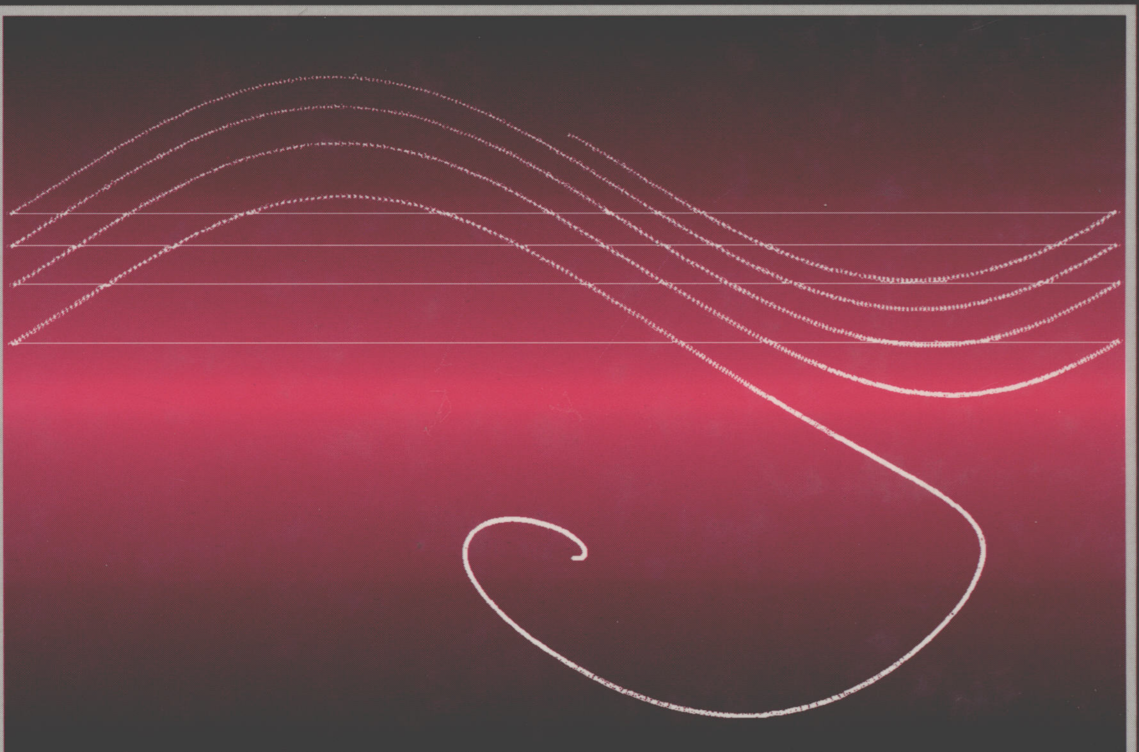
Advanced System Modelling and Simulation with Block Diagram Languages



Nicholas M. Karayanakis

Advanced System Modelling and Simulation with Block Diagram Languages

Nicholas M. Karayanakis
Professor
University of North Florida

CRC

CRC Press
Boca Raton New York London

Susan C. Karayanakis was responsible for the page layout and manuscript design executed using Word Perfect® 5.1. The typeface used was CG Times Scalable at 10.7 points before reduction. Formulas were created using MathEdit™ and imported into the document in the TIFF output format. Some figures were created using Flow Carting™ 3. Simulation language examples were captured using HiJaak® PRO.

Cover concept by Susan C. Karayanakis.

Catalog record is available from Library of Congress

International Standard Book Number 0-8493-9479-1
Printed in the United States of America 1 2 3 4 5 6 7 8 9 0
Printed on acid-free paper

This book is dedicated to my father,
Μάριος N. *Καραγιαννάκης*,
to *Καίτη* and to *Διάνα*, with lots of love.

ABOUT THIS BOOK

This work was conceived as a logical sequel to *Computer-Assisted Simulation of Dynamic Systems with Block Diagram Languages* published by CRC Press in 1993. Its contents reflect suggestions, challenges and requests from academic, industrial, and military people in the U.S. and from around the world.

The main objective of this book is two-fold: first to discuss the role of block languages as tools and to expose the technical features of several advanced languages. In the interest of diversity, we have selected ACSL/GM (Advanced Continuous Simulation Language/Graphic Modeller), ESL (European Space Agency Simulation Language), Extend, $MATRIX_x$, SIMULINK, SystemView, TUTSIM (Twente University of Technology Simulation Language, U.S. version), and VisSim. Most of the time, languages are discussed alphabetically. These discussions revolve about the technical aspects of each language. There is no intent of product comparison — that is a reader's task. Secondly, we have included discussions on critical simulation-related topics and on material pertaining to special simulation demands and their solutions.

Our efforts toward the synthesis of an informative and self-contained book on block languages led to the inclusion of a review section on block diagram algebra and applied transfer functions. To reiterate a position of long standing, we believe that block diagram algebra is clearly a branch of mathematics and is necessary knowledge for those working in continuous dynamic system simulation.

ACKNOWLEDGMENTS

First and foremost, I thank my wife, Susan, who dropped much of her own CPA work to become the true architect of my book one more time. My appreciation is extended to my editor Joel Claypool and to Michelle Veno of CRC Press. I have been fortunate to have the same outstanding editor who took a chance on me back in 1993 and has allowed me the comfort of author's autonomy throughout.

In the process of researching this book, I met and talked to many truly extraordinary people in the field of simulation languages. The lengthy discussions with language designers and marketing people provided me with unique knowledge and experience in a very complex domain. I am most grateful to the people listed here for their advice, courtesy, and the time they allotted so generously so that this book could come to fruition. I wish to thank Dusty Rhodes and Vera Mottino of Actuality, Geoffrey M. Chatfield, Dr. Patrick J. Ready, and Dr. Maurice L. Schiff of Elanix, Bob Diamond, Pat Diamond, and Kathi Hunt of Imagine That, Jeffrey Bach of Integrated Systems, Dr. John L. Hay of iSiM Simulation at Salford University, Peter S. Trogos and Dave Wakstein of The MathWorks, Dr. William M. Toscano and Marilyn Kloss of MGA Software, and Tracy L. Indresano, Peter Darnell, Vaughn Darnell, and Jim DeRemer of Visual Solutions. Great thanks are also extended to Faye Lu and the folks of the Peking Restaurant of Jacksonville for providing my wife and I with kindness and endless beverages while editing this book.

ACKNOWLEDGMENTS

[illegible], I thank my wife, Susan, who dropped much of her own CPA [illegible] the [illegible] of my book one more time. My appreciation is extended [illegible] Chappel and to Michelle Veno of CRC Press. I have been fortunate to [illegible] outstanding editor who took a chance [illegible] and has allowed [illegible] author's autonomy throughout.

[illegible] the process of researching this book, [illegible] many people in the field of simulation [illegible] designers and marketing people provided [illegible] experience in a very complex domain. I am most grateful [illegible] courtesy, and the time they allotted so generously [illegible] that this book could [illegible]. I wish to thank Dusty Rhodes and Vera Montone of [illegible], Geoffrey M. [illegible]field, Dr. Patrick J. Ready, and Dr. Maurice L. Schiff of [illegible], Bob Diamond, [illegible] Diamond, and Kathi Hunt of Imagine That; Jeffrey Bach of [illegible] Systems; Dr. John [illegible] of ISiM Simulation at Salford University; [illegible] and Dave W[illegible] of The MathWorks, Dr. William M. Toscano and Marilyn K[illegible] of MGA Software, and Tracy [illegible], Peter Darnell, Vaughn Darnell, and Jim DeKoven of Visual Solutions. Great thanks are also extended to Faye Lo and the folks of the Peking Restaurant [illegible] for providing my wife and I with lunches and endless beverages while [illegible] this book.

MANUFACTURERS ACKNOWLEDGMENTS

ACSL is a registered trademark and ACSL Model, ACSL Vision, and ACSLrt Realtime are trademarks of MGA Software/Mitchell and Gauthier Associates.

Elanix, MetaSystem, and SystemView are trademarks of Elanix, Inc.

Extend, Extend + Manufacturing, and Extend + BPR are registered trademarks of Imagine That, Inc.

FTN77 and FTN77/386 are trademarks of Salford Software, Ltd.

iSiM is a registered trademark of Salford University Business Services, Ltd.

Macintosh is a registered trademark of Apple Computer, Inc.

MATLAB and SIMULINK are registered trademarks and Handle Graphics is a trademark of The MathWorks, Inc.

$MATRIX_x$ is a registered trademark and AC-100, AutoCode, HyberBuild, MathScript, RT/Expert, RT/Fuzzy, Signal Analysis Module (SAM), SystemBuild, and Xmath are trademarks of Integrated Systems, Inc.

MS-DOS, Windows, and Windows NT are trademarks of Microsoft Corporation.

PC, XT, and AT are trademarks of International Business Machines Corporation.

Prospero Fortran and Pro Fortran 77 are trademarks of Prospero Software.

TableCurve is a trademark of Jandel Scientific.

TUTSIM and FANSIM are registered trademarks and TUTCAD is a trademark of TUTSIM Products, now Actuality Corporation.

VisSim is a registered trademark and VisSim/C-Code, VisSim/Neural-Net, VisSim/*Fuzzy*TECH, VisSim/Comm, and VisSim/Real-Time are trademarks of Visual Solutions, Inc.

Other product or brand names are trademarks or registered trademarks of their respective holders.

PROLEGOMENA

In recent years the proliferation of personal computers (PCs) and powerful portable continuous dynamic system simulation (CDSS) languages have influenced academia and industry profoundly. The virtual laboratory revolution based on the dynamic modelling and simulation of concepts, ideas, and things on the digital machine is expanding at a fast rate around the world.

We observe that industrial establishments are rapidly overcoming inertia, placing simulation in proper perspective — not as a form of video game for rogue white collar employees but as a *formidable tool* and a *necessary process*. In parallel, computer simulation is gradually finding its way into the educational and training schemes with great success. The process of dynamic simulation is based on the creative and heuristic interaction between people and computers for the purpose of solving problems, some of which are unsolvable by other methods.

Practically speaking, there is much to be said about freeing the worker (or the learner) from the tedium of repetitive calculation and plotting, from the burden of seeking and applying obscure and specialized analytical tools of marginal utility, and from the embarrassment of simplification and linearization. As Kemeny (1988) explains, many traditional topics and ways are now obsolete because of the computer. We believe it is all for the best — after all, it is difficult to argue on behalf of the small-angle-deflection assumptions when talking about, say, pendulum dynamics. After spending much time and effort to teach and learn trigonometry, we sanction the official elimination of sines and cosines from the equations, pretending that a pendulum only travels a few degrees each way!

Another important practical aspect of simulation is that the need for traditional laboratories, equipment, and parts inventories can be reduced to minimum in both the educational and the industrial settings. With simulation there is no need for exhaustive and expensive hardware prototyping or for outrageous R&D budgets. From a human factors perspective, there is no penalty for imagination because the virtual environment accepts both mathematical and intuitive thought. Workers may explore freely, casting doubts, inventing and investigating alternatives, and circumnavigating tradition (see also Karayanakis and Karayanakis, 1992a and 1992b).

Block language is a universally applicable tool, useful not only in the investigation and articulation of known systems, but also in inventing and developing new ones. Block diagram simulations point out the dynamic essence of things and creativity is defined in operational and repeatable terms (back in the 1960's Gordon wrote a lot about this in his best seller, *Synectics* (1961)).

In summary, simulation is good, useful, effective and, above all, cheap. At this point, the opening paragraph of *Winnie the Pooh* comes to mind,

> *Here is Edward Bear coming downstairs now, bump, bump,*
> *bump, on the back of his head, behind Christopher Robin.*
> *It is, as far as he knows, the only way of coming downstairs,*
> *but sometimes he feels that there really is another way ...*
> *if only he could stop bumping for a moment and think of it!*
>
> —— A. A. Milne

Nicholas M. Karayanakis
Jacksonville, Florida

CONTENTS

1 MODERN CDSS LANGUAGES

1.1 Introduction: Block Languages and Analog/Hybrid Machines

Modern block diagram (or CDSS) languages are invaluable tools in both teaching and research, especially when a mathematical model of the system under study is available. Historically CDSS languages were developed as "analog computers in a box," as digital simulators of the analog/hybrid (A/H) machine itself (Karayanakis, 1993, pp. 3-6). As modern computer languages, they are of the highest level with programming done by linking function blocks either as visual objects or by means of code. We view block languages as contemporary superior replacements of the A/H machines. Sentimental aspects aside, these machines (still operating in some places) share the dinosaur designation with the mainframes of the past. Contrary to the objections of the A/H computer hardliners and paleotraditionalists, it makes sense to consider today's block languages as natural descendants and evolutionary products of A/H machines, which in our best interest belong in museums only.

The publication of the author's 1993 book on block languages led to comments by A/H computer enthusiasts still brooding over the scrapping of their machines by their employers or institutions. These folks feel that their beginning to use block languages will imply betrayal of basic principles and old electromechanical friends. *As a result, valuable and highly transferable simulation expertise remains dormant, eventually to disappear.*

Without doubt, a well-chosen CDSS language is far superior to the best of A/H machines. A comparison of the basic features shows that:

1. A major tactical advantage of block languages is that amplitude or time scaling and the associated check procedures *are not required*. Toiling over maximum value calculations, potentiometer, and amplifier assignment sheets and static checks are gone forever. Instead workers may address simulation problems without engaging in error-prone, time-consuming preparations. However, block languages will accommodate a scaled simulation diagram just as easily.

2. Analog/hybrid computation is hardware intensive as opposed to the highly-portable block language approach. Users may obtain the software and use them in their own computers anywhere. By degree, simulation projects *do not require* dedicated facilities, as is the case in A/H computation.

3. The use of block languages frees workers from the drudgery of x-y plotter, strip chart, and repetitive oscilloscope hard-copy records. Instead, easy-to-follow menus and dialog boxes allow the specification of plot type, range, and input; the final result is publication-ready graphics of the highest quality and precision. In this context, A/H machine outputs and readouts are no less than primitive.

4. Most CDSS languages have analysis and signal processing features which defy imagination. Some CDSS languages are subsets of large, extremely robust mathematical systems offering computational and analytical tools unknown just a few years ago (Section 3).

5. All useful block languages have user-programmable function-generator blocks used in arbitrary-function generation. This approach is a radical departure from the highly inaccurate diode-function generation known to analog computation and superior to the digital lookup tables of hybrid computers. The topic is discussed in detail in Section 4.2.

6. The integrator is the heart of a simulator. Analog/hybrid machines are based on inaccurate and error-ridden integrators constructed with operational amplifiers (OAs) and problematic passive components. Block languages feature a wide choice of integration algorithms having an accuracy exceeding the best hardware by many orders of magnitude (Section 1.3). Virtual integrator blocks have no saturation limits like their OA-based counterparts. There are *no* linearity, drift or offset issues, *no* potentiometers to trim. Unlike A/H computers, digital block languages assure *repeatability* of experiments. Virtual integrators are available in all configurations an algorithms suitable for *stiff* systems exist (see Sections 1.3 and 4.11).

7. The nemesis of 'not enough amplifiers' or 'not enough integrators' to handle a given problem does not exist in block languages (except perhaps in some very inexpensive student editions). The same goes for other mathematical function blocks. As a result, not only problems of any size and complexity can be handled, but also programming shortcuts are unnecessary.

8. Workers using block languages can design their own custom computational blocks, custom signal processing blocks, etc. This is often done by creating *macroblocks as embedded block systems* (i.e., hierarchical models) or by writing code. *Multilevel nesting* or *encapsulating* is a powerful feature in block languages (see Section 1.4.2). Cases in pooint are ACSL's **PowerBlock**, Extend's hierarchical block, the **SuperBlock** of $MATRIX_X$, SIMULINK's group, the **MetaSystem** object found in SystemView, and the **Compound** block of VisSim, to name a few. Real world interfaces providing a number of analog and digital input/output (I/O) channels allow the block simulator to be as real as hardware, but more flexible and with much less burden.

9. CDSS languages are characterized by fast learning curves. Users need not be electronics experts or know anything about operational amplifier-based circuitry. Extraneous variables like impedance matching, electronic noise, component loading, and calibration are absent. There are no downtimes and worn-out patching cables. Just about any analog/hybrid computational device can be put into block form. In addition, block languages have a wealth of features and functions that do not or cannot exist in A/H computation. As a reminder, the proverbial "garbage in, garbage out" rule holds true here — it is assumed that workers are proficient in whatever they are doing and have taken time to learn the basics of block language simulation.

Finally we shall establish some connectivity between electronic circuit signal processing units and block diagram structures. A variety of electrical and electronic circuits and their block simulators are shown here to include: the generalized voltage divider of Figures 1.1.1 and 1.1.2; the R, L, and C models of Figure 1.1.3; and the series LP and HP filters of Figures 1.1.4 and 1.1.5. Figure 1.1.6 shows two possible ways to simulate uni- and bi-directional passive limiters. Several operational amplifier (OA) circuits follow: the simple noninverting follower with gain of Figure 1.1.7; the open-loop noninverting comparator of Figure 1.1.8; and the basic open-loop comparator and OA model of Figure 1.1.9 (note that the basic three-stage OA model *does not provide* frequency selectivity). Figure 1.1.10 shows a summer/subtractor network; Figure 1.1.11 shows the basic inverting summer; and the summer with gain (or attenuation) is shown on Figure 1.1.12. A scaled-input summer is shown in Figure 1.1.13. Other interesting OA circuits and their uncomplicated block representations include the zero-threshold comparator of Figure 1.1.14 and the polarity difference detector of Figure 1.1.15. Also the arithmetic average circuit of Figure 1.1.16, the weighted average circuit of Figure 1.1.17, and its specialized-input version (Figure 1.1.18) are representative OA circuits found in analog computation.

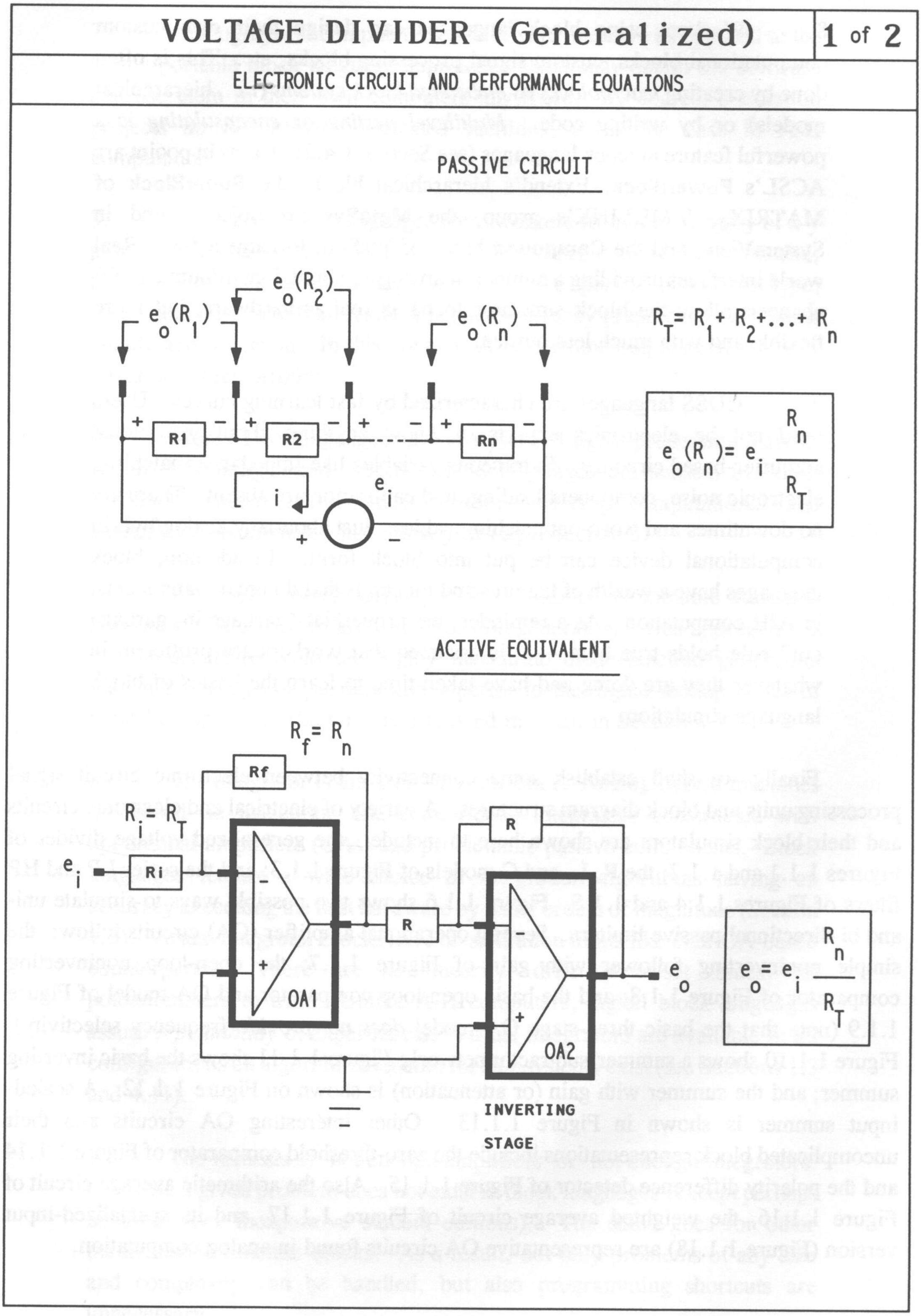


Figure 1.1.1 The generalized voltage divider.