,k}^{j+1} &= x_{1,k-1} + \frac{\omega_0 Q h_k}{2}[u_{in}(t_k) + u_{in}(t_{k-1}) - x_{2,k}^j - x_{2,k-1}] \\ x_{2,k}^{j+1} &= \frac{(1 - \frac{\omega_0 h_k}{2Q})x_{2,k-1} + \frac{\omega_0 h_k}{2Q}(x_{1,k}^{j+1} + x_{1,k-1})}{1 + \frac{\omega_0 h_k}{2Q}} \end{aligned} \tag{5.15}$$

where $x_{i,k}^{j+1}$ represents the TR estimate of $x_i(t_k)$ for the j-th Gauss-Siedel iteration. After simple manipulations

$$x_{2,k}^{j+1} - x_{2,k}^j = \frac{-\frac{(\omega_0 h_k)^2}{4}}{1 + \frac{\omega_0 h_k}{2Q}}(x_{2,k}^j - x_{2,k}^{j-1}) \tag{5.16}$$

Next, following again the contraction-mapping theorem the TR-based iterations converge when

$$\frac{(\omega_0 h_k)^2}{4} < 1 + \frac{\omega_0 h_k}{2Q} \tag{5.17}$$

Solving (5.17) for h_k yields $h_k < (1 + \sqrt{1 + 4Q^2})/(\omega_0 Q)$. Hence, for the filter specification used, we have $h_k < 0.466$ ms that is relatively close to the result obtained from (5.14).

To put this discussion into a broader perspective consider an n-th order system composed of the basic building blocks, i.e. inertial blocks (labelled with index k) and integrators (index s):

$$\begin{aligned} T_k\dot{x}_k + x_k &= f_k[L(x_1), L(x_2), \ldots L(x_n), u], \quad k \in \{1, ..n\} \\ T_s\dot{x}_s &= f_s[L(x_1), L(x_2), \ldots L(x_n), u], \quad s \in \{1, ..n\} \end{aligned} \tag{5.18}$$

where $f_k, f_s : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ for $t \in \langle t_0, t_{stop} \rangle$ are PWL functions, and $u(t) \in \mathbb{R}^m$ denotes the vector of external PWL input signals. Apparently, the filter from Fig.5.2 is a special case of (5.18). Next, assume that at some driving point the functions f_k and f_s take the form of

$$f_m = f_{mo} + \sum_{i=1,\ i \neq m}^{n} \alpha_{mi} x_i + b_m^T u, \qquad m = 1, \ldots n \tag{5.19}$$

In Section 5.1 the general convergence conditions for the WR-PWL algorithm have been given. Here, a more stringent convergence conditions for a system described by eqns.(5.18) and (5.19) with respect to the WR PWL will be presented.

First, observe that with the definition (5.19) we assume the input of the m-th block to depend only indirectly on its x_m output. As a consequence, the resulting matrix $\mathbf{A}=[\alpha_{mi}]$ has zeros on its diagonal. Further, assume $\mathbf{A}$ to be irreducible [VAR63]. Typically, it means that the analogue system (5.18) does not fall into two (or more) sub-systems in cascade. For illustration, see Fig.5.7, which is an example of a reducible system.

In other words, there exists some global feedback loop comprising all the sub-blocks in the irreducible system, and hence, at some driving point the iterations tend to approach some limit segment length t^*, which is same for all the involved building blocks.

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & g & 0 \\ e & 0 & f & d \\ h & 0 & 0 & 0 \\ b & a & c & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & a & b & c \\ d & 0 & e & f \\ 0 & 0 & 0 & g \\ 0 & 0 & h & 0 \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ 0 & \mathbf{A}_{22} \end{bmatrix}$$

Fig. 5.7. After reordering of variables: $x_1 \leftrightarrow x_4$ and next $x_3 \leftrightarrow x_1$, **A** becomes upper triangular block matrix corresponding to cascade block structure (so **A** is reducible); however, the matrices $\mathbf{A}_{11}$, $\mathbf{A}_{12}$, $\mathbf{A}_{22}$ themselves represent irreducible subsystems

Based on it the following theorem may be formulated:

Theorem 5.2 (2nd convergence theorem): For any irreducible system composed of the basic building blocks, defined by eqns.(5.18-5.19), the OSR Gauss-Seidel PWL algorithm (approximation based) is convergent for any initial guess $\{x_m^0\}$ if

$$1 \geq g_m \sum_{i=1, i\neq m}^{n} |\alpha_{mi}|, \qquad m = 1, \ldots n \tag{5.20}$$

where for at least one m the inequality must be strict, and

$$g_m = 1 - \frac{T_m}{t^*}\left(1 - e^{\frac{-t^*}{T_m}}\right) \qquad \text{for inertial blocks,} \tag{5.21a}$$

$$g_m = \frac{t^*}{2T_m} \qquad \text{for integrators,} \tag{5.21b}$$

where t^* denotes the PWL segment length, common for all blocks in the system.

Proof: For the PWL operator one obtains

$$L(x_i) = x_{i0} + \frac{\partial L(x_i)}{\partial t} t = x_{i0} + \frac{x_i(t_{i1}) - x_{i0}}{t_{i1}} t, \tag{5.22}$$

where t_{i1} is the segment length computed for i-th block. Next, invoke eqns.(2.7) and (2.35), so that the PWL solution for (5.18), for the current output segment, may be represented as

$$\begin{aligned} x_k(t) &= (x_{k0} - u_{k0} + r_k T_k) e^{-t/T_k} + r_k (t - T_k) + u_{k0}, \qquad k \in \{1, \ldots n\} \\ x_s(t) &= x_{s0} + \frac{u_{s0} t}{T_s} + \frac{r_s t^2}{2T_s}, \qquad s \in \{1, \ldots n\} \end{aligned} \tag{5.23}$$

where for each block

$$u_{m0} = f_{m0} + \sum_{i=1, i\neq m}^{n} \alpha_{mi} x_{i0}, \qquad r_m = \sum_{i=1, i\neq m}^{n} \alpha_{mi} \frac{\partial L(x_i)}{\partial t}, \qquad m = 1, \ldots n$$

and without loss of generality we assume $u = 0$. Using the Gauss-Seidel relaxation scheme, (5.23) may be rewritten as

$$\begin{aligned} x_k^{j+1}(t) &= (x_{k0} - u_{k0}) e^{-t/T_k} \\ &\quad + \left(T_k e^{-t/T_k} + t - T_k\right) \cdot \left(\sum_{i=1}^{k-1} \alpha_{ki} \frac{\partial L(x_i^{j+1})}{\partial t} + \sum_{i=k+1}^{n} \alpha_{ki} \frac{\partial L(x_i^{j})}{\partial t} \right) + u_{k0}, \\ x_s^{j+1}(t) &= x_{s0} + \frac{u_{s0} t}{T_s} + \frac{t^2}{2T_s} \left(\sum_{i=1}^{s-1} \alpha_{si} \frac{\partial L(x_i^{j+1})}{\partial t} + \sum_{i=s+1}^{n} \alpha_{si} \frac{\partial L(x_i^{j})}{\partial t} \right), \quad k, s \in \{1, \ldots n\} \end{aligned} \tag{5.24}$$

Since the system is irreducible, during iterations the segment lengths t_{i1} approach some limit t^* (common for all blocks), for which the iterations are expected to converge. Hence, putting (5.22) into (5.24) for $t = t^*$ it follows

$$\begin{aligned} x_k^{j+1}(t^*) &= (x_{k0} - u_{k0}) e^{\frac{-t^*}{T_k}} \\ &\quad + \left(T_k e^{\frac{-t^*}{T_k}} + t^* - T_k \right) \cdot \left(\sum_{i=1}^{k-1} \alpha_{ki} \frac{x_i^{j+1}(t^*) - x_{i0}}{t^*} + \sum_{i=k+1}^{n} \alpha_{ki} \frac{x_i^{j}(t^*) - x_{i0}}{t^*} \right) + u_{k0}, \\ x_s^{j+1}(t^*) &= x_{s0} + \frac{u_{s0} t^*}{T_s} + \frac{t^{*2}}{2T_s} \left(\sum_{i=1}^{s-1} \alpha_{si} \frac{x_i^{j+1}(t^*) - x_{i0}}{t^*} + \sum_{i=s+1}^{n} \alpha_{si} \frac{x_i^{j}(t^*) - x_{i0}}{t^*} \right), \\ &\qquad k, s \in \{1, \ldots n\} \end{aligned} \tag{5.25}$$

and after simplifications

$$\begin{aligned} x_k^{j+1}(t^*) &= \left(1 - \frac{T_k}{t^*}\left(1 - e^{\frac{-t^*}{T_k}}\right)\right) \cdot \left(\sum_{i=1}^{k-1} \alpha_{ki} x_i^{j+1}(t^*) + \sum_{i=k+1}^{n} \alpha_{ki} x_i^{j}(t^*) \right) + \beta_k, \\ x_s^{j+1}(t^*) &= \frac{t^*}{2T_s} \left(\sum_{i=1}^{s-1} \alpha_{si} x_i^{j+1}(t^*) + \sum_{i=s+1}^{n} \alpha_{si} x_i^{j}(t^*) \right) + \gamma_k, \qquad k, s \in \{1, \ldots n\} \end{aligned} \tag{5.26}$$

Now we represent the eqns. (5.26) in a matrix form

$$x^{j+1} = \mathbf{L} x^{j+1} + \mathbf{U} x^{j} + d \tag{5.27}$$

where **L** and **U** are respectively the strictly lower- and the upper triangular matrices

$$\mathbf{L} = Diag[g_1, g_2, \ldots g_n] \begin{bmatrix} 0 & \ldots & 0 & 0 \\ \alpha_{21} & 0 & & 0 \\ \ldots & \ldots & & \\ \alpha_{n1} & \alpha_{n2} & \ldots & 0 \end{bmatrix}$$

$$\mathbf{U} = Diag[g_1,\ g_2,\ \dots\ g_n]\begin{bmatrix} 0 & \alpha_{12} & \dots & \alpha_{1n} \\ \dots & 0 & & \alpha_{2n} \\ 0 & \dots & & \dots \\ 0 & 0 & \dots & 0 \end{bmatrix}$$

where $g_m = 1 - \frac{T_m}{t^*}(1 - e^{\frac{-t^*}{T_m}})$ for inertial blocks, and $g_m = \frac{t^*}{2T_m}$ for integrators. Hence, it follows

$$x^{j+1} = (1 - \mathbf{L})^{-1}\mathbf{U}\,x^j + (1 - \mathbf{L})^{-1}d \qquad (5.28)$$

and by virtue the basic matrix theory [VAR63, Chpt.3] the iterations (5.28) converge for any initial approximation x^0, if the matrix (**1–L–U**) is strictly diagonally dominant or is irreducibly diagonally dominant. Since A is irreducible and

$$Diag[g_1,\ g_2,\ \dots\ g_n]\cdot \mathbf{A} = \mathbf{L} + \mathbf{U}, \qquad (5.29)$$

(**1–L–U**) is irreducible as well, so it should be irreducibly diagonally dominant, i.e.

$$1 \ge g_m \sum_{i=1, i\ne m}^{n} |\alpha_{mi}|, \qquad m = 1, \dots n$$

where for at least one m the inequality must be strict.□

It may be observed that the convergence condition (5.20) holds for sufficiently small t^*, since g_m defined by (5.21*a,b*) are ascending functions with respect to t^*.

Applying the obtained result to the low-pass filter described by eqns. (5.11), it follows

$$1 \ge \omega_0 Q t^* / 2 \qquad (5.30a)$$

$$1 > 1 - \frac{Q}{\omega_0 t^*}(1 - e^{\frac{-\omega_0 t^*}{Q}}) \qquad (5.30b)$$

Since (5.27*b*) holds for any $t^* > 0$, the condition (5.30*a*) is crucial. However, it appears to be stronger than (5.14). For the considered Bessel filter, from (5.30*a*) obtains $t^* \le 2/(\omega_0 Q) = 0.433$ms (the former result was $t^* < 0.505$ms). In fact, the derived conditions correspond to each other in a sense that the product of (5.30*a*) and (5.30*b*) yields perfectly (5.14).

In practice, when nonlinear PWL functions are used in models, a multiple checking for the condition (5.20) may be cumbersome. So, it seems reasonable to reduce the resulting segment length without checking, if too many iterations occur.

When using the enhanced TR algorithm as an alternative to the PWL approximation-based technique, Theorem 5.2 can be adopted to provide a convergence condition as well. Replacing the operator $L(\cdot)$ in (5.18) and (5.19) by the trapezoidal rule, yields

$$x_{k,i}^{j+1} = \frac{(1 - \frac{h_i}{2T_k})x_{k,i-1} + \frac{h_i}{2T_k}\left[2u_{k0} + b_k^T[u(t_i) + u(t_{i-1})]\right]}{1 + \frac{h_i}{2T_k}} + \frac{\frac{h_i}{2T_k}\left[\sum_{l=1}^{k-1}\alpha_{kl}x_{l,i}^{j+1} + \sum_{l=k+1}^{n}\alpha_{kl}x_{l,i}^{j} + \sum_{l=1,l\ne k}^{n}\alpha_{kl}x_{l,i-1}\right]}{1 + \frac{h_i}{2T_k}}, \qquad (5.31)$$

$$x_{s,i}^{j+1} = \frac{h_i}{2T_s}\left[2u_{k0} + b_k^T[u(t_i) + u(t_{i-1})] + \sum_{l=1}^{s-1}\alpha_{sl}x_{l,i}^{j+1} + \sum_{l=s+1}^{n}\alpha_{sl}x_{l,i}^{j} + \sum_{l=1,l\ne s}^{n}\alpha_{sl}x_{l,i-1}\right] \qquad k, s \in \{1, .. n\}$$

where $x_{m,i}^{j+1}$ represents the TR estimate of $x_m(t_i)$ for the j-th Gauss-Siedel iteration. After simplifications (5.31) reduces to

$$x_{k,i}^{j+1} = \frac{\frac{h_i}{2T_k}}{1 + \frac{h_i}{2T_k}}\left[\sum_{l=1}^{k-1}\alpha_{kl}x_{l,i}^{j+1} + \sum_{l=k+1}^{n}\alpha_{kl}x_{l,i}^{j}\right] + \beta_k \qquad (5.32)$$

$$x_{s,i}^{j+1} = \frac{h_i}{2T_s}\left[\sum_{l=1}^{s-1}\alpha_{sl}x_{l,i}^{j+1} + \sum_{l=s+1}^{n}\alpha_{sl}x_{l,i}^{j}\right] + \gamma_s, \qquad k, s \in \{1, .. n\}$$

From (5.32) one can pick up the relevant coefficients $g_m = \frac{h_i}{2T_m}\Big/\left(1 + \frac{h_i}{2T_m}\right)$ for inertial blocks, and $g_m = \frac{h_i}{2T_m}$ for integrators. Apparently, when using those g_m coefficients, the convergence condition (5.20) applies to the PWL OSR TR-based technique as well (compare (5.32) to (5.26)). Moreover, observe that the convergence condition for the integrating blocks does not depend on the PWL technique used (i.e. approximation-based or TR-based).

For the low-pass filter considered before, the convergence condition is identical to (5.30*a*), i.e. $1 \ge \omega_0 Q h_i / 2$, whereas the other condition is trivial as it holds for any positive step h_i, i.e. $1 > \frac{\omega_0 h_i}{2Q}\Big/\left(1 + \frac{\omega_0 h_i}{2Q}\right)$. Observe also that a product of those two inequalities yields perfectly (5.17).

Next, we would consider stability of the PWL OSR algorithm. For this purpose assume a stable linear system composed of the basic building blocks, defined in a unified form

$$Diag[T_1, T_2, \dots T_n]\cdot \dot{x}(t) = \mathbf{A}x(t), \qquad x(0) = x_0 \qquad (5.33)$$

where $\mathbf{A} \in \mathbb{R}^{n\times n}$, $x(t) \in \mathbb{R}^n$, all $T_i > 0$, and $\lim_{t\to\infty} x(t) = 0$. Applying the PWL OSR algorithm to solve for (5.33) results in an integration scheme of general form

$$x_{k+1} = \mathbf{H}\, x_k\,, \qquad x_0 = x(0) \tag{5.34}$$

where x_k is an estimate of the exact solution $x(t_k)$. We would prove that $\lim_{k\to\infty} x_k = 0$, too. Although (5.33) covers both inertial and integrating blocks, for simplicity of the proof all the blocks will be considered as integrators (the proof for general case appears difficult). Clearly, any integrator provided with a local feedback (when $\alpha_{ii} = -1$) represents an inertial block. Consequently, the matrix **A** is no more assumed to have zeros on its diagonal. It is to be pointed out that in this case Theorem 5.2 holds as well, provided the condition (5.20) is modified to the form

$$\left|1 - g_m\alpha_{mm}\right| \geq g_m \sum_{i=1, i\neq m}^{n} \left|\alpha_{mi}\right|, \qquad m = 1, \ldots n \tag{5.35}$$

To see this, note that **L** in (5.27) is not strictly lower triangular in this case, but it contains some non-zero elements ($g_m\alpha_{mm}$) on its diagonal.

> **Theorem 5.3:** For any irreducible and asymptotically stable system defined by eqns.(5.33), the region of stability of the OSR Gauss-Seidel PWL algorithm is the same as for the convergence of the relevant relaxation iterations.

Proof: Both approximation- and TR-based technique will be addressed. First, applying the PWL operator to (5.33) we have

$$T_s \dot{x}_s(t) = \sum_{i=1}^{n} \alpha_{si} L\big(x_i(t)\big), \qquad s = 1..n \tag{5.36}$$

that may be viewed to represent a network of n PWL integrating blocks. Hence, based on the Gauss-Seidel scheme and using (5.22) for $L(\cdot)$ one obtains (like in (5.25))

$$\begin{aligned} x_{s,k+1}^{j+1} &= x_{s,k} + \frac{h_k}{T_s}\sum_{i=1}^{n}\alpha_{si}x_{i,k} \\ &+ \frac{h_k^2}{2T_s}\left[\sum_{i=1}^{s}\alpha_{si}\,\frac{x_{i,k+1}^{j+1} - x_{i,k}}{h_k} + \sum_{i=s+1}^{n}\alpha_{si}\,\frac{x_{i,k+1}^{j} - x_{i,k}}{h_k}\right], \quad s = 1..n \end{aligned} \tag{5.37}$$

where $x_{i,k+1}^{j+1}$ represents the PWL estimate of $x_i(t_{k+1})$ for the j-th iteration, and $h_k = t_{k+1} - t_k$ is a common time-step for all blocks. Observe that (5.37) also holds for the TR-based approach (compare (5.37) with (5.31)), so no extra proof is required in that case.

If the relaxation iterations are brought to convergence, following the condition (5.35), then for all i $x_{i,k+1}^{j+1} = x_{i,k+1}^{j} = x_{i,k+1}$, so that

$$x_{s,k+1} - \frac{h_k}{2T_s}\sum_{i=1}^{n}\alpha_{si}x_{i,k+1} = x_{s,k} + \frac{h_k}{2T_s}\sum_{i=1}^{n}\alpha_{si}x_{i,k}\,, \qquad s = 1..n \tag{5.38}$$

Next, putting (5.38) to the matrix form

$$\left(1 - \frac{h_k}{2}\,Diag\left[\frac{1}{T_1}, \ldots \frac{1}{T_n}\right]\cdot \mathbf{A}\right) x_{k+1} = \left(1 + \frac{h_k}{2}\,Diag\left[\frac{1}{T_1}, \ldots \frac{1}{T_n}\right]\cdot \mathbf{A}\right) x_k \tag{5.39}$$

Hence, the integration matrix due to (5.34) is

$$\mathbf{H} = \left(1 - \frac{h_k}{2}\,Diag\left[\frac{1}{T_1}, \ldots \frac{1}{T_n}\right]\cdot \mathbf{A}\right)^{-1} \cdot \left(1 + \frac{h_k}{2}\,Diag\left[\frac{1}{T_1}, \ldots \frac{1}{T_n}\right]\cdot \mathbf{A}\right) \tag{5.40}$$

To assure stability of the algorithm, all the eigenvalues λ_i of **H** should be placed inside the unity circle, i.e. $|\lambda_i| < 1$ for $i = 1, \ldots n$. Observe also that as (5.33) represents an asymptotically stable system, all the eigenvalues s_i of $Diag\left[T_1^{-1}, T_2^{-1} \ldots T_n^{-1}\right]\cdot \mathbf{A}$ should belong to the complex open left-half plane, i.e. $\mathrm{Re}(s_i) < 0$. Since the relation between the eigenvalues λ_i and s_i follows eqn. (5.40), one obtains

$$|\lambda_i| = \left|\left(1 - \frac{h_k}{2}s_i\right)^{-1} \cdot \left(1 + \frac{h_k}{2}s_i\right)\right| < 1, \qquad i = 1,2,\ldots n \tag{5.41}$$

and hence, $\left|\frac{h_k}{2}s_i + 1\right| < \left|\frac{h_k}{2}s_i - 1\right|$ that holds for any $h_k > 0$. Thus, the convergence condition for the relaxation iterations is sufficient for the OSR Gauss-Seidel PWL algorithm to be stable. □

Finally, we would address the relation between convergence of OSR and WR. For linear systems the both techniques correspond closely to each other. To discuss this, the following theorem can be cited due to [WHI87, Chpt.6].

> **Theorem 5.4:** Let a consistent and stable multistep integration algorithm be applied to a linear system of the form
>
> $$\mathbf{C}\,\dot{x}(t) = \mathbf{A}\,x(t), \qquad x(0) = x_0$$
>
> where $\mathbf{C},\mathbf{A} \in \mathbb{R}^{n\times n}$, **C** is nonsingular; and $x(t) \in \mathbb{R}^n$. Assume further that the Gauss-Seidel algebraic relaxation algorithm is used to solve the linear algebraic equations generated by the integration algorithm (e.g. trapezoidal rule). Given a sequence of timesteps $\{h_k\}$, the Gauss-Seidel algebraic relaxation algorithm will converge at every step, for any initial guess, if and only if the global-timestep discretised Gauss-Seidel WR algorithm, generated by solving iteration equations with the same multistep integration algorithm and the same timestep sequence, converges for any initial guess.

Consistency means here that the local truncation error (LTE) goes to zero as the time-step $h_k \to 0$. With respect to Theorem 5.4 the PWL OSR algorithm may be viewed as a kind of the algebraic relaxation, and the PWL WR as the corresponding global-time-step discretised WR algorithm (all Gauss-Seidel) satisfying the respective assumptions. As explained earlier, when simulating an irreducible system, the PWL WR/OSR algorithm tends to work with a common (i.e. global) time-step. Hence, by virtue of Theorem 5.4 we conclude that the convergence condition (5.20) (or (5.35)) holds not only for OSR, but also for the WR Gauss-Seidel PWL algorithm (approximation-based or TR-based) when applied to a linear system.

The above theorem also applies to nonlinear systems if it is assumed that an arbitrarily close initial guess for each of the relaxation schemes is available [WHI87, Chpt.6]. Although this is

not a realistic assumption, it does indicate that even for nonlinear systems the OSR and WR present very similar time-step constraints for an analysis method.

5.4 Simulation examples

In the previous section merely the feedback loop systems of analogue nature have been addressed and shown to be critical for the simulation process. On the other hand, as opposed to such loops, the loop structure including mixed A/D components (also referred to as switching or clocked) does not tend to be permanently closed. Instead, for a given portion of time only a part of that loop is active.

The reason may be two-folded: blocking the signal propagation by an external clock or zero gain of one of the loop elements. The latter case usually pertains to units that are modelled with an ideal switching DC transfer function. As a consequence, Theorem 5.2 is not applicable to mixed A/D structures in typical cases (which are not irreducible), and the segment lengths produced by the PWL algorithm during simulation are not critical for them.

Nevertheless, proper using of the windowing technique seems still to be essential for the mixed A/D loops. If it is possible to predict the boundaries of the subsequent periods of activity (assumed to be the time windows), the WR (or OSR) performed for those loops can converge even in a single iteration (Example 2). For clocked systems, fixed time windows that match the clock period are usually the best choice. This applies e.g., to the system presented in Section 4.3, when treated with the WR technique.

Example 1. In Fig.5.8 a model of the frequency division-mode telemetric receiver is shown. Fourth order digitally controlled band-pass filter is used. Each section of the filter is provided with an 8-bit DAC, which may be viewed here as a linear loop element (Fig.5.9). The control part of DAC follows the fundamental formula: $U_{DAC} = \sum a_i 2^i \Delta U$, where i=0..7 and ΔU represents the converter resolution proportional to reference V_r (see also Sec.4.3). The a_i parameters are set either to 0 or to 1 with respect to the control bits d_i after the prescribed delay time. In this way the multiplying DAC is defined as a mixed-signal model. Its output stage is assumed to be an inertial block with controlled gain equal d/d_{mx} and time constant T_{DAC}. Clearly, the digitally controlled 4-th order band-pass filter is of particular interest here for its feedback structure. The time constants T_{DAC} and T_3 may be viewed to represent some parasitic elements, usually not available directly at this level of abstraction. Thus, by putting $T_{DAC} = 0$ and $T_3 = 0$ a transfer function for the single-section filter takes the form of

$$H(s) = \frac{-\frac{1}{sT_1}}{1+\frac{1}{sT_1}\cdot\left(k+\frac{d}{d_{mx}}\cdot\frac{1}{sT_2}\right)} = \frac{-s\frac{1}{T_1}}{s^2+s\frac{k}{T_1}+\frac{d}{d_{mx}}\cdot\frac{1}{T_1T_2}} \qquad (5.42)$$

By comparison to the standard-form transfer function we obtain

$$\omega_0^2 = \frac{d}{d_{mx}}\cdot\frac{1}{T_1T_2}, \qquad Q^2 = \frac{d}{d_{mx}}\cdot\frac{T_1}{T_2k^2}, \qquad H_0 = \frac{-1}{k},$$

which make a low-pass filter of variable centre frequency ω_0, variable selectivity Q and constant bandwidth $\Delta\omega_{-3dB} = k/T_1$.

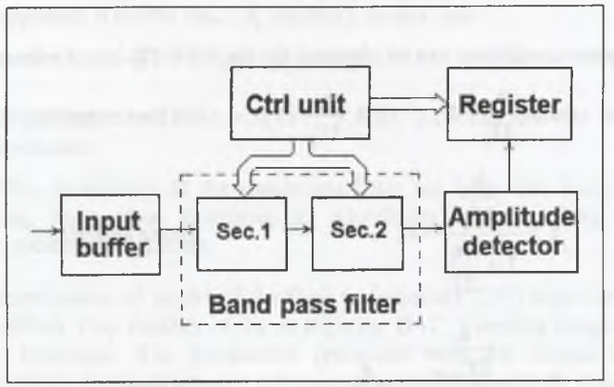

Fig. 5.8. Model of frequency division-mode telemetric receiver

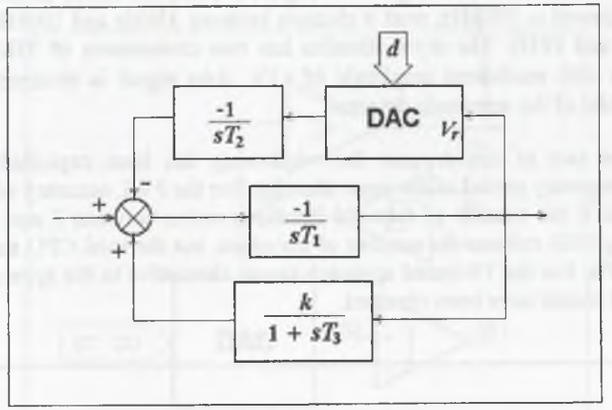

Fig. 5.9. Structure of single section band-pass filter

Apparently, the loop structure of Fig.5.9 represents an irreducible feedback system, and when using the PWL approximation-based technique, due to (5.20) for either section obtains:

$$1 \ge \frac{t^*}{2T_1}\cdot 2, \qquad 1 \ge \frac{t^*}{2T_2}\cdot 1, \qquad \text{for the integrators,} \qquad (5.43a)$$

$$1 \ge \left(1-\frac{T_3}{t^*}\left(1-e^{\frac{-t^*}{T_3}}\right)\right)\cdot k, \qquad \text{for the amplifier,} \qquad (5.43b)$$

$$1 \ge \left(1-\frac{T_{DAC}}{t^*}\left(1-e^{\frac{-t^*}{T_{DAC}}}\right)\right)\cdot\frac{d}{d_{mx}}, \qquad \text{for DAC.} \qquad (5.43c)$$

The condition involved with DAC may be neglected, since $d/d_{mx} < 1$. The same holds for the amplifier, if $k \le 1$. Hence, $t^* \le Min(T_1, 2T_2)$, and for the given parameter set [T_1, T_2, T_3, k, d_{mx}] = [1.6μs, 1.6μs, 15ns, 1, 256] obtains: $t^* \le 1.6\mu s$. This result enables us to define a

segment length limiter for the PWL approximation-based algorithm to assure its stable operation.

Similar convergence conditions can be obtained for the PWL TR-based relaxation (see (5.32))

$$1 \geq \frac{h_i}{2T_1} \cdot 2, \quad 1 \geq \frac{h_i}{2T_2} \cdot 1, \qquad \text{for the integrators,} \qquad (5.44a)$$

$$1 \geq \frac{\frac{h_i}{2T_3}}{1 + \frac{h_i}{2T_3}} \cdot k, \qquad \text{for the amplifier,} \qquad (5.44b)$$

$$1 \geq \frac{\frac{h_i}{2T_{DAC}}}{1 + \frac{h_i}{2T_{DAC}}} \cdot \frac{d}{d_{max}}, \qquad \text{for DAC.} \qquad (5.44c)$$

Some of the simulation results are given in Fig.5.10. First, the centre frequency f_0 of band-pass filter is adjusted to 100kHz, next it changes between 33kHz and 100kHz due to control signal d (55H and FFH). The input stimulus has two components of 33kHz and 100kHz frequency each with modulated amplitude of ±1V. *Adet* signal is obtained from a simple behavioural model of the amplitude detector.

To enhance the rate of convergence the windowing has been exploited with windows matching the temporary period of the input stimulus. For the PWL accuracy of 50mV and WR accuracy of 10mV the number of required iterations varies between 7 and 15 for different windows. Using OSR reduces the number of iterations, but the total CPU simulation time is increased by 10%. For the TR-based approach (as an alternative to the approximation-based) almost identical results have been obtained.

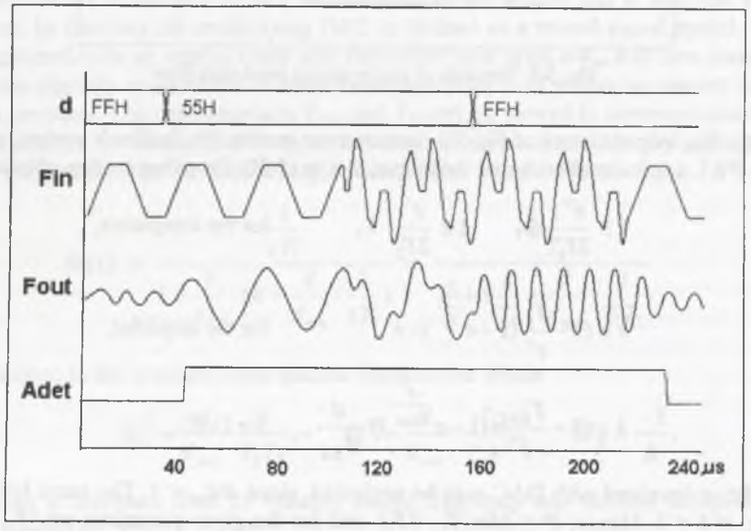

Fig.5.10. PWL waveforms for frequency-division mode telemetric receiver

Some other improvement of the convergence speed might be expected with the successive over-relaxation approach [OGR94 Sec.3.4, VAR63]. In this case

$$x^{j+1} = (1-\omega)x^j + \omega \hat{x}^{j+1} \qquad (5.45)$$

where the relaxation factor $\omega \in (1, 2)$, and $\hat{x}^{j+1} = F(x^j)$, which denotes the basic Gauss-Seidel relaxation scheme.

The WR/OSR-PWL simulation of the standalone filter has been also compared to SPICE estimates showing very close matching in waveforms (approx. 2%) with speed-up approaching two orders of magnitude.

Example 2. A functional-level model of the digit to frequency (D/f) converter is depicted in Fig.5.11. The feedback loop consists of the multiplying DAC, inverting integrator and voltage comparator with hysteresis. The comparator (provided with the inertial block) delivers alternatively a positive- or negative-value reference signal Ur to the DAC. Apart from the two switching points (±Vth in DC characteristic) the comparator features zero gain, so the convergence condition (5.20) is not applicable here. The model of DAC is similar to that of Example 1.

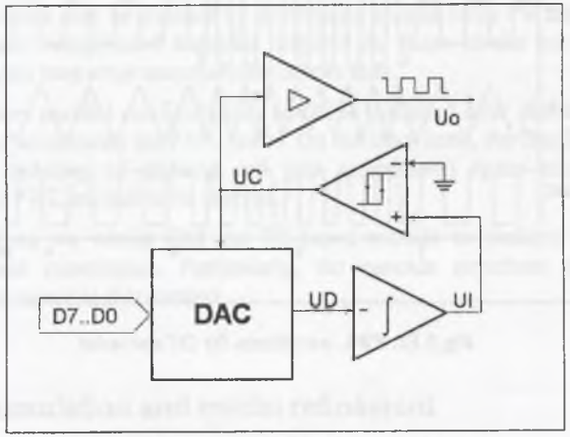

Fig.5.11. Structure of digit to frequency converter

In Fig.5.12 some of the waveforms obtained for the D/f converter are plotted. For constant input the comparator delivers a square wave of stable frequency proportional to the input digital word. Since the period of switching can change, there is no easy way to adjust an optimal size of the time windows for the WR. For example, if the events at the input D07 are used to define the window boundaries, the number of iterations for a time window equals the number of signal changes (between -Ur and +Ur) at the comparator output within this window. In this case for the first window, starting at t=0 and ending on the first change of D07, obtains 7 iterations (the last one is only to check for convergence). In the next window as many as 17 iterations occur.

Moreover, during the initial iterations the integrator tends to produce very large amplitudes at its output, unless it is provided with a limiter. For example, in the first iteration of the first window UI falls linearly until the window end is reached (with no limiter). In the second iteration, it falls only until the comparator threshold is crossed and UD changes to negative.

Next, UI rises linearly as long as the window boundary is again reached. (i.e. partial convergence is observed for each iteration [DEB87]).

Fortunately, the OSR performs much better in this case as its takes advantage of the partial convergence. The initial window follows the first two events on D07, but once the comparator switches at its output, a new event occurs and limits the window size for the following blocks. In other words, the windows match the actual PWL segments. For the windows where UD remains constant, the one-segment waveforms converge in only two iterations. Otherwise, when UD changes between a positive and negative value, more iterations are required for that window, e.g. three to five iterations for Ur = 2V, Vth = 0.5V, PWL accuracy of 50mV and relaxation accuracy of 10mV. As a consequence, the CPU-time savings obtained with OSR are as much as 30%. This result is mainly due to the partial convergence. That is, it comes out from avoiding unnecessary iterations over the former PWL segments that already converged, as opposed to the WR with windows comprising more segments.

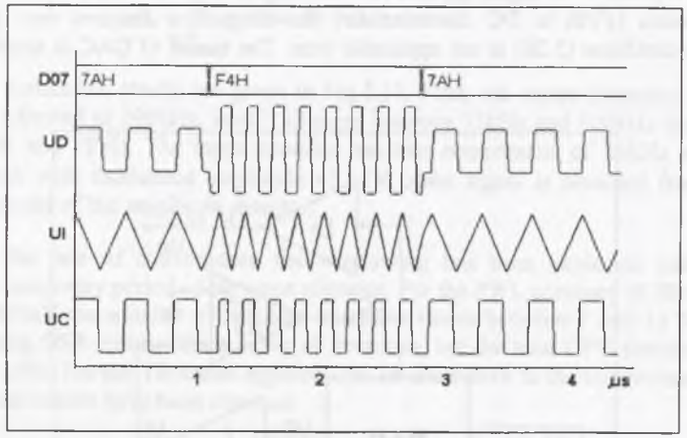

Fig.5.12. PWL waveforms for D/f converter

6. SYNTHESIS OF COMPLEX ANALOGUE MODELS

In this chapter we perform a synthesis of complex PWL models relevant to higher order linear blocks. These usually represent some analogue filters used in mixed A/D systems to be designed. Nonlinearities, such as saturation or slew-rate can be incorporated as well. Unless the architecture of such functional units is defined, one is free to represent them in a most suitable way to assure effectiveness of the macrosimulation process used. Hence, making an abstraction from the prospective architecture of the unit seems to be useful. As a consequence, the discussed problem has a few aspects.

First, the higher order models are inclined to accumulate the PWL errors when modelled with basic building blocks arranged in cascades or loop structures. In this case an attempt can be made to reduce or to compensate for the resulting global errors, already mentioned in Section 5.3.

Second, loop structures may be avoided by developing second-order PWL building blocks at the expense of more complicated formulas required for those blocks but with no need of iterations. Clearly, no loop error accumulation occurs then.

Third, in some cases parallel configurations could be preferred over cascade configurations that usually tend to accumulate the PWL errors. On the other hand, the flexibility in designing of cascades (i.e. ordering of sections and gain assignment) opens some new area for optimisation of the PWL models to be derived.

In some applications we would find the TR-based models to perform better than their approximation-based counterpart. Particularly, the cascade structures and second-order building blocks are meant in this context.

6.1 Error accumulation and model refinement

In order to investigate the effect of error accumulation consider a simple loop structure composed of a single integrator, which output is fed back directly to the input via the PWL approximator. Such a model can be described as

$$\begin{aligned} T\dot{x} &= u, \\ u &= u_{in} - L(x) \end{aligned} \tag{6.1}$$

where x denotes the smooth output, and u_{in} the PWL input of the network. Observe that without the approximator eqns.(6.1) make a perfect inertial block of unity gain and time constant equal T. In practice, the inertial building block defined in Chpt.2 would be preferred over this loop structure. Here, however, it serves as an analytical example.

Assume the input $u_{in} = rt$ ($r > 0$), and $x(0)=0$. Invoking eqns.(2.35-36) we observe propagation of the resulting first PWL segment in the loop (i.e. during iterations). Its length is $t_1 = \sqrt{8Tp_{mx}/r}$, and hence, this first segment ends at

$$x_1^{(1)} = \frac{r\cdot(t_1)^2}{2T} = \frac{r\cdot\frac{8Tp_{mx}}{r}}{2T} = 4p_{mx}, \qquad (6.2a)$$

For the second iteration one obtains $u^{(2)} = rt - 4p_{mx}t/t_1 = (r - \sqrt{\frac{2p_{mx}r}{T}})\cdot t$, so if $p_{mx} < rT/2$ then $t_1^{(2)} = \sqrt{\frac{8Tp_{mx}}{r - \sqrt{\frac{2p_{mx}r}{T}}}} > t_1^{(1)} = t_1$, and the algorithm retains the former segment length t_1.

Thus,

$$x_1^{(2)} = \frac{(r - \sqrt{\frac{2p_{mx}r}{T}})\cdot(t_1)^2}{2T} = 4p_{mx}(1 - \sqrt{\frac{2p_{mx}}{rT}}), \qquad (6.2b)$$

and next

$$x_1^{(3)} = 4p_{mx}(1 - \sqrt{\frac{2p_{mx}}{rT}} + \frac{2p_{mx}}{rT}), \qquad (6.2c)$$

so that for $p_{mx} < rT/2$ the segment length remains unchanged, i.e. $t_1^{(k)} = t_1 = \sqrt{\frac{8Tp_{mx}}{r}}$ and

$$x_1^{\infty} = \lim_{k\to\infty} x_1^{(k)} = \frac{4p_{mx}}{1 + \sqrt{\frac{2p_{mx}}{rT}}} \qquad (6.3)$$

On the other hand, the original value $\hat{x}(t_1)$ obtained from (6.1) without operator $L(\cdot)$ is

$$\hat{x}(t_1) = rT\cdot e^{-\frac{t_1}{T}} + r(t_1 - T) = rT\cdot(e^{-\sqrt{\frac{8p_{mx}}{rT}}} - 1) + \sqrt{8Tp_{mx}r} \qquad (6.4)$$

Table 6.1. Global error ε_1 and maximum error ε_{mx} against approximation accuracy p_{mx} and the involved correct value $\hat{x}(t_1)$ for $r = T = 1$

$p_{mx}\times10^{-3}$	$t_1\times10^{-3}$	$\hat{x}(t_1)\times10^{-3}$	$\varepsilon_1\times10^{-3}$	$\varepsilon_{mx}\times10^{-3}$
5	200	19	0.55	4.9
10	283	37	1.4	9.6
20	400	70	3.7	18
50	633	164	11.8	43
80	800	249	20.8	65
100	894	303	26.9	78

Now, define the global error for the first breakpoint coming at t_1 as $\varepsilon_1 = x_1^{\infty} - \hat{x}(t_1)$. The resulting values of ε_1 for different p_{mx} are shown in Table 6.1. The parameters r and T are normalised, i.e. $r = T = 1$.

As shown, those errors might be kept low with the reduced value of p_{mx}, which in turn results in reducing the corresponding segment length. On the other hand, the involved maximum errors $\varepsilon_{mx} = \underset{t\in[0,t_1]}{Max} \left| x_{lin}(t) - \hat{x}(t) \right|$, which are more important, are less than the accuracy assumed. Apparently, the loop structure tends to reduce the amplitudes of PWL errors as compared to the accuracy assumed, although some errors for breakpoints arise. However, when needed, it is possible to compensate for this kind of error at the expense of some extra calculations.

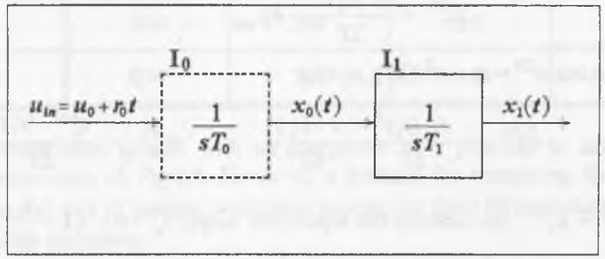

Fig. 6.1. Test structure for PWL errors compensation

To discuss this, first consider a cascade structure of two integrators $\mathbf{I}_0$, $\mathbf{I}_1$ depicted in Fig.6.1. Assume the output of the driving integrator to be

$$x_0(t) = x_0(0) + \frac{1}{T_0}(u_0 t + \frac{r_0 t^2}{2}) \qquad (6.5)$$

Now, we can calculate the exact response of $\mathbf{I}_1$ to $x_0(t)$ rather than to the PWL approximation $L[x_0(t)]$, which in this case is overestimated (i.e. $x_0(t) < L[x_0(t)]$ for $t < t_1$)

$$\hat{x}_1(t) = x_1(0) + x_0(0)\frac{t}{T_1} + \frac{u_0 t^2}{2T_0T_1} + \frac{r_0 t^3}{6T_0T_1} \qquad (6.6)$$

Consequently, for $t_1 = \sqrt{8T_0p_{mx}/r_0}$ one obtains $\hat{x}_1(t_1)$ rather than $x_1(t_1)$, which would be the response to $L[x_0(t)]$. To take advantage of this refinement $\hat{x}_1(t)$ should match the standard quadratic response $x_{1q}(t)$ such as (6.5), i.e.

$$x_{1q}(t) = x_1(0) + x_0(0)\frac{t}{T_1} + \frac{r_q t^2}{2T_1} \qquad (6.7)$$

By letting $x_{1q}(t_1) = \hat{x}_1(t_1)$ an equivalent input slope is

$$r_q = \frac{u_0}{T_0} + \frac{r_0 t_1}{3T_0} \qquad (6.8)$$

Otherwise, we would have $r = \frac{u_0}{T_0} + \frac{r_0 t_1}{2T_0}$, which is the slope of $L[x_0(t)]$ obtained from (6.5). Apparently, $r_q < r$, so that the PWL overestimation of $x_0(t)$ is reduced. In fact, the obtained

linear approximation $[x_0(0) + r_q t]$ crosses $x_0(t)$ at some $t_q < t_1$, splitting the PWL segment in two parts, one overestimated and the other underestimated, unlike $L[x_0(t)]$. In this way, we would find this technique to provide the PWL approximation, which is balanced better, against the original waveform, than that of the standard operator $L(\cdot)$. As a result, $x_{1q}(t)$ fits better to the original response $\hat{x}_1(t)$ than $x_1(t)$ as stated above.

Observe that before using eqns.(6.6-6.8) the parameters $x_0(0)$, u_0, r_0, T_0 must be passed from the integrator $\mathbf{I}_0$ to $\mathbf{I}_1$. By means of such a refinement one can compensate for the PWL errors arising at breakpoints.

Consider again the loop structure defined by (6.1). Using the same input $u_{in} = rt$ ($r > 0$), and $x(0)=0$, the first iteration yields

$$x_1^{(1)} = \frac{r\cdot(t_1)^2}{2T} = 4p_{mx} \tag{6.9a}$$

like in eqn.(6.2*a*) but $u^{(2)} = rt - rt^2/(2T)$, so that

$$\hat{x}_1^{(2)} = \frac{r\cdot(t_1)^2}{2T} - \frac{r\cdot(t_1)^3}{6T^2}, \qquad x_{1q}^{(2)} = \frac{r_q^{(2)}\cdot(t_1)^2}{2T} \tag{6.9b}$$

By letting $x_{1q}^{(2)} = \hat{x}_1^{(2)}$ one obtains the equivalent slope $r_q^{(2)} = r\cdot(1-\frac{t_1}{3T})$. Next, for the third iteration

$$u^{(3)} = rt - \frac{r_q^{(2)}t^2}{2T} = rt - r(1-\frac{t_1}{3T})\frac{t^2}{2T}, \qquad \text{and}$$

$$\hat{x}_1^{(3)} = \frac{r\cdot(t_1)^2}{2T} - \frac{r\cdot(1-\frac{t_1}{3T})(t_1)^3}{6T^2}, \qquad x_{1q}^{(3)} = \frac{r_q^{(3)}\cdot(t_1)^2}{2T} \tag{6.9c}$$

Again $x_{1q}^{(3)} = \hat{x}_1^{(3)}$ implies $r_q^{(3)} = r\cdot[1-\frac{t_1}{3T}+\frac{(t_1)^2}{(3T)^2}]$. Apparently, if $\frac{t_1}{3T} < 1$ ($p_{mx} < 9rT/8$), then

$$r_q^{\infty} = \lim_{k\to\infty} r_q^{(k)} = \frac{r}{1+\frac{t_1}{3T}} \tag{6.10}$$

$$\hat{x}_1^{\infty} = x_{1q}^{\infty} = \frac{r_q^{\infty}\cdot(t_1)^2}{2T} = \frac{4p_{mx}}{1+\sqrt{\frac{8p_{mx}}{9rT}}} \tag{6.11}$$

Using the formula (6.11) the compensated PWL breakpoint errors, defined as $\varepsilon_{1r} = \hat{x}_1^{\infty} - \hat{x}(t_1)$, can be evaluated (see Table 6.2). As compared to the previous results, summarised in Table 6.1, the errors ε_1 are reduced here 5.5 times for $p_{mx} = 0.100$ up to 27 times for $p_{mx} = 0.005$. Apparently, the maximum errors ε_{mx} are also reduced. Since ε_1 and ε_{1r} reflect a linear component in $\varepsilon(t)$, so $\varepsilon_{rmx} \approx \varepsilon_{mx} - (\varepsilon_1 - \varepsilon_{1r})/2$.

Moreover, observe that in this case the TR-based algorithm would perform in the same way, since the basic PWL model of an integrator makes use of formulas common for both methods.

Table 6.2. Compensated global error ε_{1r} and maximum error ε_{rmx} against approximation accuracy p_{mx} and the involved correct value $\hat{x}(t_1)$

$p_{mx}\times10^{-3}$	$t_1\times10^{-3}$	$\hat{x}(t_1)\times10^{-3}$	$\varepsilon_{1r}\times10^{-3}$	$\varepsilon_{rmx}\times10^{-3}$
5	200	19	0.02	4.7
10	283	37	0.07	9.0
20	400	70	0.27	16.4
50	633	164	1.4	37.6
80	800	249	3.3	56.0
100	894	303	4.9	67.6

Based on the experience gained with an integrator it is possible to enhance the PWL approximation procedure of Fig.2.6. However, a formula for computing the exact response $\hat{x}_1(t_1)$ as well as the way of passing waveform parameters from input to output of the inertial block need separate derivation.

To cope with this problem assume an inertial input waveform in the standard form

$$x_0(t) = s_0 e^{-t/T_0} + r_0(t-T_0) + u_0 \tag{6.12}$$

where $s_0 = x_0(0) - u_0 + r_0T_0$. The corresponding time response of an inertial block (of unity gain, time constant T_1 and without PWL approximator) is provided with two transient components

$$\hat{x}_1(t) = [x_1(0) - u_0 + r_0(T_0+T_1) - \frac{s_0T_0}{T_0-T_1}]e^{-t/T_1} + \frac{s_0T_0}{T_0-T_1}e^{-t/T_0} + r_0[t-(T_0+T_1)] + u_0 \tag{6.13}$$

On the other hand, an equivalent standard response may be assumed as

$$x_{1e}(t) = [x_1(0) - x_0(0) + r_{1e}T_1]e^{-t/T_1} + r_{1e}(t-T_1) + x_0(0) \tag{6.14}$$

and $x_{1e}(t_1) = \hat{x}_1(t_1)$, where t_1 is found from (2.17) for $p_{mxr} = p_{mx} / \left| x_1(0) - x_0(0) + r_{0e}T_1 \right|$, where r_{0e} corresponds to PWL approximation of (6.12). Hence, one obtains

$$r_{1e} = r_0 + s_0\cdot\frac{\frac{T_1}{T_1-T_0}e^{-t_1/T_1} + \frac{T_0}{T_0-T_1}e^{-t_1/T_0} - 1}{t_1 + T_1(e^{-t_1/T_1}-1)} \tag{6.15}$$

Clearly, eqns. (6.14-6.15) are required to proceed $x_{1e}(t)$ further.

Similar formulas can be derived for a quadratic input waveform, such as (6.5) that represents the inertial block driven by an integrator. In this case

$$\hat{x}_1(t) = [x_1(0) - x_0(0) + \frac{u_0 - r_0 T_1}{T_0} \cdot T_1] e^{-t/T_1} + \frac{u_0 - r_0 T_1}{T_0} \cdot (t - T_1) + \frac{r_0 t^2}{2T_0} + x_0(0) \qquad (6.16)$$

$$r_{1e} = \frac{u_0 - r_0 T_1}{T_0} + \frac{r_0 (t_1)^2}{2T_0 \cdot [t_1 + T_1 (e^{-t_1/T_1} - 1)]} \qquad (6.17)$$

that enable to put $\hat{x}_1(t)$ into the standard form of (6.14).

The enhanced PWL approximator relevant to the inertial building block is given in Fig.6.2. As opposed to the procedure of Fig.2.6, here the input waveform must be recognised for shape (quadratic or exponential), and the involved shape parameters must be passed to the model. The actual segment length is evaluated based on u_0 and the equivalent r coming from the driving block. Consequently, to compensate for PWL errors, the PWL breakpoints are calculated either with eqns.(6.14-6.15) or (6.14), (6.17), and a new equivalent r is calculated using (6.8) or (6.15). In practice, the input waveform can be identified based on the fan-in table, available after the netlist is compiled.

```
identify_input_shape;
select_formula_x;    (* to compute x̂(τ1T) *)
repeat
  if x0 - u0 + rT <> 0 and | x0 - u0 + rT | > pmx
                then {τ1 ← Φ(pmx);
                      if τ1 > τmx then τ1 ← τmx}
                else τ1 ← τmx;
       xk ← x̂(τ1T); x0 ← xk;
       u0 ← ( u0 + rτ1T );
       tk ← ( tk-1 + τ1T );
       k ← ( k + 1 );
       τmx ← ( τmx - τ1 );
       compute_equivalent_r;
until τmx = 0
```

Fig. 6.2. Algorithm for enhanced PWL approximator

Using the enhanced approximator for the loop structures allows reducing the PWL global breakpoint errors as well as the maximum errors. Some of the results obtained for the 2nd order Bessel filter (already discussed in Section 5.3) are shown in Fig.6.3 (the same input stimulus has been applied).

The compensation mechanism affects mainly the error component relevant to PWL breakpoints, and the maximum error amplitudes are reduced by some (ε_1 - ε_{1r})/2. However, the required CPU simulation-time is approximately as much as twice of that with the standard approximator (as compared to the results of Sec.5.3).

On the other hand, as opposed to integrators, the inertial blocks do not need this kind of error compensation when processed with the TR-based PWL technique. It is because the TR (and in particular the enhanced TR) provides self-compensation, as shown in Section 2.4 (see Fig.2.12). In practice, when applied to cascades, the TR-based PWL technique and the enhanced PWL approximation technique produce very similar results (except of overshoots in

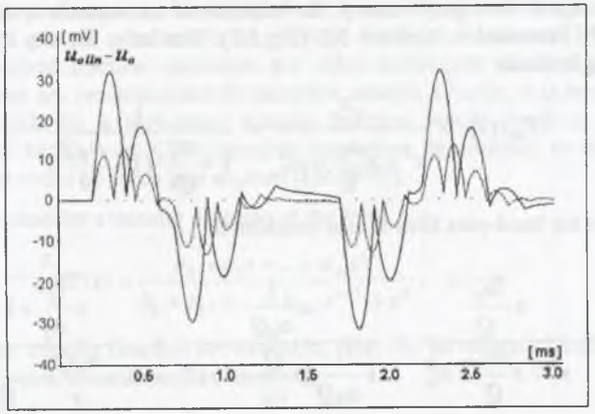

Fig. 6.3. Reduced global WR-PWL errors corresponding to waveforms of Fig.5.4 obtained with enhanced PWL approximator for p_{mx} =50mV (solid line) and p_{mx} =20mV (dashed line).

some cases, shown in Sec.2.5), but the latter requires more computations. Because of it in the following section we will take advantage of the TR-based PWL technique to keep the global PWL error low, in particular for cascade structures.

6.2 Synthesis with basic building blocks

At the functional-level a linear analogue unit can be adequately defined by the transfer function. Nonlinearities may be incorporated too, as shown in Chapter 3. For the purpose of modelling with the PWL technique, some synthesis procedures relevant to RC active filters could be used. A variety of approaches exist [BIA79], which account for such aspects as: a number of required active elements, sensitivity to RC elements' variations, flexibility of tuning, power consumption etc. However, if an abstraction from the prospective architecture of the unit can be made (as suggested earlier), most of those design aspects might be omitted, and in this case mainly the resulting model structure is of interest. On the other hand, different synthesis issues arise. These are complexity of a model (i.e. of computations), PWL error accumulation and need for iterations when feedback loops are required.

We start with a simple synthesis pertaining to the second order transfer functions. A quality factor of $Q > ½$ (i.e. for complex poles) is assumed. For instance, the standard low-pass function may be decomposed as follows

$$T_{LP}(s) = \frac{\omega_0^2}{s^2 + s\frac{\omega_0}{Q} + \omega_0^2} = \frac{\omega_0 Q}{s(1 + s\frac{Q}{\omega_0}) + \omega_0 Q} = \frac{\frac{\omega_0 Q}{s(1 + s\frac{Q}{\omega_0})}}{1 + \frac{\omega_0 Q}{s(1 + s\frac{Q}{\omega_0})}} \qquad (6.18)$$

By virtue of the signal-flow graph theory, the latter result corresponds to the feedback loop structure, already discussed in Section 5.3 (Fig.5.3). Similarly, for the high-pass transfer functions one might obtain

$$T_{HP}(s) = \frac{s^2}{s^2 + s\frac{\omega_0}{Q} + \omega_0^2} = \frac{1}{1 + \frac{\omega_0}{Qs}\cdot(1 + \frac{\omega_0 Q}{s})} \tag{6.19}$$

For completeness the band-pass filter is also considered

$$T_{BP}(s) = \frac{s\cdot\frac{\omega_0}{Q}}{s^2 + s\frac{\omega_0}{Q} + \omega_0^2} = \frac{s\frac{1}{\omega_0 Q}}{s\frac{1}{\omega_0 Q}(s\frac{Q}{\omega_0}+1)+1} = \frac{\frac{1}{s\frac{Q}{\omega_0}+1}}{1+\frac{\omega_0 Q}{s}\cdot\frac{1}{s\frac{Q}{\omega_0}+1}} \tag{6.20}$$

The respective structures composed of the basic building blocks are presented in Figs.6.4 - 6.5. The corresponding PWL models are provided with the approximators attached to the output of each building block. Alternatively, those blocks can be discretised with respect to the TR algorithm.

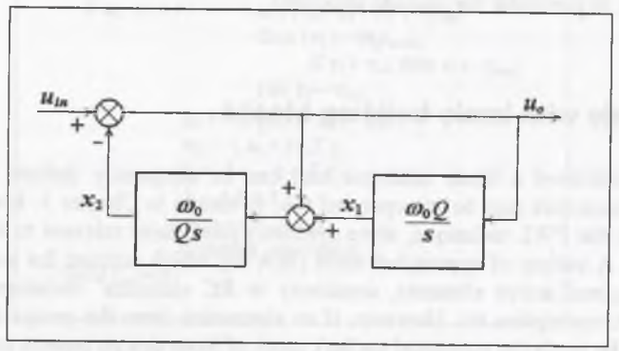

Fig. 6.4. Functional model of bi-quad high-pass filter

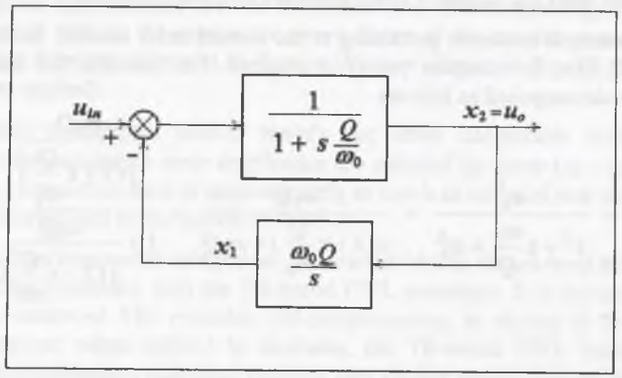

Fig. 6.5. Functional model of bi-quad band-pass filter

The above decomposition may be regarded as a special case of the continued fraction expansion that in general case results in multiple feedback structures [BIA87]. However, neither the continued fraction expansion nor other techniques oriented towards multiple feedback structures are recommended for complex models. Clearly, it is because of the need of iterations, which for a high-order transfer function would result in slow relaxation convergence, and hence, very CPU-intensive simulation. In contrary to this, cascade- and parallel structures could be taken into account [DAB99S].

For this purpose consider a transfer function of the form

$$H(s) = \frac{a_0 + a_1 s + \ldots + a_m s^m}{b_0 + b_1 s + \ldots + b_{n-1} s^{n-1} + s^n}, \quad n > m \tag{6.21}$$

If the poles of the transfer function are available, $H(s)$ can be expanded into a sum of partial fractions (single poles are assumed for simplicity)

$$H(s) = \sum_{i=1}^{n} \frac{h_i}{s - s_i}, \qquad h_i = \underset{s=s_i}{Res}\, H(s) \tag{6.22}$$

Clearly, the components with real poles represent inertial blocks with the time constants equal $(-1/s_i)$, excluding the poles equal zero that correspond to integrators. On the other hand, for the pairs of conjugate poles the second order structures, such as (6.18-6.20) are adequate.

Moreover, if the zeros of $H(s)$ are known, a cascade connection (often used for practical filters) consisting of the first- and second order blocks can be exploited

$$H(s) = a_m \frac{(s - z_1)(s - z_2)\ldots(s - z_m)}{(s - s_1)(s - s_2)\ldots(s - s_n)} \tag{6.23}$$

Clearly, the conjugate zeros should be grouped into pairs as well, to match realistic building blocks.

To make some comparison consider a simple example of the two-pole low-pass transfer function with $Q < ½$. In this case it can be represented with its time constants

$$H_2(s) = \frac{1}{(1 + sT_1)\cdot(1 + sT_2)} \tag{6.24}$$

Three PWL models will be investigated for (6.24): parallel structure, cascade structure $1/(1+sT_1) \rightarrow 1/(1+sT_2)$ and reversed cascade structure $1/(1+sT_2) \rightarrow 1/(1+sT_1)$, each provided with the PWL approximator. For the parallel structure one obtains

$$H_2(s) = \frac{T_1}{T_1 - T_2}\cdot\frac{1}{1 + sT_1} + \frac{T_2}{T_2 - T_1}\cdot\frac{1}{1 + sT_2} \tag{6.25}$$

The PWL models' performance is exemplified in Figs.6.6-6.10 for T_1 = 2ms and T_2 = 1ms and the approximation accuracy of p_{mx} = 80mV.

In the lower plot of Fig.6.6 the waveforms $\hat{x}_1$, $\hat{x}_2$ of the cascade (exact responses of the 1st- and 2nd stage, respectively) against u_{in} are shown. The PWL responses are not presented for clarity of the diagram. The upper plot shows the PWL errors $\varepsilon_1(t)$, $\varepsilon_2(t)$ relevant to $\hat{x}_1(t)$, $\hat{x}_2(t)$, respectively. Apparently, ε_1 is zero-valued at breakpoints, and its amplitudes are equal to p_{mx} (with the exception of the "closing" segments that end at t = 2ms and t = 10ms), whereas ε_2 exhibits the effect of temporal error accumulation, and it varies between −130mV and +113mV.

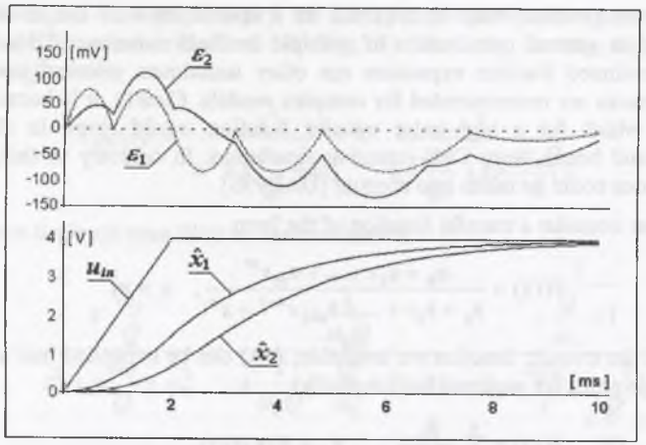

Fig. 6.6. Exact responses of cascade stages against input waveform (lower plot), and corresponding PWL errors for p_{mx} =80mV (upper plot)

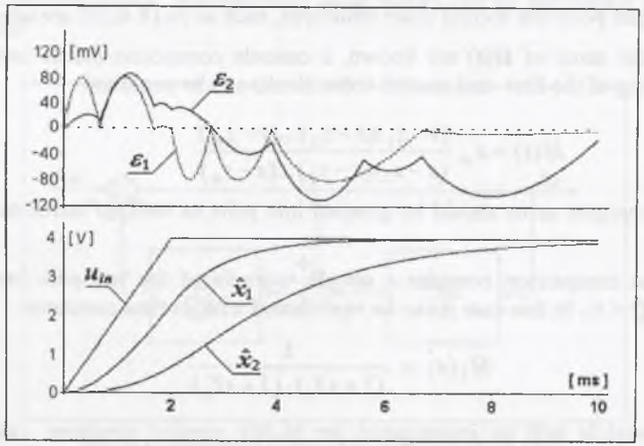

Fig. 6.7. Exact responses of reversed cascade stages against input waveform (lower plot), and relevant PWL errors for p_{mx} =80mV (upper plot)

The waveforms involved with the reversed cascade structure are shown in Fig.6.7. Apparently, the amplitudes of ε_2 are here less than those of the primary cascade, and its range is (-110mV, +86mV). In fact, the second stage plays, in some sense, a role of a dumping filter for ε_1. Clearly, the reversed structure is more effective because of a larger time constant in the second stage (2ms). Observe also that the dumping works better for the fast changing portions of the waveform.

The latter model can be enhanced further, when using stages of diversified gain. The corresponding results for the reversed cascade of $\mathbf{2}/(\mathbf{1}+s\cdot\mathbf{1}\text{ms}) \rightarrow \mathbf{0.5}/(\mathbf{1}+s\cdot\mathbf{2}\text{ms})$ are shown in Fig.6.8, where ε_2 varies between –107mV and +54mV. Similar results for the cascade of $\mathbf{5}/(\mathbf{1}+s\cdot\mathbf{1}\text{ms}) \rightarrow \mathbf{0.2}/(\mathbf{1}+s\cdot\mathbf{2}\text{ms})$ are shown in Fig.6.9 (with $\varepsilon_2 \in$ (-89mV, +28mV)). Since the approximation accuracy here is the same (80mV), so are the amplitudes of ε_1. However, as compared to the case of equal-gain stages, the errors ε_1 are damped the stronger, the less the

second-stage gain. For sufficiently small gain of the second stage (sufficiently large gain of the fist stage, respectively) one could expect the errors not to accumulate (compare ε_1 and ε_2 in Fig.6.9). On the other hand, a trade-off exists. The arising large amplitudes at the first stage output cause an increasing of the number of PWL segments. As a consequence, this approach is equivalent to reducing of p_{mx}. Also here, the dumping is rather poor for the slowly changing part of $\hat{x}_1$.

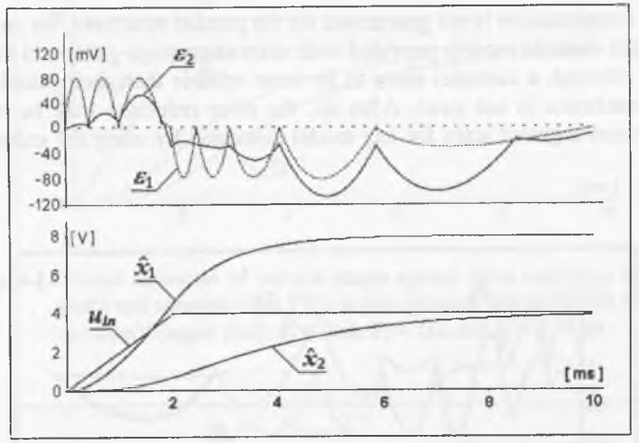

Fig. 6.8. Exact responses of reversed cascade stages with diversified gains (k_1 = 2 and k_2 = 0.5) against input waveform (lower plot), and corresponding PWL errors for p_{mx} =80mV (upper plot)

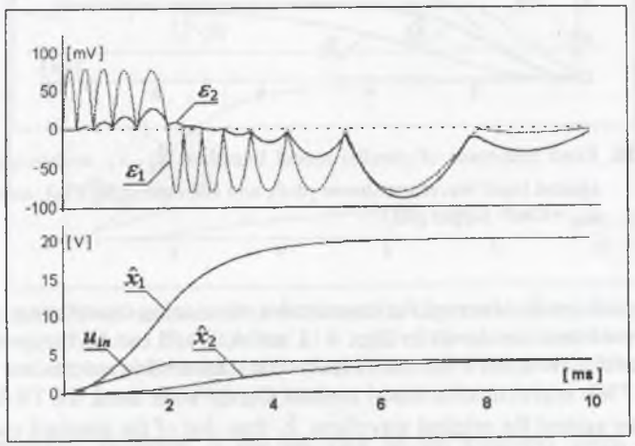

Fig. 6.9. Exact responses of reversed cascade stages with diversified gains (k_1 = 5 and k_2 = 0.2) against input waveform (lower plot), and corresponding PWL errors for p_{mx} =80mV (upper plot)

The performance of the parallel model is depicted in Fig.6.10. As opposed to the cascade structures, here the PWL errors relevant to parallel branches of the model are independent from each other, and moreover, are likely to compensate when summing the branch signals. As a consequence, the resulting PWL output errors are within (-68mV, +79mV). However, the effect of error compensation is not a general rule for parallel models. In some cases error accumulation can occur as well.

Since the error compensation is not guaranteed for the parallel structures, we conclude that the properly arranged cascade models provided with decreasing stage-gains and increasing stage time constants (through a cascade) seem to be more reliable than their parallel counterpart. Clearly, this conclusion is not strict. After all, the error reduction may be obtained at the expense of reduced segment sizes for any model discussed by using the enhanced accuracy p_{mx}.

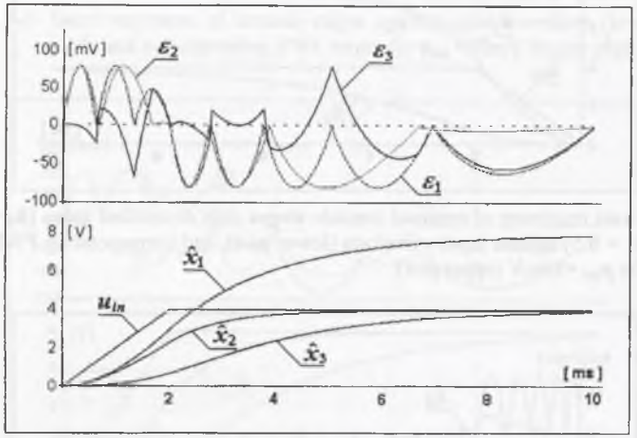

Fig. 6.10. Exact responses of parallel model branches $\hat{x}_1$, $\hat{x}_2$ and output $\hat{x}_3$ against input waveform (lower plot), and corresponding PWL errors for p_{mx} =80mV (upper plot)

Different behaviour can be observed for the cascades when using the enhanced TR technique. The relevant waveforms are shown in Figs. 6.11 and 6.12 and can be compared to Figs. 6.6 and 6.9, respectively. Here, the PWL errors at the first stage tend to accumulate temporarily as opposed to the PWL approximation-based method. On the other hand, the TR-based response is balanced better against the original waveform $\hat{x}_1$ than that of the standard operator $L(\cdot)$. As a consequence, the resulting error amplitudes at the output stage are usually less than those obtained with the operator $L(\cdot)$. In Fig. 6.11 the error ε_1 at the front-end stage varies between -120mV and +91mV, whereas ε_2 is reduced to (-84mV, 77mV). Similarly, the PWL errors of the reversed cascade with diversified gains (Fig. 6.12) are $\varepsilon_1 \in$ (-167mV, 171mV) and $\varepsilon_2 \in$ (-58mV, 80mV).

From the latter results we conclude that no manipulations on cascade structure are required to keep the PWL errors at the cascade output low. The enhanced TR approach appears more reliable than its approximation-based counterpart, as far as the error accumulation at a cascade

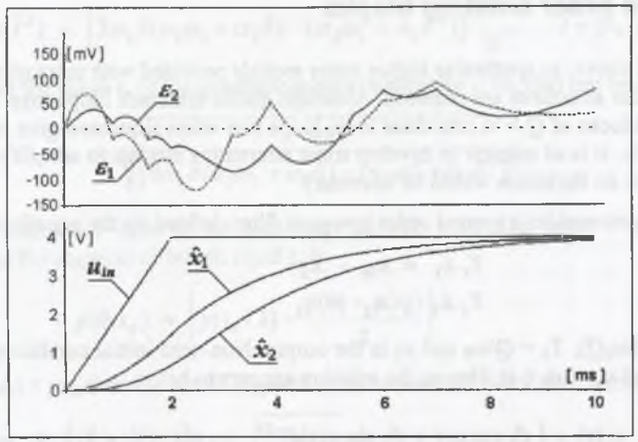

Fig. 6.11. Exact responses of cascade stages against input waveform (lower plot), and corresponding PWL errors obtained with enhanced TR for ε_0 =80mV (upper plot); T_1 = 2ms, T_2 = 1ms and $k_1 = k_2 = 1$

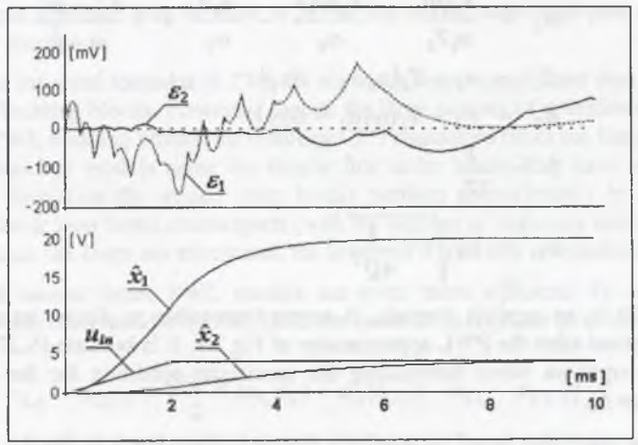

Fig. 6.12. Exact responses of reversed cascade stages with diversified gains (T_1 = 1ms, T_2 = 2ms and k_1 = 5, k_2 = 0.2) against input waveform (lower plot), and corresponding PWL errors obtained with enhanced TR for ε_0 =80mV (upper plot)

output is considered. However, it not the case of the front-end stage, for which the approximation-based algorithm proceeds perfectly and no error accumulation occurs then. On the other hand, the enhanced approximation-based PWL technique (derived in Sec. 6.1) that reduces the effect of error accumulation is computationally much more intensive. As a consequence, a tradeoff is faced here. If just the output of the cascade is of interest, using the enhanced TR technique seems to be the best choice.

6.3 Second order building blocks

As mentioned above, to synthesise higher order models provided with pairs of conjugate poles the second order structures are required. Although global feedback loops may be avoided, the second order blocks of $Q > ½$ introduce local loops that when simulated give rise to iterations as well. Clearly, it is of interest to develop some alternative models to simplify the simulation process, so that no iterations would be necessary.

For this purpose consider a second order low-pass filter defined by the equations

$$\begin{aligned} T_1 \dot{x}_1 &= u_{in} - x_2 \\ T_2 \dot{x}_2 + x_2 &= x_1 \end{aligned} \tag{6.26}$$

where $T_1 = 1/(\omega_0 Q)$, $T_2 = Q/\omega_0$ and x_2 is the output. Non-zero initial conditions are assumed: $x_1(0)$, $x_2(0)$ and $u_{in} = u_0 + rt$. Hence, the solution appears to be

$$\begin{aligned} x_1(t) &= \left(\beta_1 \cos\omega_1 t + \beta_2 \sin\omega_1 t\right)e^{-\delta t} + r\left(t - T_1 + T_2\right) + u_0 \\ x_2(t) &= \left(\alpha_1 \cos\omega_1 t + \alpha_2 \sin\omega_1 t\right)e^{-\delta t} + r\left(t - T_1\right) + u_0 \end{aligned} \tag{6.27}$$

where

$$\begin{aligned} \alpha_1 &= x_2(0) - u_0 + rT_1 \\ \alpha_2 &= \frac{x_1(0)}{\omega_1 T_2} - \frac{x_2(0)\delta}{\omega_0} - \frac{u_0\delta}{\omega_1} + r\frac{\delta^2 - \omega_1^2}{\omega_1\omega_0^2} \\ \beta_1 &= \alpha_1 + T_2\left(\omega_1\alpha_2 - \delta\alpha_1\right) \\ \beta_2 &= \alpha_2 - T_2\left(\omega_1\alpha_1 + \delta\alpha_2\right) \\ \delta &= \frac{1}{2T_2} \\ \omega_1 &= \omega_0\sqrt{1 - \frac{1}{4Q^2}} \end{aligned} \tag{6.28}$$

Although (6.27) is an explicit formula, it seems impossible to derive an approximation algorithm patterned after the PWL approximator of Fig.2.8. It is because (6.27) gives rise to transcendental equation when formulating the maximum condition for the corresponding distance function Δ_x.

Fortunately, the algorithm based on quadratic approximation given in Sec.2.3 is of use (see eqns. (2.9-2.11)). Thus, following this idea the formula for x_2 in (6.26) can be replaced by

$$y_q(t) = y(0) + \left.\frac{dy}{dt}\right|_{t=0} \cdot t + \frac{1}{2}\cdot\left.\frac{d^2y}{dt^2}\right|_{t=0} \cdot t^2 \tag{6.29}$$

where for simplicity we put $y = x_2$, and

$$\begin{aligned} \left.\frac{dy}{dt}\right|_{t=0} &= \alpha_2\omega_1 - \alpha_1\delta + r \\ \left.\frac{d^2y}{dt^2}\right|_{t=0} &= \alpha_1(\delta^2 - \omega_1^2) - 2\alpha_2\omega_1\delta \end{aligned}$$

Moreover, the third order Lagrange rest is

$$\Theta(\hat{t}^3) = \left[3\omega_1\delta(\alpha_1\omega_1 + \alpha_2\delta) - (\alpha_2\omega_1^3 + \alpha_1\delta^3)\right]\cdot\frac{\hat{t}^3}{3!}, \qquad \hat{t} \in (0,t) \tag{6.30a}$$

Hence, to control the range t_a of the quadratic approximation for y, (6.30a) can be converted to

$$t_a = \sqrt[3]{\frac{6\varepsilon_a}{\left|3\omega_1\delta(\alpha_1\omega_1 + \alpha_2\delta) - (\alpha_2\omega_1^3 + \alpha_1\delta^3)\right|}} \tag{6.30b}$$

where ε_a is an estimated value of the maximum allowed truncation error. The performance index relevant to the segment of length equal t_a is

$$p(0,t_a) = \left| y(t_a/2) - \frac{y(0) + y(t_a)}{2}\right| \tag{6.31}$$

Finally, if $p(0, t_a) > p_{mx}$ then the actual segment must be reduced due to the formula

$$t_1 = t_a \cdot \sqrt{\frac{p_{mx}}{p(0,t_a)}} \tag{6.32}$$

The algorithm of quadratic approximation applies in this case without any other modifications.

Clearly, the model defined by eqns. (6.26) is equivalent to the transfer function $T_{LP}(s)$ given by (6.18). The same approach may be used to derive the second order high-pass and band-pass PWL building blocks.

Apparently, the involved formulas (6.27-6.30) are much more complicated than those used for the first order building blocks. However, despite the large number of operations necessary for second order PWL building blocks, the resulting CPU simulation times are fairly moderate for them as compared to models using the simple first order blocks that have to iterate. As a result, during simulation the second order blocks perform approximately by 20-50% faster than their feedback loop based counterparts (with the number of iterations ranging between 8-12). Clearly, since the loops are eliminated, the involved PWL error accumulation is avoided.

The TR-based second order PWL models are even more efficient. To see this, recall eqns.(5.15), which correspond to (6.26). Here, the iteration index j can be dropped

$$\begin{aligned} x_{1,k} &= x_{1,k-1} + \frac{\omega_0 Q h_k}{2}\left[u_{in}(t_k) + u_{in}(t_{k-1}) - x_{2,k} - x_{2,k-1}\right] \\ x_{2,k} &= \frac{(1 - \frac{\omega_0 h_k}{2Q})x_{2,k-1} + \frac{\omega_0 h_k}{2Q}(x_{1,k} + x_{1,k-1})}{1 + \frac{\omega_0 h_k}{2Q}} \end{aligned} \tag{6.33}$$

After manipulations (6.33) can be rearranged to the explicit form of

$$\begin{aligned} x_{1,k} = {} & \frac{4Q + 2\omega_0 h_k - Q(\omega_0 h_k)^2}{D}x_{1,k-1} + \frac{-4\omega_0 h_k Q^2}{D}x_{2,k-1} \\ & + \frac{2\omega_0 h_k Q^2 + Q(\omega_0 h_k)^2}{D}\left[u_{in}(t_k) + u_{in}(t_{k-1})\right] \end{aligned} \tag{6.34a}$$

$$x_{2,k} = \frac{4\omega_0 h_k}{D} x_{1,k-1} + \frac{4Q - 2\omega_0 h_k - Q(\omega_0 h_k)^2}{D} x_{2,k-1} + \frac{Q(\omega_0 h_k)^2}{D}\left[u_{in}(t_k) + u_{in}(t_{k-1})\right] \tag{6.34b}$$

where $D = 4Q + 2\omega_0 h_k + Q(\omega_0 h_k)^2$.

As many as 30 elementary operations per step are required to calculate the response as compared to some 60 in the previous case. However, also here one can hardly develop an effective time-step control, such as that of the enhanced TR. Because of it the standard formula (2.6*b*) will be adopted as follows

$$h_k = \inf_i \left\{ \sqrt[3]{12\varepsilon_0 / \left| x^{(3)}_{i,k-1} \right|} \right\}, \quad x^{(3)}_{i,k-1} \neq 0, \quad i = 1,..n \tag{6.35}$$

where the third derivatives can be found by differentiating (6.26) that yields

$$x^{(3)}_{1,k-1} = \frac{T_1 x_{1,k-1} + (T_2 - T_1) x_{2,k-1} - T_2 u_{k-1}}{(T_1 T_2)^2}$$
$$x^{(3)}_{2,k-1} = \frac{T_1^2 (T_1 - T_2) x_{1,k-1} + T_1^2 (2T_2 - T_1) x_{2,k-1} + (T_1 T_2)^2 u^{(1)}_{k-1} - T_1^2 T_2 u_{k-1}}{(T_1 T_2)^3} \tag{6.36}$$

and $u_{k-1} = u_{in}(t_{k-1})$, $T_1 = 1/(\omega_0 Q)$, $T_2 = Q/\omega_0$.

The latter model will be preferred over the quadratic approximation-based model defined by formulas (6.27) – (6.32), since it is computationally more efficient, although the time-steps h_k appear to be shorter than the involved PWL segments, governed by eqns.(6.27-6.32). The CPU time required is by 20-30% shorter in this case.

To summarise, consider a 6th order Chebyshev low-pass filter arranged in cascade. The cut-off frequency of 30kHz is assumed and the overall gain equal unity. The characteristic frequencies and quality factors are as follows [BUR89]: $\omega_{01} = 190.7 \cdot 10^3$ rd/s, $Q_1 = 6.513$, $\omega_{02} = 144.8 \cdot 10^3$ rd/s, $Q_2 = 1.810$, $\omega_{03} = 74.7 \cdot 10^3$ rd/s, $Q_3 = 0.684$. Moreover, to reduce error accumulation between filter sections their gains are assumed to be: $k_1 = 5$, $k_2 = 1$, $k_3 = 1/5$.

The high-Q section (front-end) appears very sensitive to p_{mx} (ε_0) when modelled as a loop with first order building blocks (even with error compensation). In Fig.6.13 the waveforms relevant to the high-Q section are depicted for p_{mx} = 100mV. Apparently, the results obtained by the second-order building block are entirely under control as opposed to the feedback-loop model, for which the PWL approximators may strongly influence the self-oscillatory behaviour. Very similar results are obtained with the TR approach. Fortunately, the discrepancy in waveforms observed could be substantially reduced when using the accuracy of 50mV or less, both for the approximation-based and the TR-based technique. In this case all those filter models, including that of the second order building blocks, yield almost same results. In Fig.6.14 the response of the complete filter is given. Relatively small PWL errors (< 70mV) accumulate at the cascade output.

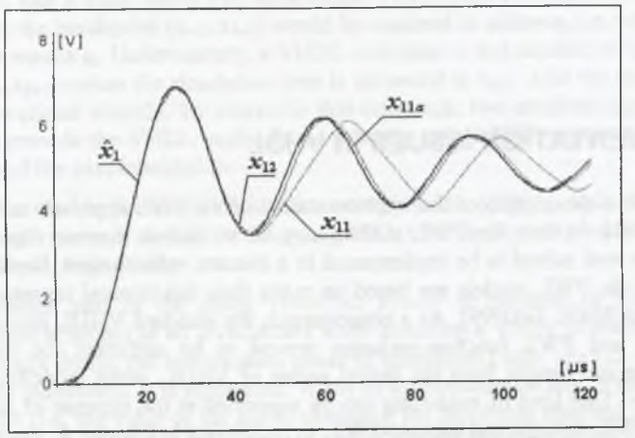

Fig. 6.13. PWL responses x_{11} (feedback-loop model) and x_{12} (second order building block) of high-Q filter section for p_{mx} =100mV against exact response $\hat{x}_1$ (x_{11a} is the feedback-loop response for p_{mx} = 50mV)

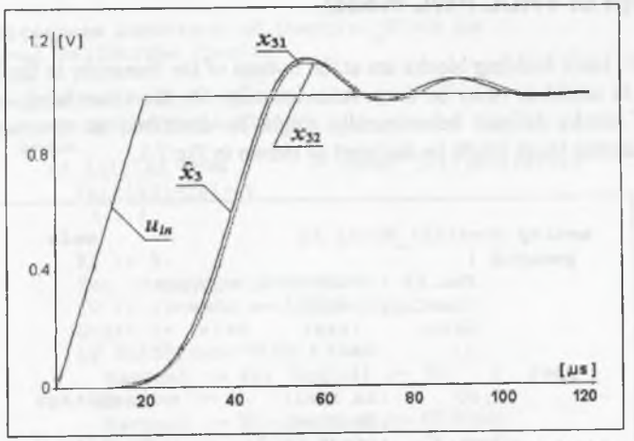

Fig. 6.14. PWL responses x_{31} (feedback-loop model – dotted line) and x_{32} (second order building blocks – dashed line) of 6th order Chebyshev filter for p_{mx} =50mV against exact response $\hat{x}_3$ (solid line) and u_{in}

7. IMPLEMENTATION ISSUES IN VHDL

In this chapter, a description of the implementation of the PWL approach in VHDL is given [ASH90, LIP91]. In fact, the PWL models may be viewed as discrete objects, and in this sense they are well suited to be implemented in a discrete environment. Besides, the explicit formulas that the PWL models are based on make their behavioural representation feasible [DAB98D, DAB98N, DAB99]. As a consequence, the standard VHDL provided with some mathematical and PWL function-packages proved to be sufficient for this purpose. In particular, one can benefit from the digital nature of VHDL, when modelling mixed-signal A/D networks. This kind of modelling can be supported at the expense of additional signal conversions PWL-to-logic and logic-to-PWL, as mentioned in Chapter 4. Apparently, prior to those efforts, some work has already been done, however, in most cases only standard algorithms have been implemented in VHDL [HAR91].

7.1 Concept of VHDL PWL model

Since the PWL basic building blocks are at the bottom of the hierarchy in the functional-level modelling, it is useful to describe them behaviourally. On the other hand, complex models composed of blocks defined behaviourally might be described as structural models. For example; an inertial block might be declared as shown in Fig.7.1.

```
entity Inertial_Block is
 generic (
          Pmx,X0    :real; -- voltage
          TimeConst :real; -- time
          Gain      :real
          );
 port  (
       R,U0      :in real;      -- rate,voltage
       Tend      :in time;
       Rout,X1   :inout real;   -- rate,voltage
       TendOut   :inout time
       );
end Inertial_Block;
```

Fig. 7.1. Inertial block declared as VHDL entity module

The generic constants define parameters of an inertial block, whereas the entries encountered in the port list provide an interface for the *entity* module, and as required in VHDL, are all of class *signal*. One could expect a PWL waveform to be represented adequately as a collection

of pairs (t_k, x_k), like a logic signal i.e., by a single VHDL *signal* declared. Because of the PWL algorithm, the breakpoint (t_{k+1}, x_{k+1}) would be required in advance, i.e. when the actual simulation time equals t_k. Unfortunately, a VHDL simulator is not capable of identifying the next point (t_{k+1}, x_{k+1}) unless the simulation time is advanced to t_{k+1}. Also the time instants t_k, t_{k+1} cannot be accessed directly. To overcome this drawback, two artificial signals more are introduced that provide the VHDL model (at t_k) with the actual PWL segment rate r and the time instant t_{k+1} of the next breakpoint.

The signal names used in the model declaration correspond to notation exploited in previous chapters. `Tend` and `TendOut` denote the following PWL time breakpoints (end of segment) at the input and output, respectively.

The implementation part of the entity `Inertial_Block` is depicted in Fig.7.2. It is a behavioural model arranged as an *architecture body* including a VHDL *process*. The process is activated initially during the initialisation phase of simulation (with `Initial=true`). After executing all the statements, it is suspended. When an *event* occurs on any of the signals from its sensitivity list (i.e. the value of the signal changes), the process is resumed, and execution repeats from the beginning (with `Initial=false`). The PWL input waveform is identified by `R` (representing the slope) and `Tend` (representing the end of segment). If the resulting output segment is shorter than the input one then the additional signal `SelfStrobe` is used to resume the process for the given input segment.

```
architecture behaviour of Inertial_Block is
 signal SelfStrobe :boolean;
 begin
   process(R,Tend,SelfStrobe)
    -- variable declarations(...)
    begin
      if Initial then       -- model initialisation
        Initial:=false;
         (...)
      else
        R1 := R;
        Tmx := time_to_real(Tend);
        T0 := time_to_real(NOW);
        Start := false
        if SelfStrobe'Stable then
          Xactual := X0; Uactual := U0;
         else
          Xactual := X1; Uactual := U1;
        end if
      end if;
      --calculation of time step T1 (...);
      --calculation of Xactual,Uactual (...);
      Tstep := real_to_time(T1);
      X1 <= Xactual after Tstep;    -- update signals
      U1 <= Uactual after Tstep;    -- for next breakpoint
      Rout <= (Xactual-Xlast)/T1;
      TendOut <= (NOW + Tstep) after Tstep;
      if (Tmx > T0+T1) then       --(out segm)<(in segm)
        SelfStrobe <= not(SelfStrobe) after Tstep;
      end if;
    end process;
end behaviour;
```

Fig. 7.2. Architecture body of inertial block model arranged as VHDL process

To assure clarity of the architecture body a few fragments of the VHDL code have been omitted. Nevertheless, some distinctive features relevant to this implementation can be observed. Firstly, the actual values of signals (*transactions*) are assigned to variables to enable further calculations (e.g. `R1:=R`). Special functions `time_to_real` and `real_to_time` are used to obtain `Tmx,T0` and `Tstep` respectively, since VHDL is a strongly-typed language [IEE93], and operations on variables of different type are not allowed. To check whether the process has been activated with the input segment or self-strobed, the attribute `'Stable` of `SelfStrobe` is used. The signals relevant to the output are updated concurrently.

The other building blocks might be implemented in a similar way. By defining them as entities one can easily decompose a complex model, and assure adequate communication between the connected components. In particular, the connected components are executed concurrently when activated with the same signal (because of processes included).

7.2 Structural description

When the entity is used in a design, its generic constants must be specified and the signals connected to module ports. This procedure is referred to as *component instantiation* and must be preceded by *component declaration.* In this case, the declared component can be thought as a template defining a virtual design entity, to be instantiated later within the architecture body. An example of a component declaration referred to the `Inertial_Block` presented above is given in Fig.7.3.

```
component Inertial_Block
 generic (
         Pmx,X0    :real;    -- voltage
         TimeConst :real;    -- time
         Gain      :real
         );
 port  (
       R,U0      :in real;       -- rate,voltage
       Tend      :in time;
       Rout,X1   :inout real;    -- rate,voltage
       TendOut   :inout time
       );
end component;
```

Fig. 7.3. Inertial block declared as VHDL design component

Now, assume the component `Inertial_Block` might be used to model a simple two-stage amplifier with a front-end gain equal 10 and dominant pole 1MHz, and the output stage gain equal 2 with dominant pole 5MHz. In this case, the respective model should be provided with some virtual inputs and outputs corresponding directly to the signals declared in the port list in Fig.7.3. Such a model can be performed as shown in Figs.7.4 and 7.5.

In Fig.7.5 the amplifier is declared as an entity `two_stage_amplifier` defining external ports. The architecture body of the model is of structural type. It comprises a declaration of internal signals (`R1,X1,T1`) and of the component `Inertial_Block` , which is next instantiated to arrange the amplifier stages. The latter are labeled with identifiers: `Front_end` and `Output_stage`. In the both instances generic- and port map specifications are given,

which pertain to model parameters (p_{mx}, gain, time constant, initial condition) and structure connections (provided by signals), respectively. Nanoseconds are assumed as simulation time units.

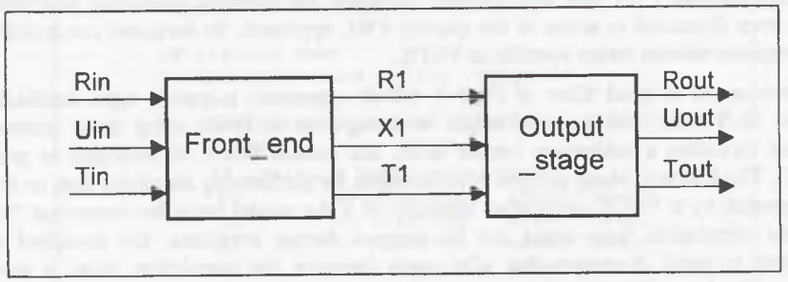

Fig. 7.4. Signal flow between components of two-stage amplifier model

```
entity Two_stage_amplifier is
  port (Rin,Uin     :in real;         -- [V/ns],[V]
        TendIn      :in time;         -- [ns]
        Rout,Uout   :inout real;      -- [V/ns],[V]
        TendOut     :inout time);     -- [ns]
end two_stage_amplifier;

architecture block_structure of Two_stage_amplifier is
 component Inertial_Block
  generic(Pmx,X0   :real;       -- [V]
          TimeConst:real;       -- [ns]
          Gain     :real
          );
  port (R,U0     :in real;      -- [V/ns],[V]
        Tend     :in time;      -- [ns]
        Rout,X1  :inout real;   -- [V/ns],[V]
        TendOut  :inout time    -- [ns]
        );
 end component;
 signal R1,X1 :real;    -- internal signals [V/ns],[V]
 signal T1    :time;    -- [ns]
 begin
   Front_end: Inertial_Block
    generic map (Pmx=>0.05, TimeConst=>159,
          gain=>10, X0=>0);
    port map  (R=>Rin, U0=>Uin, Tend=>TendIn,
          Rout=>R1, X1=>X1, TendOut=>T1);
   Output_stage: Inertial_Block
    generic map (Pmx=>0.05, TimeConst=>32,
          gain=>2, X0=>0);
    port map  (R=>R1, U0=>X1, Tend=>T1,
          Rout=>Rout, X1=>Uout, TendOut=>TendOut);
 end block_structure;
```

Fig. 7.5. VHDL model of two-stage amplifier composed of inertial blocks

7.3 Other implementation issues

In this section we address the VHDL implementation of analogue feedback loops and interfacing between PWL and logic models. Basically, the involved modelling problems have already been discussed in terms of the general PWL approach. So here, we concentrate only on the implementation issues specific to VHDL.

First, invoke the bi-quad filter of Fig.5.3, which represents a typical tight feedback loop structure. In VHDL such a model might be thought as an entity using three components: integrator including a subtractor, inertial block and control block, all arranged as processes (Fig.7.6). The Control_block process is responsible for performing iterations that, in fact, are not supported by a VHDL simulator. Because of it the model must be somewhat "tricky". Since the simulation time could not be stopped during iterations, the so-called *delta's generation* is used. It means that after each iteration the simulation time is advanced automatically by *delta-time* (increment, which depends on a resolution declared in a simulator). As a consequence, the resulting total time increment for a well-posed model, with a limited number of iterations, can be neglected. In this way the PWL segment obtained after the iterations are brought to convergence is correct.

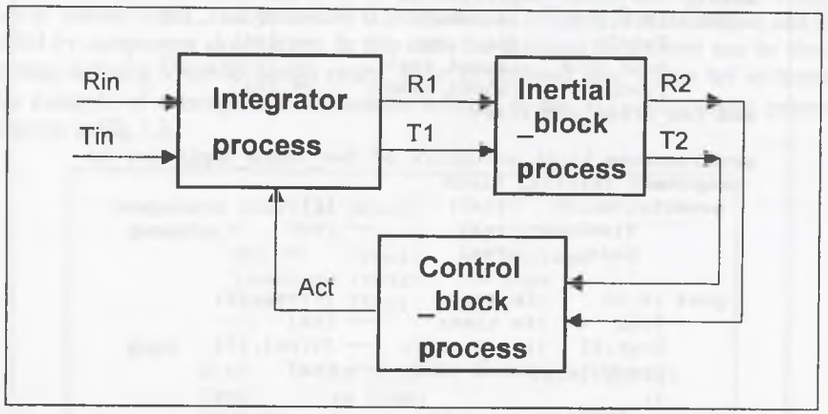

Fig. 7.6. Communication between three processes constituting model of bi-quad filter

Clearly, the simulation algorithm must conform to rules of the VHDL simulator. Using the one-segment relaxation scheme (OSR) seems to be indispensable here, since the VHDL simulator is only capable of processing the event next to the actual simulation time. As explained before, in OSR each iterative cycle proceeds for a single PWL segment, regarded in VHDL as an event. In contrary to this, in WR a multi-segment waveform would be represented by a sequence of upcoming events that cannot be processed jointly by the VHDL simulator with respect to the WR rule.

In Fig.7.7 an architecture body of the `Control_block`, arranged as a process is shown. The process is sensitive to the signals supplied by the `Inertial_block`, and when activated it checks an actual PWL segment for convergence. Unless the assumed accuracy is reached, the `Control_block` activates the process of the `Integrator` component and forces the next iteration (signal Act). Finally, when the end of iterations is detected, it yields correct signals relevant to the filter output, and updates the involved internal signals. When the simulation time advances to (`NOW+Tstep`), the signal `Act` activates the loop (Integrator process) again, so the self-strobing mechanism (shown in Fig.7.2) may be omitted.


```
architecture behaviour of Control_block is
begin
  process(R2,T2)
  -- variables' declaration (...)
  begin
    if Initial then
    -- Initialization of the variables (...)
      else
        Difference  := abs((X2actual - Xlast)/X2v);
        Xlast       := X2actual;
        if (Delta > Difference) then
         X2       <= X2actual   after Tstep;
         T2       <= T2actual   after Tstep;
         X1       <= Xlactual   after Tstep;
         T1       <= Tlactual   after Tstep;
         Act      <= not(Act)   after Tstep;
        else
         X2       <= X2actual;
         T2       <= T2actual;
         X1       <= Xlactual;
         T1       <= Tlactual;
         Act      <= not(Act);
        end if;
    end if;
  end process;
end behaviour;
```

Fig. 7.7. Architecture body of control block arranged as VHDL process

Next, consider the problem of interfacing between the PWL- and the logic domain. The principles of the relevant conversions have been presented in Section 4.2. Following them the VHDL models might be developed as shown below. The Boolean logic is assumed to avoid ambiguity having its origin in unknown logic states, when converting to the PWL domain. Moreover, some unknown states that might be generated by a PWL-to-logic converter during the rising- or falling edge of the signal should be avoided as well. Apparently, an unnecessarily produced unknown state tends to spread through the digital part of A/D system and may appear at the analogue input again. Detailed analysis of such a model is rather difficult.

Instead, a trade-off is recommended. Converters that model inertial delay, rising/falling edges, and provided with two logic levels, seem to be a reasonable solution. From electrical point of view we address here mainly MOS circuits, for which electrical loading (bi-directional coupling) is a second order effect. On the other hand, the accuracy of the functional-level models is usually limited by assumption, and hence too many details should not be expected.

In Fig.7.8 a logic-to-PWL virtual converter is presented. Once a logic event appears at its input, the conversion begins. If the actual PWL segment rises or falls (`Ractual /= 0`), the output waveform is updated for the actual time instant `NOW`. Then, the time step `delta` is calculated to locate the next PWL breakpoint at `Vlow` or `Vhigh`. Finally, all the output signals are updated. However, the last value of `TendOut` cannot be updated, since the simulation time has moved forward. Fortunately, the breakpoint at `TO` precedes the faulty value of `TendOut` and overrides it, when applied to the input of any analogue block. For the same reason the generic parameter `dt` should be large enough to avoid discontinuities in the PWL waveform.

```
entity Logic_to_PWL is
  generic(slope_neg,slope_pos :real;    -- rate
          Vlow,Vhigh          :real;    -- voltage
          dt                  :time
          );
  port (Logic_in   :in  std_logic;
        Rout,Uout  :inout real;         -- rate,voltage
        TendOut    :inout time
        );
end Logic_to_PWL;

architecture behaviour of Logic_to_PWL is
 variable   delta,Unext,Ractual  :real; -- voltage,rate
 variable   Slope,Tact,T0        :real; -- rate,time
 variable   Tstep,Tactual        :time;
 begin
   process(Logic_in)
    begin
      if Initial then
      -- initialisation of variables and signals (...);
      else
        Tactual := TendOut;
        Uactual := Uout;
        Ractual := Rout;
        if (logic_in = '1') then
          Unext := Vhigh;
          Slope := Slope_pos;
          else Unext := Vlow;
          Slope := Slope_neg
        end if;
        if (Ractual /= 0)) then  --update Uactual for T0
          Tact := time_to_real(Tactual);
          T0 := time_to_real(NOW);
          Uactual := Unext + Ractual*(T0-Tact)
        end if;
        delta := (Unext - Uactual)/Slope;
        Tstep := real_to_time(delta);
        Tstep1 := Tstep + dt;
        Uout <= Uactual, Unext after Tstep;
        Rout <= Slope, 0 after Tstep;
        TendOut <= (NOW+Tstep),(NOW+Tstep1) after Tstep;
      end if;
    end process;
end behaviour;
```

Fig. 7.8. VHDL model of logic-to-PWL converter

When instantiated as a VHDL component, the model should reflect the rise- and fall delay time of the driving logic model via its generic parameters. Following the basic definition for the rise delay time one obtains

$$\Delta t_{LH} = \frac{V_{th} - V_L}{r_{pos}} \tag{7.1}$$

and for the fall delay

$$\Delta t_{HL} = \frac{V_{th} - V_H}{r_{neg}} \tag{7.2}$$

Based on the waveforms shown in Fig.7.9, observe how the converter works. When the logic event occurs at t_1, the PWL breakpoint at this time instant is not defined yet, so it is added to the waveform using the actual value of `Uout`, since `Ractual=0` at t_0. Unlike this, the value used for the breakpoint to be added at t_4 requires updating (`Ractual/=0`). The value of `Unext` that has been projected at t_3 is increased by (`Ractual*(T0-Tact)`), where `T0` = t_4, `Tact` = t_5 and `Ractual` is the segment rate at t_3. Apparently, t_5 should be replaced by t_4, but the simulator does not allow this.

Note that the converter is capable of damping all short logic pulses applied to its input.

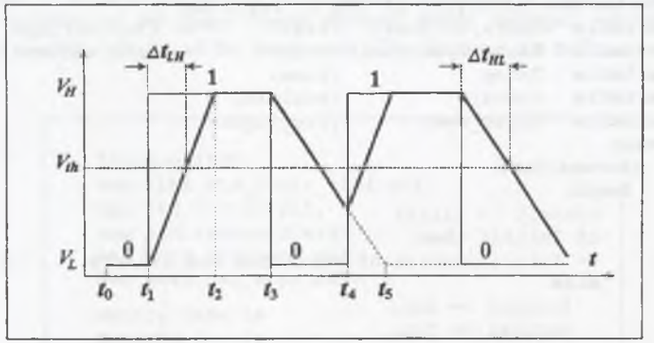

Fig. 7.9. Inertial delay performed in logic-to-PWL conversion

As mentioned in Section 4.2, the PWL-to-logic converter is simpler. It might be implemented as a VHDL entity including a process, too (Fig.7.11). In this case, the PWL signals drive the logic output. The process used is sensitive to the input segment rate, and it proceeds when the PWL waveform can be expected to cross the logic threshold, as shown in Fig.7.10. After computing the amplitude of the end point, the relevant segment is checked for crossing the threshold V_{th}. Next, when applicable the respective delay `delta` is calculated, and a transaction on the logic output signal is scheduled.

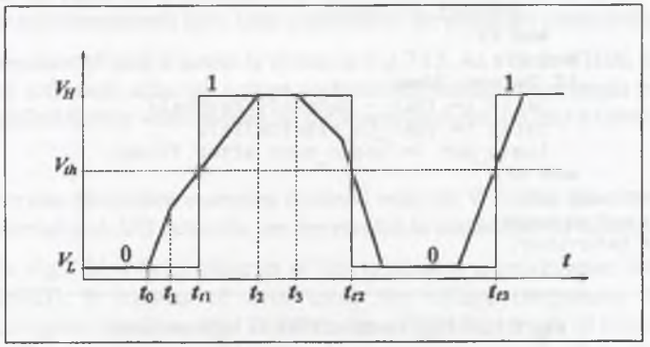

Fig. 7.10. Logic transactions scheduled for t_{t1}, t_{t2}, t_{t3} during PWL to logic conversion

For the waveform shown in Fig.7.10 the first two segments are not capable of projecting a transaction at logic output. The process finds the segment starting at t_1 to cross the logic threshold. Hence, the first logic transaction is scheduled to occur at t_{t1}.

```
entity PWL_to_logic is
  generic(Vth    :real);          -- voltage
  port (Rin,Uin    :in real;      -- rate,voltage
        TendIn     :in time
        Logic_out  :out std_logic;
        );
end PWL_to_logic;

architecture behaviour of PWL_to_logic is
 variable   delta,T0,Unext   :real;      -- time,voltage
 variable   Ractual,Uactual  :real;      -- rate,voltage
 variable   Tstep            :time;
 variable   Convert          :boolean;
 variable   Logic_next       :std_logic;
 begin
   process(Rin)
    begin
      Convert := false;
      if Initial then
      -- initialisation of variables and signals (...);
      else
        Ractual := Rin;
        Uactual := Uin;
        Tactual := time_to_real(TendIn);
        T0 := time_to_real(NOW);
        if ((Uactual < Vth) and (Ractual > 0)) then
          Unext := Uactual + Ractual*(Tactual - T0);
          if Unext > Vth then
            Logic_next := '1';
            Convert := true;
          end if;
        end if;
        if ((Uactual > Vth) and (Ractual < 0)) then
          Unext := Uactual + Ractual*(Tactual - T0);
          if Vth > Unext then
            Logic_next := '0'
            Convert := true;
          end if;
        end if;
        if Convert then
          delta := (Vth - Uactual)/Ractual;
          Tstep := real_to_time(delta);
          Logic_out <= Logic_next after Tstep;
        end if;
      end if;
    end process;
end behaviour;
```

Fig. 7.11. VHDL model of PWL-to logic converter

7.4 Simulation examples

To be compiled successfully, the model units need support by *packages* providing all the required predefined data types, subprograms, functions, etc. These are in particular the analogue package `analogpg`, defining analogue operations, and the PWL-approximation package `pwl_appr`. Both are put into the `work` library. Besides, the standard library called `std` with two packages `standard` and `textio`, and the library `IEEE` are needed.

As a consequence, a typical unit described in this chapter, allowing logic and/or PWL operations might be declared completely using a scheme depicted in Fig.7.12. The clause **`use`** provides the visibility of the items' names declared inside packages, whereas the clause **`library`** - the needed visibility of the name `IEEE`. The libraries `std` and `work` are visible by default.

```
library IEEE;
use IEEE.std_logic_1164.all;
use std.textio.all;
use std.standard.all;
use work.analogpg.all;
use work.pwl_appr.all;

entity name is
generic (...);
port (...);
end name;
```

Fig. 7.12. Complete declaration scheme of VHDL unit supporting PWL/logic mixed-mode modelling

A complex structure might be partitioned using entities, which are referenced and instantiated within a higher-level entity. Basically, the lower-level entities describe behaviour (of inertial block, integrator, logic gates etc.), whereas the higher-level entities describe structure. The entity used as an upper-level module with architecture composed of lower-level modules declared as its sub-components have been exploited in the examples presented below.

The overall structure of such a model is shown in Fig.7.13. As a basic VHDL implementation it proved to be sufficient, although a more sophisticated configuration might be used as well. It can be simplified further when putting all the lower-level entities into a library package.

Below we give two simulation examples obtained with the V-System simulator [VSY94]. In both cases mixed-signal A/D networks are represented to emphasise the flexibility of VHDL.

Example 1. In Fig.7.14 a block diagram of the successive approximation A/D converter is shown [DAB98D]. It consists of three units: the voltage comparator, the successive approximation register (SAR) and the D/A converter. Eight-bit version of this model has been implemented. The comparator model is based on two-stage cascade of inertial blocks. This structure is capable to mimic adequately the comparator timing specifications including the influence of initial polarisation and overdrive at its input (as discussed in Section 3.1). The D/A converter makes also use of the inertial block at the output. Its control part follows the fundamental formula $U_{out} = \Sigma\, 2^i\, a_i \cdot \Delta U$, where i = **0..7** and ΔU represents the resolution (here 16mV). The a_i parameters are set either to 0 or to 1 with respect to the digital input of this

```
-- full description of lower-level entities
  ( ... );
  ...
-- declaration of upper-level entity
--( ... );                          -- referencing library packeges
  entity upper is
    generic (...);
    port (...);
  end upper;

  architecture structure of upper is
  ( ... );                -- variables and internal signals declaration
  -- sub-components' declaration referred to lower-level entities
    component module_1
      ( ... );            -- generic and port lists
    end component;

      ...                 -- declarations of remaining components
  begin
      M1:  module_1
      ( ... );            -- generic and port map
      M2:  module_2
      ( ... );            -- generic and port map

      ...                 -- instantiation of remaining components
  end structure;
```

Fig. 7.13. Overall structure of VHDL models used for simulation examples

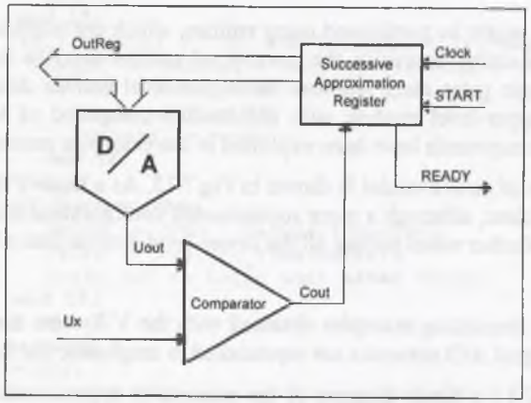

Fig. 7.14. Functional-level block diagram of successive approximation A/D converter

unit. Since this operation is physically involved with switching of signals inside the converter, any change of the a_i coefficients is controlled by their own drivers, which are modelled as digital objects with a prescribed inertial delay. In this way the D/A unit is defined as a mixed-signal entity. On the other hand, the SAR consists exclusively of logic components, and its behaviour is defined at RT-level of abstraction.

Besides, the comparator- and the D/A unit are provided with the PWL-to-logic and logic-to-PWL virtual converters, accordingly, to assure interfacing between the std_logic- and the PWL domain. The complete model is a nested entity structure, which comprises 430 lines of VHDL code, excluding the packages `analogpg` and `pwl_appr`.

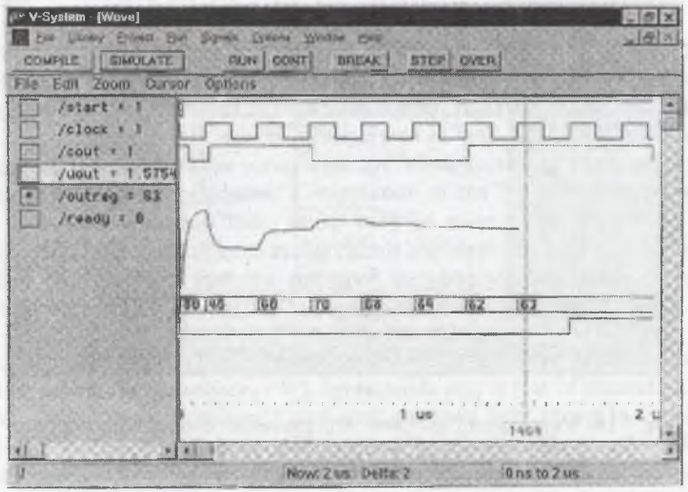

Fig. 7.15. Waveforms SAR A/D converter obtained with V-System

Some of the timing waveforms obtained for the A/D converter model are shown in Fig.7.15. Constant input signal Ux is assumed. The converter is reset when Start = 0. Then the successive approximation proceeds following the active signal Clock.

The signal names on the left are given in the order corresponding to the waveforms plotted. The simulation process depends heavily on the clock signal, which when inactive simply breaks the existing feedback loop. As discussed in Chapter 5, no iterations are required in that case.

Example 2. This example invokes the half-flash A/D converter presented in detail in Section 4.3. Like in Example 1, the VHDL model of this converter constitutes a nested entity structure. The top-level entity `Half_flash` consists of eight components representing the functional units of the converter (Fig.4.7): `Track_hold`, `Control_unit`, `Differential_amplifier`, `Analog_multiplexer`, `DA_converter`, `AD_flash`, `Register_1` and `Tri_state_register`. The required virtual converters providing interface between the PWL- and the std_logic domain are their sub-components, like the `Inertial_block`. The VHDL model of `Half_flash` includes 1040 lines of code without the packages `analogpg` and `pwl_appr`.

Samples of the waveforms obtained with the V-System simulator are given in Figs. 7.16 and 7.17. The names of PWL signals U1, U2, U3, U4, depicted on the left, correspond to the outputs of track-hold, D/A converter, differential amplifier and multiplexer, respectively. The input PWL stimulus is denoted as b1/xpwl.

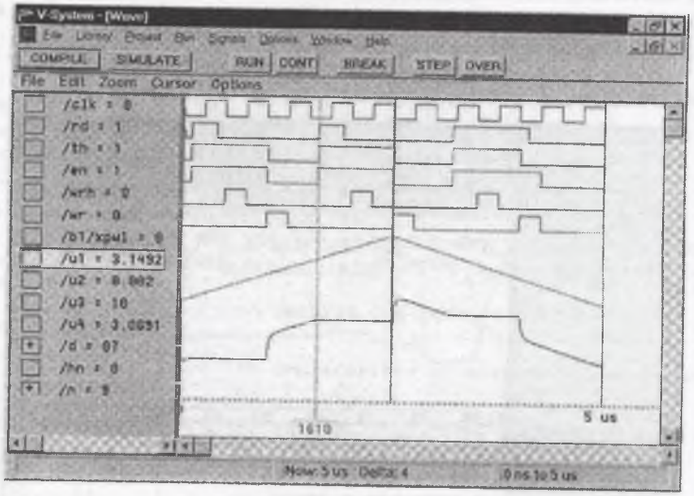

Fig. 7.16. Waveforms of half-flash A/D converter obtained with V-System

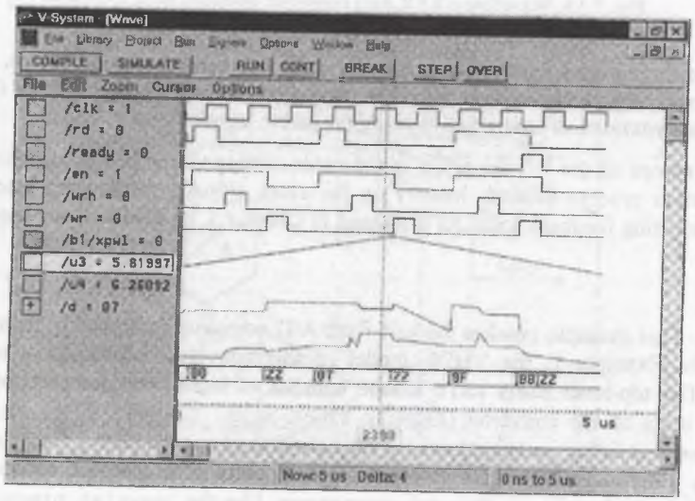

Fig. 7.17. Waveforms of half-flash A/D converter obtained with V-System

As compared to the prototype PWL simulator, presented in Chapter 4, the V-System performs by 30% slower. In the main context, however, it does not seem to play the VHDL implementation down. In fact, the VHDL implementation may be viewed as an effort of putting the PWL macrosimulation technique into a broader perspective.

8. SUMMARY

A macrosimulation technique for the modelling of analogue networks and mixed-signal A/D systems at the functional level has been presented. MOS circuits have been addressed mainly. Their component units are assumed to be unidirectional, however, capacitive loading effects are allowed. Basic nonlinearities may be incorporated as well. The signals of analogue units are represented as piecewise linear waveforms and when applied to the models they closely approximate the real timing behaviour. Computation of the PWL time responses has been formulated as an approximation task, which may be solved efficiently with no need of iterations that all standard approximation algorithms are based on. As an alternative approach the trapezoidal rule provided with the enhanced step control mechanism is introduced. It appears particularly useful when applied to cascade structures of analogue blocks.

Some of the macromodels have been implemented within a prototype event-driven functional-level simulator, where the subsequent PWL breakpoints play a role of simulation events. The obtained PWL waveforms of commonly used analogue units were shown to closely match the respective SPICE estimates with a simulation speed-up of two to three orders of magnitude. However, as mentioned earlier, no claim is made regarding the PWL models suitability to perfectly mimic the real circuit behaviour for all situations.

Some drawbacks arising for analogue feedback loops and cascade structures are evident. In this case the PWL approximation errors tend to accumulate temporarily, but they can be reduced in a few ways. First, the enhanced PWL approximator allows compensating for those errors to some extent at the expense of extra computations. Second, the analogue loops may be avoided if a designer is not interested in having insight into the loop. For example, complex analogue filters can be synthesised using cascade or parallel structures composed of second and/or first order building blocks. The ordering of sections in a cascade due to their filtering properties, and proper gain assignment influence substantially the resulting global PWL errors as well. In case of cascade structures, however, the trapezoidal rule-based models prove to perform better than their approximation-based counterpart.

On the other hand, when dealing with mixed A/D networks, a mixed mode simulation technique is preferred. For digital units usually behavioural logic models are used. As a consequence, these models require logic-to-PWL and PWL-to-logic signal converters. The possible ambiguity having its origin in unknown logical states may be avoided by means of relatively simple converters' models. In this way, the fully unified PWL treatment of the A/D network, introduced in [RUA91], is no longer applied here. Nevertheless, a distinctive feature that the PWL and logic models have in common still remains. In fact, all those models may be viewed as discrete objects, so that their implementation in a discrete HDL environment is feasible.

The VHDL implementation was expected to put the PWL approach into a broader perspective. Several models have been successfully developed in VHDL as discrete objects. Explicit formulas available for PWL timing enable defining the basic VHDL entities as behavioural models, for which the PWL breakpoints are referred to as simulation events. The models are supported by extra analogue and PWL approximation packages. The procedures and functions used may be viewed as a kind of behavioural decomposition, whereas the VHDL entities correspond to structural decomposition in this modelling. Besides, one can benefit from the digital nature of VHDL when modelling mixed-signal A/D systems. Although the new extension to VHDL, the VHDL-AMS [BER95, VAC97], oriented towards analogue- and mixed systems seemed to be better suited to PWL simulation, it has not been used because of lack of AMS tools during the course of preparing this work.

Since the presented macrosimulation technique saves the CPU time, it is also well suited to work within iterative processes. This is of particular interest, when dealing with non-clocked systems featuring global feedback loops. In this case the waveform relaxation has been shown to suit well to the PWL macrosimulation. Because local couplings are not used in the PWL modelling, only global feedback loops are responsible for the occurrence of iterations. Despite the slow convergence of the WR in some cases, the CPU simulation time remains moderate, since the PWL algorithm is very time effective. However, for too large segment length it can suffer from non-convergence or instability. The derived convergence conditions enable to evaluate the maximum stable segment length for linear analogue feedback loops. For other feedback structures direct segment length reduction is useful if too many iterations occur. The properly used windowing technique is shown to be essential for the WR process since the accumulation of WR error may be limited in time. In some situations one-segment relaxation (OSR) appears to be more efficient than WR. In case of the switching (or clocked) loops the number of required iterations may be substantially reduced, and even a single iteration is possible. Moreover, as compared to SPICE, the speed-up obtained for analogue loops with the WR-PWL simulation is up to one hundred.

Apparently, a variety of issues relevant to the PWL approach to macrosimulation of analogue and mixed A/D systems have been addressed in this monograph. The presented results validate this method over a wide range of applications. In this context the PWL approach proves to be efficient. Finally, also other applications of the PWL technique supported by the waveform relaxation begin to emerge. These are for example regular structures such as RC-trees [KON95, DAB00] used for interconnect timing verification of VHDL designs, or cellular neural networks [ROS95]. In both cases the respective models might be decomposed into a set of inertial blocks, and analysed with explicit formulas relevant to the PWL techniques.

REFERENCES

[ACU89] Acuna E. et al., *iSPLICE3: A New Simulator for Mixed Analog/Digital Circuits*, Proc.of CICC, May 1989, pp. 1311-1314.

[ACU90] Acuna E. et al., *Simulation Techniques for Mixed Analog/Digital Circuits,* IEEE J. Solid-State Circuits, Vol.25, Apr.1990, pp. 353-362.

[ANA94] ANACAD, *HDL-A Language Reference Manual,* 1994.

[ANT92] Antao B., Brodersen A., *Behavioral Simulation for Analog System Design Verification,* IEEE Trans. VLSI Systems, Vol.3, May 1992, pp. 417-429.

[ANT96] Antao B., El-Turky F., Leonowich R.: *Behavioral Modeling Phase-locked Loops for Mixed-Mode Simulation,* Analogue Integrated Circuits and Signal Processing, Kluwer Academic Pub. vol.10, No 1/2, June 1996, pp. 45-65.

[ANG81] Antognetti P., Pederson D.O., De Man, H., (Eds.):*Computer Design Aids for VLSI Circuits*, NATO Adv.Study, Noordhoff & Sijithoff Gronningen 1981

[ARM88] Armstrong J.R., *Chip Level Modeling*, Prentice Hall 1988.

[ARN78] Arnout G., De Man H., *The Use of the Threshold Functions and Boolean-Controlled Network Elements for Macromodeling of LSI Circuits*, IEEE J. Solid St.Cir., Vol.SC-13, No.3, June 1978, pp. 326-332.

[ASH90] Ashenden P., *The VHDL Cookbook,* Dept.Computer Science, Univ.of Adelaide, South Australia, 1990.

[BAR81] Barbacci M.R., *Instruction Set Processor Specification (ISPS): The notation and its applications*, IEEE Trans.C-30, Jan.1981, pp. 24-40.

[BER95] Berge J.M., Elvia O., Rouillard J., *Modeling in Analog Design*, Kluwer Acad. Pub. 1995.

[BIA79] Białko M. (Ed.), *RC Active Filters* (in Polish), WNT, Warszawa, 1979.

[BLO87] Bloom M.: *Mixed-mode simulators bridge the gap between analog and digital design,* Computer Design, Jan.1987, pp. 51-65.

[BLA98] Błachut J., *Private communication,* Silesian University of Technology, Gliwice, 1998.

[BOK87] Bokhoven W.M.G., *Piecewise Linear Analysis and Simulation,* in Circuit Analysis, Simulation and Design, Part 2, A.E.Ruehli K., (Ed.), Elsevier Science Pub. 1987.

[BOY74] Boyle G. et al., *Macromodeling of Integrated Circuit Operational Amplifiers,* IEEE J. Solid-State Circuits, Vol.SC-9, Dec.1974, pp. 353-364.

[BRE72] Breuer M.A., *Recent Developements in the Design and Analysis of Digital Systems,* Proc.IEEE, vol.60, No.1, Jan.1972, pp. 12-27.

[BRE75] Breuer M.A, (Ed.), *Digital Design System Automation: Languages, Simulation and Data Base*, Computer Scince Press, 1975.

[BRE76]. Breuer M.A, Friedman A.D., *Diagnosis and Reliable Design of Digital Systems*, Pitman Pub.Ltd. 1976, Computer Science Press Inc.

[BRO88] Brocco L.M. et al., *Macromodeling CMOS Circuits for Timing Simulation,* IEEE Trans. Computer-Aided Design, Vol.7, No.12, Dec.1988, pp. 1237-1249.

[BRY84] Bryant R.E., *A Switch-Level Model and Simulator for MOS Digital Systems*, Trans. IEEE, Vol.C-33, Feb.1984.

[BRY87] Bryant R.E., *A Survey of Switch-Level Algorithms*, IEEE Design and Test, Aug.1987 pp. 26-38.

[BUR89] *Burr-Brown General Catalog*, Tucson, Arizona, 1989.

[BUK91] Bukat D.,Ogrodzki J., *Enhanced Network Description Language for Behavioral Circuit Description Simulators Including External Models*, Proc.of 14th KKTOiUE, Waplewo, Poland, Oct.1991, pp. 160-165.

[CAS91] Casinovi G., Sangiovanni-Vincentelli A.L., *A Macromodeling Algorithm for Analog Circuits*, IEEE Trans.on CAD, vol.CAD-10, No.2, Feb.1991.

[CHA92] Chadha R., Visweswariah Ch.: Chen Ch., *M^3 - A Multilevel Mixed-Mode Mixed A/D Simulator,* IEEE Trans.on CAD, Vol.CAD-11,No.5, May 1992, pp. 575-585.

[CHN88] Chang F.C., Chen C.F., Subramaniam P., An Accurate and Efficient Gate Level Delay Calculator for MOS Circuits, Proc.of 25th DAC, 1988.

[CHW75] Chawla B.R., Gummel H.K., Kozak P., *MOTIS - An MOS Timing Simulator*, IEEE Trans. Vol.CAS-22, 1975 pp. 901-909.

[CHE66] Cheney E.W., *Introduction to Approximation Theory,* McGraw Hill, London, 1966

[CHU80] Chua L.O., *Device Modeling Via Basic Nonlinear Circuit Elements*, IEEE Trans.CAS, Vol.CAS-27, No.11, Nov.1980, pp. 1014-1044.

[CHU86] Chua L.O., Deng A., *Canonical Piecewise Linear Modeling,* IEEE Trans. Circ. and Syst., Vol.CAS-33, May 1986, pp. 511-525.

[CHU75] Chua L.O., Lin P., *"Computer-Aided Analysis of Electronic Circuits"*, Englewood Cliffs, N.Y.:Prentice Hall, 1975.

[COT90] Cottrell R.A., *Mixed Analogue-Digital Simulation of ASICs using Transfer Function Models,* Journal of Semicustom Ics, Vol.7, No,4, Elsevier Science Pub., 1990, pp.21-25.

[DAB88] Dąbrowski J., *Design of Multilevel Mixed-Mode Simulator for LSI/VLSI Circuits*, Proc.of ISCAS'88, Helsinki, Finland, 1988, pp. 1685-1688.

[DAB89] Dąbrowski J., *Mixed-Mode Timing Verification for VLSI Designs*, AMSE Press, Vol.4, 1989, pp. 129-138.

[DAB90] Dąbrowski J., *LOGSIM1: Gate-level logic Simulator,* (in Polish), Proc. of 13th National Conf. TOiUE Bielsko-Biała 1990, pp. 259-264.

[DAB91E] Dąbrowski J., Konopacki J., *Experience with Block Waveform Relaxation for Analog Networks*, Proc.of 14th KKTOiUE, Waplewo, Poland, Oct.1991, pp. 145-150.

[DAB91S] Dąbrowski J.,.Konopacki J., *Hybrid Simulator for Complex A/D Circuits,* (in Polish), Raport Instytutu Elektroniki, Pol.Śl. 1991, (grant MEN nr DNS-T/05/054/90-2).

[DAB92] Dąbrowski J., *Efficient Timing Verification via Mixed-Mode Technique*, Microprocessing and Microprogramming 34, North-Holland, 1992, pp. 183-186.

[DAB92K] Dąbrowski J., Konopacki J., *HYBRID: A Mixed-Mode Simulator for A/D Networks,* Proc.of 15th KKTOiUE, Szczyrk, Poland, Oct.1992, pp.103-108.

[DAB93] Dąbrowski J., Konopacki J., *Delay Models for Mixed-Mode Timing Verification of Bipolar Circuits,* Proc.of 16th KKTOiUE, Kołobrzeg, Poland, Oct.1993, pp.193-198.

[DAB94] Dąbrowski J., Konopacki J., *A/D Networks Functional-Level Macromodeling with Piecewise Linear Signals,* Proc.of 17th National Conf. TOiUE, Polanica, Poland, Oct.1994, pp. 353-358.

[DAB95] Dąbrowski J., *Functional-Level Analog Macromodeling with Piecewise Linear Signals,* Proc.of EURO-DAC'95, Brighton, England, 1995, pp.222-227.

[DAB95K] Dąbrowski J., Konopacki J., *Implementation of Analog Macromodels in PWL Functional-Level Simulator,* Proc.of 18th National Conf. TOiUE, Zakopane, Poland, Oct.1995, pp. 251-256.

[DAB96] Dąbrowski J., *Algorithms for Hybrid Simulation,* (in Polish), Zeszyty Naukowe Politechniki Śl., Seria ELEKTRONIKA z.5, Gliwice, 1996, pp. 37-81.

[DAB96K] Dąbrowski J., Konopacki J., *Implementation of A/D Network Macromodels in PWL Functional-Level Simulator*, Bull.of the Polish Academy of Sciences, Technical Sciences, Vol.44, No.3, 1996, pp. 293-312.

[DAB96P] Dąbrowski J., Pułka A., *PWL-Based A/D Networks Macromodeling in Discrete VHDL Environment,* Proc.of 19th KK TOiUE, Krynica, Poland, 1996, pp. 221-226.

[DAB97E] Dąbrowski J., *Experience with Waveform Relaxation for Functional-Level PWL Simulation,* Proc.of 19th National Conf. TOiUE, Poland, 1997.

[DAB97W]Dąbrowski J., *Waveform Relaxation Approach to PWL Simulation of Analog and Mixed A/D Networks at the Functional Level*, Proc.of ECCTD'97, Budapest, Hungary, Sept.1997, pp. 507-512.

[DAB98D] Dąbrowski J., Pułka A., *Discrete Approach to PWL Analog Modeling in VHDL Environment*, Analogue Integrated Circuits and Signal Processing, Kluwer Academic Pub. Vol.16, No 2, June 1998, pp. 91-99.

[DAB98E] Dąbrowski J., Pułka A., *Efficient Modeling of Analog & Mixed A/D Systems via Piecewise Linear Technique*, Proc.of FDL'98, Lausanne, Switzerland,1998, pp. 295-304.

[DAB98N] Dąbrowski J., Pułka A., *Network Designing with Programmable Devices,* Proc.of 21st National Conf. TOiUE, Poznań-Kiekrz, Poland, Oct.1998, pp.:149-154.

[DAB99] Dąbrowski J., Pułka A., *Experiences with Modeling of Analog and Mixed A/D Systems Based on PWL Technique,* Proc.of DATE'99, Munich, Germany, March 1999, pp. 790-791.

[DAB99F] Dąbrowski J., *Functional-level analogue macromodelling with piecewise linear signals,* IEE Proc. Circuits, Devices and Systems, UK, vol.146, No.2, April 1999, pp. 77-82.

[DAB99S] Dąbrowski J., *Synthesis of Functional-Level Analog Models with PWL Technique,* Proc. of MIXDES'99, Kraków, Poland, June 1999, pp. 243-246.

[DAB99W] Dąbrowski J., *Waveform Relaxation Approach to PWL Simulation of Analog and Mixed A/D Networks at the Functional Level*, accepted to IEE Proc. Circuits, Devices and Systems, UK.

[DAB00] Dąbrowski J., *Efficient Interconnect Timing Analysis via Piecewise Linear Technique,* Proc. of ISCAS-2000, Geneva, Switzerland, 2000.

[DEB87] Debefve P., Odeh F., Ruehli A.E., *Waveform Techniques* in [RUE86]

[DEM81] De Man H. et.al., *Mixed-Mode Simulation Techniques and Their Implementation in Diana* in [ANG81] pp. 113-174.

[DEM87] De Micheli G. et al., *Decomposition Techniques for Large Scale Circuit Analysis and Simulation,* Chpt.7 in [RUE86]

[DRY85] Drygajło A., Dąbrowski J., *Circuit Macromodels of TTL ICs*, [in Polish], Rozprawy Elektrotechniczne, vol.31, nr 3-4, 1985, 817-834.

[FAN78] Fan S.P., Hssueh M.Y., Newton A.R., Pederson D.O., *MOTIS-C: a New Circuit Simulator for MOS LSI Circuits,* Proc.ISCAS New York 1978, pp. 700-703.

[FIL90B] Filseth E.S., Berwick J., Man H.: *Analogue Macro Modelling for System Simulation*, Electronic Engineering, Sept.1990, pp. 75-82.

[FIL90R] Filseth E.S., Roullier T., *Build Analog Behavioral Models in Six Easy Steps,* Electronic Design, Nov.22nd 1990, pp. 105-119.

[GIE92] Gielen G., Liu E., Sangiovanni-Vincentelli A.L., Gray P., *Analog Behavioral Models for Simulation and Synthesis of Mixed-Signal Systems,* Proc.of EDAC'92, 1992, pp. 464-468.

[GRI92] Griffith R., Nakhla M.S., *Mixed Frequency/Time Domain Analysis of of Nonlinear Circuits,* IEEE Trans. on CAD, Vol.11, No.8, Aug.1992, pp. 1032-1043.

[HAJ80] Hajj I.N., *Sparsity Considerations on in Network Solution by Tearing,* IEEE Trans.Circuits Syst. Vol.CAS-27, No.5, May 1980, pp. 357-366.

[HAR91] Harr R.E., Stanculescu A.G.: (Eds.), Applications of VHDL to Circuit Design, (Chapter 3,4), Kluwer Acad. Pub. 1991.

[HAT81] Hatchel G.D., Sangiovanni-Vincentelli A.L., *A Survey of Third-Generation Simulation Techniques*, Proc.IEEE, Vol.69, No.10, Oct.1981, pp. 1264-1280.

[HAY82] Hayes J.P., *A Unified Switching Theory with Applications to VLSI Design*, Proc.IEEE, Oct.1982, pp. 1140-1151.

[HEN85] Hennion B., Senn P., *A New Algorithm for Third Generation Circuit Simulators: the One-Step Relaxation Method,* Proc.of 22nd DAC, Las Vegas, 1985.

[HEY82] Heydemann M.E., *Implementation Issues in Multiple Delay Switch Level Simulation*, Proc.of ICCC, 1982, pp.46-49.

[HWA86] Hwang S.H., et al., *An Accurate Delay Modeling Technique for Switch-Level Timing Verification*, Proc.of 23rd DAC, 1986, pp. 227-233.

[IEE93] *IEEE Standard VHDL Language Reference Manual*, IEEE Std 1076-1993, IEEE Standards.

[IEE98] *IEEE Standard VHDL-AMS Language Reference Manual,* IEEE Std 1076.1-1998, http://www.vhdl..org/vi/analog

[JOU83] Joupii N.P., *Timing Analysis for nMOS VLSI*, Proc.of 20th DAC, 1983, pp.411-418.

[KAO94] Kao R., Horowitz M., *Timing Analysis for Piecewise Linear Rsim,* IEEE Trans. on CAD, Vol.13, No.12, Dec.1994, pp. 1498-1512.

[KEV91] Kevenaar T.A.M., Leenaerts D.M.W., *A Flexible Hierarachical Piecewise Linear Simulator,* Integration – the VLSI Journal, No.12, Elsevier Science Pub, 1991, pp. 211-235.

[KIM88] Kim Y.H., Kleckner K.E., Saleh R.A., Newton A.R., *Electrical-Logic Simulation and its Applications*, IEEE Trans. Vol.CAD-8, Jan.1988, pp. 8-22.

[KON95] Kong J.T., Overhauser D., *Digital Timing Macromodeling for VLSI Design Verification,* Kluwer Acad. Pub. 1995.

[KRU96] Kruiskamp W., Leenaerts D., *Behavioral and Macro Modeling using Piecewise Linear Techniques,* Analogue Integrated Circuits and Signal Processing, Kluwer Academic Pub. Vol.10, No 1/2, June 1996, pp. 67-76.

[LEE98] Lee E., Sangiovanni-Vincentelli A.: *A Framework for Comparing Models of Computation,* IEEE Trans CAD Int. C&S, Vol.17, No.12, Dec.1998, pp. 1217-1229.

[LEL82] Lelarasmee E. et al., *The Waveform Relaxation Method for the Time Domain Analysis of Large Scale Nonlinear Dynamical Systems,* IEEE Trans.CAD, Vol.1, No.3, July 1982, pp. 131-145.

[LIP91] Lipsett R., Schaefer C., Ussery C., *VHDL: Hardware Description and Design,* Kluwer Acad.Pub., 1991.

[MAL94] Maliniak L., *A/D Simulator an Expanding Array of Choises*, Electronic Design, Vol.42, No.25, Dec.1994.

[MAN94] Mantooth A., *Modeling with an Analog Hardware Description Language*, Kluwer Acad. Pub. 1994.

[MCC88] McCalla W.J., *Fundamentals of Computer-Aided Circuit Simulation,* Kluwer Academic Pub., 1988.

[MEA80] Mead C.A., Conway L.A., *Introduction to VLSI Systems*, Addison-Wesley Pub,Comp. 1980.

[MER93] Mermet J.P., Ed.: *Fundamentals and Standarts in Hardware Description Languages*, Kluwer Academic Pub., 1993.

[NAG75] Nagel L.W., *SPICE2: A Computer Program to Simulate Semiconductor Circuits*, Memo.No. ERL-M520. Electr.Research Labs., Univ.of California, Berkeley 1975.

[NEW79] Newton A.R., *Techniques for the Simulation of Large-Scale Integrated Circuits,* IEEE Trans. Circuits Syst., Vol.CAS-26, No.9, Sept.1979, pp. 741-749.

[NEW81] Newton A.R., *Timing, Logic and Mixed-Mode Simulation for Large MOS IC* in [ANG81] pp. 175-240.

[NEW84] Newton A.R., Sangiovanni-Vincentelli A.L., *Relaxation-Based Electrical Simulation*, IEEE Trans.Electron Dev., Vol.ED-30, No.9, Sept.1983, pp. 1184-1206.

[ODE90] Odent P., Claesen L., De Man H., *A combined Waveform Relaxation - Waveform Relaxation Newton algorithm for efficient parallel circuit simulation,* Proc.of EDAC'90, 1990, pp. 244-248.

[OGR94] Ogrodzki.J., *Computer-Aided Analysis of Electronic Circuits,* (in Polish) PWN, Warszawa 1994.

[ORT70] Ortega J.M., Rheinbold W.C., *Iterative Solution of Nonlinear Equations in Several Variables*, New York, Academic, 1970.

[OUS83] Ousterhaut J.K., *CRYSTAL: A Timing Analyzer for nMOS VLSI Circuits*, Proc.of 3rd Caltech VLSI Conf., 1983, pp. 57-70.

[PAW85] Pawlak A., *A Tutorial Guide to Modern Hardware Description and Design Languages* in "Microcomputers, Usage & Design" K.Waldschmidt, B.Myrhraug (Eds.), Elsevier Science Pub., EuroMicro 1985.

[PED84] Pederson D.O., *A Historical Review of Circuit Simulation*, IEEE Trans. Vol.CAS-31, No.1, Jan.1984, pp. 103-111.

[PIL90] Pillage L.T., Rohrer R.A., *Asymptotic Waveform Evaluation for Timing Analysis*, IEEE Trans. Vol.CAD-9, No.4, Apr.1990, pp. 352-366.

[RAB79] Rabbat N.B. et al., *A Multilevel Newton Algorithm with Macromodeling and Latency for the Analysis of Large-Scale Nonlinear Networks in the Time Domain*, IEEE Trans. Vol.CAS-26, No.9, Sept.1979, pp. 733-741.

[RAG89] Raghuram R., *Computer Simulation of Electronic Circuits,* John Wiley & Sons, New York, 1989.

[RAL71] Ralston A., *Introduction to Numerical Analysis,* (in Polish), PWN, Warszawa 1971.

[RAO89] Rao V., Overhauser D., Hajj I., Trick T., *Switch-Level Timing Simulation of MOS VLSI Circuits,* Kluwer Academic Pub., Boston, MA., 1989.

[RAO82] Rao V.B., Trick T.N., *Accurate Multiple Delay Calculations for MOS Circuits*, Proc. of ICCC, 1982, pp. 439-442.

[RAO83] Rao V.B., Trick T.N., Hajj I.N., *A Table Driven Delay-Operator Approach to Timing Simulation of MOS VLSI Circuits*, Proc. of ICCD, 1983, pp. 445-448.

[ROS93] Roska T., Chua L.O., *The CNN Paradigm,* IEEE Trans.Circuits Systems - I, Special Issue on CNN, Vol.40, 1993, pp. 147-156.

[RUA91] Ruan G., Vlach J., Barby J., Opal A., *Analog Functional Simulator for Multilevel Systems*, IEEE Trans.on CAD, Vol.10, No.5, May 1991, pp. 565-575.

[RUB81] Rubinstein J. et al., *Signal Delay in RC Tree Networks*, IEEE Trans. Vol.CAD-2, No.3, July 1981, pp. 202-211.

[RUE78] Ruehli A.E., Rabbat R.B., Hsieh H.Y., *Macromodeling – An Approach for Analysing Large-scale Circuits,* Computer-Aided Design, Vol.10, No.2, Mar.1978, pp. 121-130.

[RUE86] Ruehli A.E. (Ed.), *Circuit Analysis, Simulation and Design*, Part 1&2, Elsevier Science Pub.1986,1987

[SAL87] Saleh R.A., Newton A.R., *An Event-Driven Relaxation-Based Multirate Integration Scheme for Circuit Simulation,* Proc.ISCAS'87, May 1987, pp. 600-603.

[SAL89] Saleh R.A., Newton A.R., *The Exploitation of Latency and Multirate Behavior using Nonlinear Relaxation for Circuit Simulation,* IEEE Trans.CAD, Vol.8, No.12, Dec.1989, pp. 1286-1298.

[SAL94] Saleh R.A., Jou S.J., Newton A.R., *Mixed-Mode and Analog Multilevel Simulation,* Kluwer Academic Pub. 1994.

[SAL96] Saleh R.A., Antao B.A., Singh J., *Multilevel and Mixed-Domain Simulation of Analog Circuits and Systems,* IEEE Trans.on CAD of Int.Cir. and Sys., Vol.15, No.1, January 1996, pp. 68-81.

[SCH87] Schwarz A.F., *Computer-Aided Design of Microelectronic Circuits and Systems*, vol.2, ch.9.3, Academic Press 1987.

[SOL74] Solomon J.E., *The Monolithic OPAmp: A Tutorial Study*, IEEE J.of Solid State Cir., vol.SC-9, Dec.1974.

[SZY76] Szygenda S.A, Thompson E.W., *Modeling and Digital Simulation for Design Verification and Diagnosis*, IEEE Trans., Vol.C-25, Dec.1976, pp. 1242-1253.

[TER83] Terman C., *RSIM - A Logic-Level Timing Simulator*, Proc.of ICCD, Port Chester, NY, 1983.

[TSA86] Tsao D., Chen C.F.,., *A Fast Timing Simulator for Digital MOS Circuits*, IEEE Trans.Computer-Aided Design, Vol.CAD-5, No.4, Oct.1986, pp. 536-540.

[VAC93] Vachoux A., Nolan K., *Analog and Mixed-Level Simulation with Implications to VHDL* in "Fundamentals and Standards in Hardware Description Languages" J.P.Mermet, Ed., NATO ASI Series E: Applied Sciences - Vol.249, Kluwer Academic Pub. 1993.

[VAC97] Vachoux A., *Analog and Mixed-Signal Extensions to VHDL,* Analogue Integrated Circuits and Signal Processing, Kluwer Academic Pub. vol.16, No 2, June 1997, pp. 185-200.

[VAN90] Van Stiphout M.T. et al., *PLATO: A New Piecewise Linear Simulation Tool,* Proc.of EDAC'90, 1990.

[VAR63] Varga R.S., *Matrix Iterative Analysis*, Englewood Cliffs, Prentice-Hall, New Jersey, 1963.

[VIS91] Visweswariah Ch, Rohrer R.A., *Piecewise Approximate Circuit Simulation,* IEEE Trans. Vol.CAD-10, No.7, July 1991, pp. 861-870.

[VSY94] V-System/Windows User's Manual, *VHDL Simulation for PC's Running Windows & Windows NT,* Ver.4.2f, Model Technology Inc., USA 1994.

[WES85] Weste N., Eshraghian K., *Principles of CMOS VLSI Design,* Addison-Wesley Pub.Comp., 1985.

[WHI85] White J., Sangiovanni-Vincentelli A.L., *Partitioning Algorithms and Parallel Implementations of Waveform Relaxation Algorithms for Circuit Simulation,* Proc.ISCAS'85, Kyoto, Japan, June 1985.

[WHI87] White J., Sangiovanni-Vincentelli A.L., *Relaxation Techniques for the Simulation of VLSI Circuits*, Kluwer Academic Pub. 1987.

[WIL79] Wilcox P., *Digital Logic Simulation at the Gate and Functional Level*, Proc.of 16th DAC, 1979, pp. 242-248.

[ZAJ98] Zając Z., *Automatic Parameter Extractor for Digital Networks Timing Modelling,*(in Polish), B.S. Thesis, Institute of Electronics, Silesian Univ.of Technology, Gliwice, 1998.

[ZAN69] Zangwill W.I., *Nonlinear Programming*, Prentice-Hall, Englewood Cliffs, N.J., 1969.

Piecewise Linear Approach to Functional-level Macrosimulation of Analogue & Mixed A/D Systems

Abstract

The monograph deals with an analogue macromodelling technique oriented towards functional-level simulation of both analogue and mixed analogue-digital networks/systems. Based on a brief overview of the actual simulation, the relation of the developed approach to the existing modelling and simulation techniques is presented. The signals are assumed to be piecewise linear (PWL) waveforms. A class of nonlinear (PWL) inertial building blocks is introduced for modelling. The proposed macromodels are accurate in their timing behaviour and computationally efficient, since an explicit algorithm to obtain the waveforms is used based on a PWL approximation of original smooth time responses. Alternatively, an enhanced trapezoidal rule is introduced. Practical macromodels of particular functional analogue units, such as an amplifier or a voltage comparator, are derived in detail and their performances are compared with SPICE estimates. Also the PWL approximation technique is compared with the enhanced trapezoidal rule.

A prototype event-driven, selective-trace simulator is used to verify the PWL approach by macrosimulation examples of practical A/D systems. For this purpose a concept of the PWL event is introduced. A virtual interface between the PWL- and logic domain is defined to support this kind of simulation. Digital units are modelled mainly as logic behavioural blocks.

To overcome the problems with feedback loops the PWL technique is supported by the waveform relaxation (WR). A PWL-WR algorithm is formulated and implemented as a prototype tool. Several examples of practical networks/ systems including feedback loops are considered. Practical convergence and stability conditions relevant to the linear case are derived as well. Besides, the feedback loop models and cascade models are shown to accumulate the PWL errors. To reduce this effect several methods are proposed, such as PWL model refinement, second-order building blocks (to avoid local loops) or tuned cascade structures. The latter are particularly well suited to synthesis of higher order analogue blocks, such as filters. In some of those cases the required computational overhead is increased. For cascade structures the trapezoidal rule-based PWL technique is shown to perform better than its approximation-based counterpart.

An implementation of the PWL macrosimulation technique in the discrete VHDL environment is presented, too. The basic building blocks are defined to be VHDL entities provided with a behavioural body due to the explicit formulas available to proceed these blocks. Analogue VHDL packages are used. For complex analogue models a structural approach is used. The digital nature of VHDL facilitates the implementation of mixed A/D systems at the expense of the interface added to provide a link between the PWL- and standard-logic domain. Practical examples verified with a VHDL simulator are included.

Zastosowanie techniki odcinkowo-iinowej do makrosymulacji systemów analogowych i analogowo-cyfrowych na poziomie funkcjonalnym

Streszczenie

Monografia poświęcona jest technice analogowego makromodelowania, która zorientowana jest na symulację analogowych i analogowo-cyfrowych układów/systemów reprezentowanych na poziomie funkcjonalnym. W oparciu o zwięzły przegląd metod symulacyjnych dokonano porównania zaproponowanego podejścia z istniejącymi technikami modelowania i symulacji. Przyjmuje się, że przebiegi czasowe są funkcjami odcinkowo-liniowymi. Jako bazę modelowania wprowadza się klasę nieliniowych (odcinkowo-liniowych) bloków inercyjnych. Zaproponowane w ten sposób makromodele są dokładne w sensie swoich odpowiedzi czasowych, a także efektywne obliczeniowo, ponieważ do otrzymania odcinkowo-liniowej aproksymacji oryginalnych, gładkich odpowiedzi wykorzystują bezpośredni (jawny) algorytm obliczeniowy. Alternatywnie wprowadzono w pracy udoskonalony algorytm trapezów. Szczegółowo wyprowadzone są makromodele praktycznych modułów funkcjonalnych, takich jak wzmacniacz czy komparator napięcia. Jakość tych modeli jest porównana z odpowiednimi estymatami uzyskanymi w oparciu o symulator SPICE. Także sama technika odcinkowo-liniowej aproksymacji została porównana z udoskonalonym algorytm trapezów.

Do weryfikacji podejścia odcinkowo-liniowego w oparciu o praktyczne przykłady makrosymulacji układów/systemów A/C wykorzystano prototypowy symulator sterowany zdarzeniami. Wprowadzono koncepcję zdarzenia charakterystycznego dla przebiegów odcinkowo-liniowych. Zdefiniowany został również wirtualny interfejs pomiędzy dziedziną sygnałów odcinkowo-liniowych i sygnałów logicznych. Cyfrowe bloki są modelowane zasadniczo behawioralnie.

W celu rozwiązania problemów powstających przy symulacji struktur ze sprzężeniem zwrotnym wprowadzono dodatkowo technikę relaksacji przebiegów. Algorytm łączący obie techniki zaimplementowano w postaci prototypowego narzędzia. Przeanalizowano wiele praktycznych struktur zawierających pętle sprzężenia zwrotnego. Dla przypadku liniowego sformułowano praktyczne kryterium zbieżności algorytmu iteracyjnego. Pokazano ponadto, że struktury ze sprzężeniem zwrotnym oraz kaskady bloków mają tendencję do akumulowania błędów mających swe źródło w zastosowanej aproksymacji. Dla zredukowania tego efektu zaproponowano kilka metod, takich jak korekcja modelu i algorytmu aproksymacji, wykorzystanie do syntezy bloków drugiego rzędu (co pozwala uniknąć lokalnych sprzężeń) oraz zastosowanie strojonych modeli kaskadowych. Ostatnie dwa

rozwiązania nadają się dobrze do syntezy bloków analogowych wyższego rzędu, typowo filtrów. Większość jednak z wymienionych rozwiązań wymaga zwiększonego nakładu obliczeń lub dodatkowo powoduje komplikację modelu. Na tym tle udoskonalony algorytm trapezów okazuje się być bardziej efektywny obliczeniowo, w szczególności w zastosowaniu do struktur kaskadowych.

W pracy przedstawiono także implementację techniki makrosymulacji odcinkowo-liniowej w dyskretnym środowisku VHDL. Podstawowe bloki funkcjonalne zdefiniowano w języku VHDL jako moduły behawioralne (entity), co było możliwe dzięki dostępności właściwych dla nich, jawnych (bezpośrednich) wzorów. Wykorzystano w tym celu analogowe pakiety języka. Dla złożonych modeli analogowych zastosowano podejście strukturalne. Modelowanie struktur mieszanych A/C w środowisku VHDL jest ułatwione oczywiście dzięki jego zasadniczo cyfrowemu przeznaczeniu. Wymagane jest jednak wprowadzenie wirtualnego interfejsu pomiędzy dziedziną sygnałów odcinkowo-liniowych oraz logiką standardową. Dołączone są również praktyczne przykłady uzyskane w oparciu o symulator VHDL.

Druk: Drukarnia Gliwice, ul. Zwycięstwa 27, tel. 230 49 50