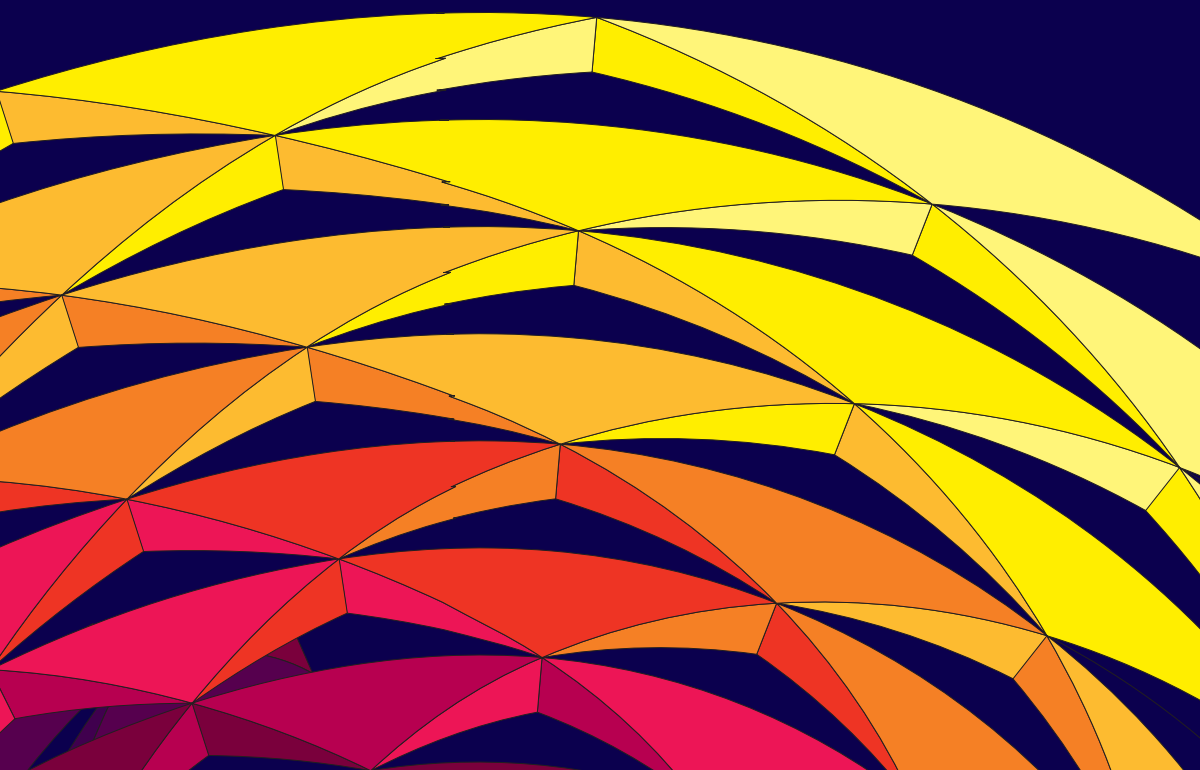
Computer Arithmetics for Nanoelectronics



Vlad P. Shmerko
Svetlana N. Yanushkevich
Sergey Edward Lyshevski

CRC Press
Taylor & Francis Group

Computer Arithmetics for Nanoelectronics

Computer Arithmetics for Nanoelectronics

Vlad P. Shmerko
Svetlana N. Yanushkevich
Sergey Edward Lyshevski

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Version Date: 20131125

International Standard Book Number-13: 978-1-4200-6623-4 (eBook - PDF)

This book is dedicated to the memory of
Claude Shannon

CONTENTS

Preface

THE MOTIVATION: TOWARD PREDICTABLE TECHNOLOGIES

This book is motivated by the emerging device- and system-level solutions to implement various computing and processing platforms created with nano- and molecular technologies. This ultimately leads not only to the deployment of unique phenomena that which never have been utilized in conventional solid-state microelectronics devices but also to advanced system organizations and architectures. The three-dimensional (3D) device and system topologies, multiterminal multiple-valued devices, and other features are subproducts of those emerging solutions. For these new generations of computing devices and systems, the conventional design and analysis concepts and techniques are the valuable source of further developments.

There are many examples of the use of the fundamentals of computer structure design and computing in nanocomputing, including particular results, that have not been adopted in contemporary design. For example, the principle of programable logic arrays (PLA) has been adopted for the crossbar-based array design and classical fault-tolerant techniques modified for manufacturing of computing nanodevices and nanomemory. Computing paradigms based on cellular arrays are the focus of massive parallel nanoarrays. Hence, the computing paradigms of contemporary discrete devices play an essential role in the development of novel computing structures based on the technological advances in molecular electronics. This book addresses these problems.

These developments integrate theoretical computer science, computer engineering, electrical engineering, microelectronics engineering, and other areas. Researchers and students from computer and electrical engineering, chemistry, biology, mechanics, and physics departments were involved in this interdisciplinary study.

It should be noted that usage of the uniform terminology is another problem since the terms used in the area of nanocomputing are acquired from different disciplines. In particular, the term *molecular electronics* covers a broad range of topics, in particular, molecular computing devices, device physics, and

biophysics of biomolecular devices. With a major emphasis on theoretical and applied computer science and engineering, in this book we focus on computing structure design based on computing primitives; the basic components such as switches, logic gates, 1-bit adders, and cells for computing arrays; and memory structures.

A wide variety of physical, biological and chemical phenomena, considered to be the candidates employed in processing, transforming, and storing the information at nano scale. These principles and phenomena are completely different compared to conventional solid-state devices. Various physical, biological, and chemical effects; phenomena; and transitions can be interpreted as computing events that are associated with the simplest logic operations, such as switching, OR, AND, and others. These logic operations correspond to the logic elements, which can be integrated and aggregated to perform complex functions.

This book provides the computing concepts, techniques, and tools in the form acceptable to specialists from various fields for the coherent and synergic design of computing structures utilizing the conventional and emerging (nano and molecular) processing devices. We also approach the problems of biomolecular processing typifying the biosystems' topologies, organizations, and potential principles of computing, such as multiple-valued, stochastic, parallel, reconfigurable, neuromorphical, etc. To this end, we elaborate the following two major aspects:

(*a*) The ability to interpret chemical, biological, and physical phenomena at the system level in terms of computing from the viewpoint of theoretical computer science and engineering, and

(*b*) The ability to utilize various chemical, biological, and physical phenomena to guarantee computing and processing at the device level.

This book integrates those developments by means of developing the fundamentals and theory of the information processing, computer structure and logic designs, stochastic computing, etc. Correspondingly, to reflect the aforementioned tasks and prospects, we title this book Computer Arithmetics for Nanoelectronics. In addition to conventional computing, fundamentals of biomolecular processing are introduced, in particular, new paradigms for delegation of computing properties into spatial nanostructures.

HOW THIS BOOK SATISFIES THE ESSENTIAL NEEDS OF INTERDISCIPLINARY COLLABORATION

With the advent of new technologies, a major shift has occurred in interdisciplinary studies. As a result, many engineers and scientists have found it necessary to understand the basic operation of digital systems, and how these systems can be designed if they carry out particular data-processing

tasks. This trend has produced a need for basic knowledge in computing structure design to provide a unified overview of the interrelationship between fundamentals of digital system design, computer organization, architectures, and micro- and nanoelectronics.

To comprehend this book, no specialized knowledge of computer science, electrical circuits, electronics, physics, or chemistry is assumed. This book is written from interdisciplinary prospects for the development of a new generation of computer devices and systems.

There are 20 chapters in the book. Each chapter targets a specific tasks and covers particular problems. The key topic is computational data structures. We emphasize on choosing an appropriate data structure under the phenomenological criteria of technology. Computational properties of various data structures are examined and characterized. Our goal is to achieve balance in introducing the computational properties provided by data structures and the phenomenological properties of current and expected technologies. For example, logic networks and decision diagrams are different data structures, and each of them is characterized by a set of specific requirements in implementation. At the device level, processing and computational properties of various physical and chemical phenomena vary, and can be systemized with respect to the computational characteristics of data structures. The balancing of these characteristics and requirements can provide an appropriate implementation, and it can be illustrated as shown in Figure 0.1.

The main goal of this book is to elaborate a consensus between computational properties provided by data structures and phenomenological properties of the technology-driven nano and molecular devices. For example, for a given biomolecular phenomenon, its representation in terms of processing must be understood by specialists from related fields and efficiently applied in computing structure design. This book has been written with these objectives in mind.

This book contributes to design of computing structures with a focus on sound contemporary technologies and devices. It is up to date, comprehensive, and pragmatic in its approach. The book aims to provide balanced coverage of concepts, techniques, and practices. We found that five topics were essentially useful in our interdisciplinary studies:

Topic 1: Switch-based computing devices and molecular switches;

Topic 2: Multivalued data structures;

Topic 3: 3D computing structures and 3D molecular topologies;

Topic 4: Design for testability and imperfection of molecular structures; and

Topic 5: Natural computing (Figure 0.2).

The design cycle from task formulation to molecular-based implementation includes several phases, which are covered in the book's chapters. We start from a brief overview of the known computing devices reported in Chapter

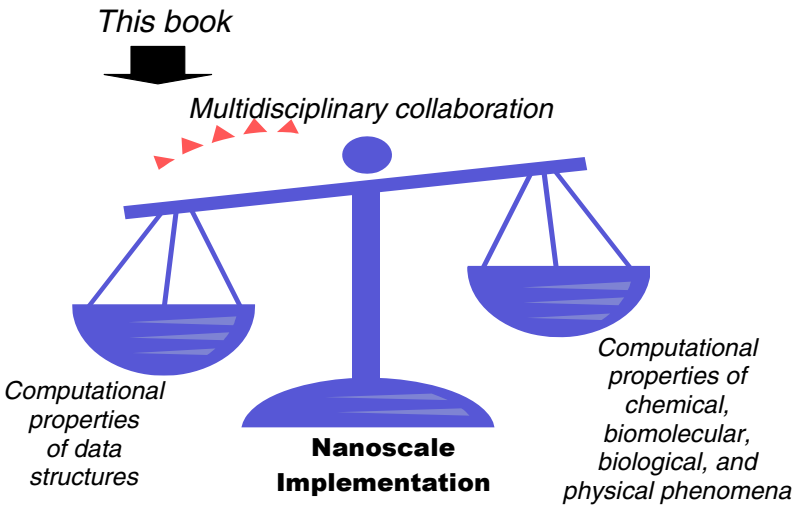


FIGURE 0.1
The main goal of this book is to contribute to the problem of the consensus between computational properties provided by data structures and phenomenological properties of nanoscale technology.

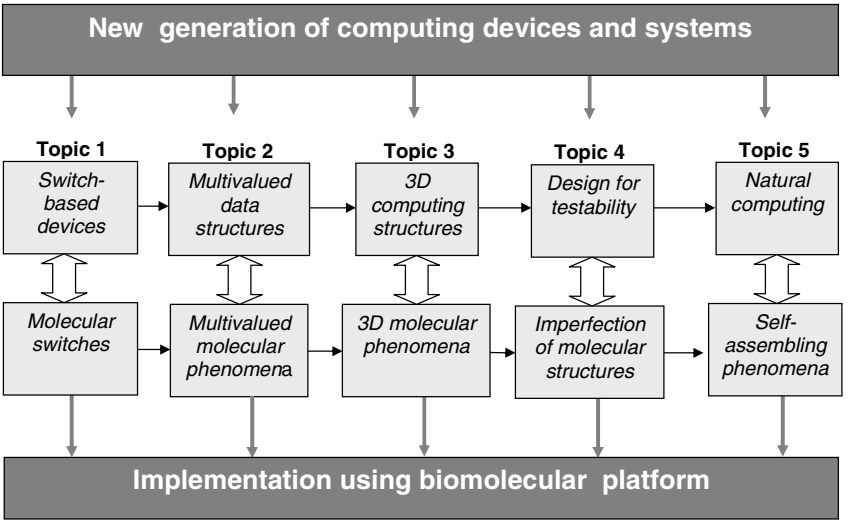


FIGURE 0.2
This book is based on seminars and discussions during the multidisciplinary project on development a new generation of computing devices.

2. The number systems, emphasizing binary arithmetic, are documented in Chapters 3 and 4. The residue number system is introduced as well, as it plays an essential role in many important applications such as encryption. Chapter 5 is a brief introduction to graphical data structures, which are useful computational models. The computational arithmetic must be embedded into the effects, phenomena, and transitions observed and exhibited by nano and molecular devices. The simplest computing devices are switches, and a trivial question is how computations can be mapped using the computing structures in which the operational nodes are implemented using switches covered in Chapters 6–10, 12, and 13. We introduced the theoretical base of various data structures in Chapter 6, their properties in Chapter 7, and techniques for their manipulation, optimization, and implementation in Chapters 8–10. This includes the polynomial forms of logic function, which are of the particular interest due to design tractability, simplicity and soundness as reported in Chapter 9. Polynomial forms are useful in many computational tasks and can be considered as appropriate candidates at the device-level implementation. Chapters 11–14 address the design techniques for the simplest logic networks, memory, and programmable devices. Molecular structures (aggregated devices) provide new opportunities for computing using multivalued logics. Techniques for representation and manipulation of multivalued signals and their implementation in discrete devices are reported in Chapter 11. The molecular devices are inherently 3D. We have proposed to delegate computational properties of the logic design data structures into 3D structures, as covered in Chapters 15 and 16. The information–theoretic measures and design aspects are given in Chapter 17. Chapters 18 and 19 report the testability problem. Natural computing based on various computing paradigms from nature is introduced in Chapter 20.

NEW CONCEPTS IN MULTIDISCIPLINARY STUDY

This book emphasizes the basic principles of computational arithmetic and computational structure design. Most of these principles are traditional in discrete devices design. However, the following key features distinguish this book from other known approaches in design and can be characterized as a major contribution:

- New design concepts and sound high-performance paradigms for data/signal processing and computing in 3D
- Extension of classical computing paradigms toward a 3D computing structures
- Reserving a central role for data structures as a key to applications; the relationships between various data structures and their manipulation

through design represent the most important aspect in the design of a new generation of computing devices

- Fundamentals of the intermediate data structures; these data structures are the bridge between physical and chemical phenomena and techniques for computing structure design

- Techniques for natural computing, such as evolutionary strategy for arbitrary logic network design, and neural computing for the elementary switching functions

This book is dedicated to the memory of Claude Shannon. Claude Elwood Shannon's inventive genius, probably more than that of any other single person, has altered humankind's understanding of communication and digital systems. He was born in Petoskey, Michigan, in 1916. At age of 20 he graduated with degrees in mathematics and electrical engineering from the University of Michigan. During the summer of 1937, Shannon obtained an internship at Bell Laboratories, and returned to MIT to work on a Master's thesis "An Algebra for Theoretical Genetics." He graduated in 1940 with his Ph.D. in mathematics and S.M. degree in electrical engineering. Shannon's Master's thesis won him the Nobel Prize, along with fame and renown. His thesis has often been described as the greatest Master's thesis of all time. In his spare time, Shannon developed the chess machine and remote control mechanical mouse, the first to do so.

Shannon was the first to notice that the work of professor George Boole, done a century earlier, yielded the necessary mathematical framework for analyzing switching networks. He demonstrated that any logical statement could be implemented physically as a network of switches. Shannon's revolutionary paper on information theory still dominates the area of communication theory. It is likely that techniques based on Shannon's information theory will become some of the main design tools for nanosystems, computing systems for the age of nanotechnology.

Vlad P. Shmerko

Svetlana N. Yanushkevich

Sergey Edward Lyshevski

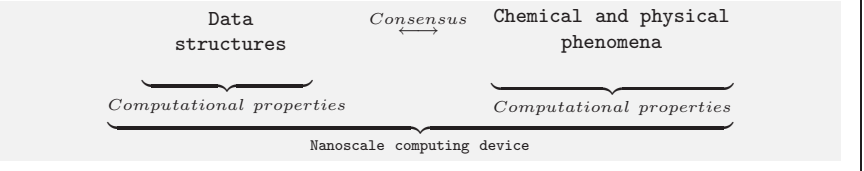
Calgary, Canada
Rochester, New York

1

Introduction

This book is aimed at introducing the reader to emerging computing based on various chemical, biochemical, and physical phenomena.

The main goal of this book is to elaborate a consensus between computational properties provided by data structures and phenomenological properties of the technology-driven nano devices:



Though one may envision a departure from matured microelectronics, this may take a while due to limited compelling needs, infrastructure expenditure, necessity constraints, and the overall ability of microelectronics to fulfill the current and near-future needs. There is a compelling need to comprehend and examine data processing by molecular hardware from the interdisciplinary point of view in the basic sciences, engineering, technology, and medicine. Among various immense tasks, the authors focus mainly on theoretical computer science and engineering, examining how to accomplish data processing and computing in the envisioned molecular platforms.

1.1 Computational paradigms for nanocomputing structures

Nanocomputing, or computing by nano and/or molecular structures (primitives), adheres to the fundamental principles of logic design of discrete devices and various molecular phenomena. While these principles, being related to data processing, may be similar to the existing ones, the computing techniques should be refined, revisited or redefined to fit the

technology requirements, device fundamentals, and system solutions. These problems can be addressed by applying contemporary logic design techniques to the design of computing structures. The computing paradigms and implementation principles define the form of computing structures, which embody the principles in the spatial and time solutions. This book provides a systematic view of the fundamentals of nano scale computing and introduces novel computing structures that satisfy the requirements of processing in three-dimensional (3D) space. Some particular cases of embodiment of contemporary design paradigms for discrete devices into nanocomputing structures are listed in Table 1.1.

Switch is the simplest computing device both in microelectronics and molecular (nano) technology. The simplest switch operates with one bit of information. For example, a switch can operate in two states, ON and OFF (connected–disconnected, open–closed, etc.). The other type of switch can direct a bit to one of two possible outputs (1-input 2-output switch). The switch is the basic element of implementation models in switching algebra.

Logic networks. These are the fundamental computing structures in logic design. Computing paradigm and design techniques of logic networks can be adopted from contemporary logic design of discrete systems, for example, two-level and multilevel logic network design and optimization. The elementary Boolean functions, such as AND, NAND, OR, NOR, and EXOR gate, can be computed using molecular switches. An arbitrary Boolean function can be computed by a logic network using the universal set (library) of logic gates.

Threshold networks. Logic networks that are designed using threshold gates are called *threshold networks*. An arbitrary Boolean function can be computed on a network of threshold gates. For example, two-input AND, OR, NOR, and NAND Boolean functions can be generated by a single threshold gate. Control over the type of Boolean function is exercised by the thresholds and weights.

Memory devices. Data storing can be accomplished by utilizing various phenomena exhibited, such as stable states, bistable states, m-stable states, and binding/unbinding. The implementation and utilization of these phenomena should be based on device physics applicable at the system level (logic networks, molecular aggregates, etc.).

Reprogrammable devices. These have provided significant performance improvements for many applications. FPGA (field programmable gate array)–based custom-computing systems can achieve high performance, providing near application-specific performance in an application-generic system. The simplest implementation of the field programmable method is a programmable logic device (PLD). PLDs are used primary for the *two-level* (sum-of-products) implementations of Boolean

TABLE 1.1
Nanocomputing structures adopt the computing paradigms, techniques, and data structures of contemporary computing

Contemporary computing	**Transferring**	**Nano scale computing**
Switch	$\Longrightarrow$	Molecular switch
Logic networks	$\Longrightarrow$	Logic networks over the library of molecular primitives
Threshold networks	$\Longrightarrow$	Logic networks over the library of molecular threshold gates
Memory devices	$\Longrightarrow$	Molecular memory devices
Reprogrammable logic arrays	$\Longrightarrow$	Crossbar-based computing arrays
Massive parallel computing	$\Longrightarrow$	Self-assembled molecular arrays
Self-reproducing automata	$\Longrightarrow$	Biomolecular self-assembly
Systolic arrays	$\Longrightarrow$	Systolic arrays over the library of molecular primitives
Stochastic computing	$\Longrightarrow$	Stochastic molecular computing
Fault-tolerant computing	$\Longrightarrow$	Molecular fault-tolerant computing
Neural networks	$\Longrightarrow$	Neuromorphic computing
Data structure	$\Longrightarrow$	Data structures for molecular computing

functions. PLDs have simple routing structures with predictable delay. PLDs and FPGAs are completely prefabricated devices. However, FPGAs are optimized for *multilevel* logic networks. This allows them to handle much more complex logic networks.

Example 1.1 *The most noticable feature of* **multi-FPGA systems** *is in the topology chosen to interconnect the blocks or chips. The most common topologies are* **mesh** *and* **crossbar** *topology. In a mesh, the chips in the system are connected to form a nearest neighbor pattern. This topology has the advantages of local interconnections, as well as expendability including to 3D, since meshes can be grown by adding resources to the edges of the array. Techniques of* **linear arrays***, which are essentially 1D meshes, are common in contemporary design. Crossbar topologies separate the elements in the system into logic-bearing and routing-only chips. These crossbar topologies are not expendable. Another topology, which combines the expendablity of meshes and the simple routing of crossbars, is known as* **hierarchical crossbars***.*

FPGA-like architectures can take advantage of architectural features from other computation domains and technology requirements, such as systolic arrays and digital signal processing, to provide a better resource structure than standard commercial general-purpose FPGA architectures.

Reconfiguration. The basic idea of reconfiguration is that the capability exists within a system to modify functionality after manufacturing. Reconfiguration is a widely recognized defect-and-fault-management technique in conventional design.

Example 1.2 *Examples of the* **reconfiguration** *are as follows: (a) ability to deactivate computing module within a module upon error diagnosis, (b) ability to switch to spare bits for single-cell failures in memory devices, and (c) bipass a memory device.*

Let us assume that a failure has been detected in a computing device. The failed unit must be bipassed, and a redundant resource must be activated and wired in. This is the key idea of fault avoidance through reconfiguration and redundancy.

Reconfiguration is often considered in conjunction with *redundancy* and *self-assembling*, and has been explored as a defect and fault mitigation approach for molecular-scale computing. the Reconfiguration can be obtained using so-called *polymorphic* principle of the computation. Polymorphic logic networks are those with multiple superimposed functionality; the output functions change due to changes in the operational point of the components.

Example 1.3 *Consider the* **polymorphic** *logic gates. Given the controlled supply voltages V_1 and V_2, the gate computes an AND and OR function on these voltages, respectively.*

Cellular arrays. The development process of biological organisms exploits essentially two mechanisms: (a) cellular differentiation (a cell copies its genetic material and splits into two identical cells), and (b) cellular division (the function a cell has to realize). In 1948, John Von Neumann introduced the model of *self-reproduction*. This model is known as a *cellular automata*. Self-reproduction is a prerequisite for any independent evolutionary process. A cellular automata is a finite or infinite network of identical deterministic finite state automata such that each cell has the same (finite) number and order of neighbors. Hence, a cellular automaton is a spatially distributed model of cells. The state of a cell at time $t+1$ depends on its own state and the states of neighboring cells at time t.

Cellular arrays are intrinsically parallel computing structures consisting of a regular latticework of identical logic elements (cells) that compute in lockstep while exchanging data with nearly cells. The properties of *local* computability, parallelism, and short-cascade chains make the performance of specific processing, such as matrix multiplication and vector operations, much higher. This is because the total length of interconnections from the inputs to the outputs decreases. Examples of cellular arrays are *crossbar-based* and *self-assembled* computing arrays.

(*a*) *Crossbar-based* computing arrays are based on the computing paradigm that a grid of the connected and addressable switches, sandwiched between the wires, have the property of storing and manipulation of data. Crossbar computing arrays are the computing paradigms that utilize a grid of the connected and addressable switches. These interfaced switches must ensure the specified functionality and capabilities from both device and circuitry prospectives.

Example 1.4 *OR, AND, and other elementary Boolean functions can be computed by the grid structure.*

This paradigm is attractive in realizing logic networks that implement memory and logic functions. The addressable molecular switches are set or reset by on–off switching, electrochemomechanical transitions, etc.

(*b*) *Self-assembled* computing arrays utilize and typify self-assembly, recognition, and other phenomena and mechanisms exhibited by biomolecules in living systems. There are no analog of this concept in microelectronics.

Biologically inspired computing is defined as a paradigm that typifies comprehended biological analogous of computing. Examples of biologically inspired computing are *artificial neural networks* and *evolutionary computing*.

(*a*) *Artificial neural networks* are an attempt to utilize natural-centric computing (with just moderate success to date) by implementing parallel processing capabilities and networked structures. The methods of contemporary logic design such as data structures design, their optimization, computer array structures, and others typify, to some extent, natural processing.

(*b*) *Evolutionary* computing centers on search and optimization, which are observed in living systems. Evolutionary algorithms have been used extensively in *evolvable hardware* [44]. Evolutionary algorithms have been shown to perform well in exploring large and complex design spaces, including logic network design.

Systolic arrays are special class of cellular arrays, that are based on the parallel-pipelined paradigm of computing. They are suitable for efficient computing if the parallel input/output is compatible with other devices in the system. *Linear* systolic arrays are a particular class of systolic arrays [47]. They do not require specific input/output interfaces when implemented as modules in computing systems. An arbitrary Boolean function can be computed on a linear systolic array.

Stochastic (probabilistic) computing is a fundamental concept in both contemporary design and nanocomputing. Envisioned nanotechnology- and molecular-based devices are expected to exhibit *deterministic* and *stochastic* electrochemomechanical phenomena that should be utilized to implement logic and memory functions. Stochastic computing promises to achieve fault tolerance by employing statistical principles in which deterministic logic signals are replaced with random variables.

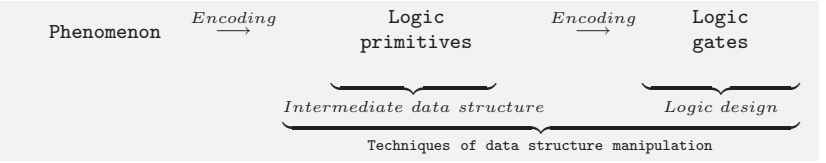
Fault-tolerant computing can be based on various classical approaches. These developments can be applied for nanocomputing since the failure rate of nanodevices is expected to be high. For example, molecular devices are expected to have a high probability of failure due to synthesis (technological) complexity, uncertainties, nonideal processing functionality, varying characteristics, etc. There are many techniques in discrete devices design that can be useful in nanocomputing, including *masking logic*, and *noise-tolerant* and *failure-immune* concepts. *Error*

correction codes and alternative number systems can be also used in the networked logic gates.

Neuromorphic networks are artificial neural networks that could be viewed to be biologically inspired or natural-centric ensure computing and memory storage. *Neuromorphic* networks utilize the principles of distributed computing, and the familiar examples of these models are the Hopfield network and the Boltzmann machine. At the gate level, this approach addresses artificial neural networks and is known as recursive stochastic models. They are based on the property of relaxation, that is, the ability to relax into a stable state in the absence of external excitations. *Relaxation* is referred to as the embedding of a correct solution into a network of logic nanocells called *artificial* neurons. There are no defined inputs and outputs in neuromorphic networks because they are distributed within the computing structure. Neuromorphic networks satisfy the criterion of fault-tolerant computing structures. One common model of neural networks is the *feedforward* network in which the processing unit is a linear threshold gate. A linear threshold gate computes an elementary Boolean function. It is wellknown that threshold gates are more computationally powerful than AND, OR, and NOT logic gates. Such a feedforward network can compute an arbitrary Boolean function.

Example 1.5 *The* **Hopfield** *networks can be used for modeling logic gates* [9, 28]. *Such a network is characterized by an energy function that has global minima only at the neuron states consistent with the truth table of the modeling gate. All other neuron states have higher energy. The energy function is uniquely specified by the weights and the thresholds of the neurons.*

Data structures provide various forms of representation of Boolean functions (algebraic, tabulated, and graphical forms). The theoretical base of data structures is Boolean and multivalued algebras. The special type of data structure, being used for the interpretation of molecular and physical phenomena, is called *intermediate* data. It is defined as *logic primitives*, which exhibit specified features of the molecular phenomenon and computing paradigm applying an appropriate encoding. Using specific assembling/interfacing/aggregating rules, logic primitives can be assembled into logic gates. The set of these gates (elementary Boolean functions) must be universal, that is, an arbitrary Boolean function can be computed using these specified library of gates. The role of intermediate data structures can be illustrated by the following scheme:



In this scheme, the encoding procedure plays a key role: all data must be specified via the assignments of the input and output variables, in particular, logical values 0s or 1s.

Example 1.6 *Using the symbolic quaternary alphabet* $\mathcal{A} = \{A, B, C, D\}$*, the* **intermediate data structure** *can be encoded (a) by the elements of this alphabet, (b) by specific orders of these elements in strings, and (c) by the operations with elements and strings, such as insertion, concatenation, deletion, and appending. These encoding schemes specify the set (library) of* **logic primitives** *and interactions between them. Such intermediate data structures are used in DNA computing (data are encoded by A, C, G, and T) and in quantum dot computing (data are encoding by dots).*

The analysis of Table 1.1 shows that the computing paradigms of contemporary discrete devices play an essential role in the development of novel computing structures based on technological advances of nanotechnology, molecular processing, and molecular (nano) electronics. This book addresses some major problems listed in Table 1.1.

1.2 Biological inspiration for computing

Processing principles of natural information are partially comprehended. The studies of natural data processing demonstrate that it is entirely different from the von Neumann computing paradigm.

1.2.1 Artificial neural networks

The human nervous system consists of small units called *neurons*. These neurons, when connected in tandem, form a nerve fiber. A biological neural net is a distributed collection of these nerve fibers. A neuron receives electrical signals from its neighboring neurons, processes those signals, and generates

signals for other neighboring neurons attached to it. The operation of a biological neuron, which decides the nature of its output signal as a function of its input signals, is not clearly known to date. However, most biologists are of the opinion that a neuron, after receiving signals, estimates the weighted average of the input signals and limits the resulting amplitude of the processed signal by a nonlinear function.

Artificial neural networks, to some extent, can be considered to be a *biology-inspired* computational concept, that implements natural-centric data processing. The cell body in an artificial neural net is modeled by a linear activation function.

Example 1.7 *The number of neurons in the human brain is estimated to be 100 billions; mice and rats have 100 millions neurons, while honeybees and ants have 1 million neurons.*

Artificial neural networks can be designed and modeled to implement *massively parallel* architectures. A neural network is a network of nodes and links. The nodes are elementary computing units that could typify to some extent neurons assuming a conventional action potential premise. Each node has an activation level that corresponds to a neuron's rate of firing, while each link has a numeric weight that corresponds to the strength of a synapse. Such networks can be trained to recognize patterns and compute functions.

Neural networks seem to be attractive for nanoelectronics due to intrinsic fault tolerance. The degree of fault tolerance of neural network can be evaluated using the degree of its redundancy.

Example 1.8 *The human retina has 125 million rod cells and 6 million cone cells. An enormous data, among other tasks, is processed by the visual system and the brain in real-time. Real-time 3D image processing, ordinarily accomplished even by primitive vertebrates and insects that consume less than* $1\ \mu W$ *energy to perform information processing, cannot be performed even by envisioned processors with trillions of transistors, device switching speed* $1\ THz$*, circuit speed* $10\ GHz$*, device switching energy* $1 \times 10^{-16}\ J$*, writing energy* $1 \times 10^{-16}\ J/bit$*, and read time* $10\ nsec$*.*

Example 1.9 *The information from the visual system, sensors and actuators is transmitted and processed within the nanoseconds range requiring μW of power. Performing enormous information processing tasks with immense performance, which are far beyond foreseen capabilities of envisioned parallel processors (which perform signal/data processing), the human brain consumes only 20 W. Only some of this power is required to accomplish information and signal/data processing.*

1.2.2 Evolutionary algorithms and evolvable hardware

Biological development is an example of a stunning mechanism that allows robust generation of complex organisms from a linear building plan, the DNA. *Evolutionary algorithms* are population-based stochastic search algorithms.

The evolution paradigm includes genetic algorithms, evolution strategies, evolutionary programming, and genetic programming. As an example of evolution paradigm, in this book, genetic algorithms are used for logic network design.

A genetic algorithm is an iterative procedure that consists of a finite-size population of individuals; each individual is represented by a string of symbols called the *genome*; a possible solution in a given problem space or search space is encoded in the genome. The algorithm sets out with an initial population of individuals that is generated at random. The individuals in the current population are decoded and evaluated according to an appropriate quality criterion called the *fitness function*. A new population is generated in the next generation. In this population, the individuals are selected according to their fitness and are transformed via genetically inspired operators, such as *crossover* and *mutation*. After some iterations, the genetic algorithm may find an acceptable solution.

Evolvable hardware is the application of evolutionary algorithms to the creation of electronic circuits. In evolutionary logic design, gate-level primitive components (gates, flip-flops, etc.) or graph-based abstraction are used to generate logic networks.

Example 1.10 *It was shown in many studies that various logic networks using FPGA of the configuration 10×10 cells can be created by a genetic algorithm with a population of size 50 and each string of size 2000 bits.*

1.2.3 Self-assembly

Molecular recognition, *complementarity*, and *aggregation* are well-established and sound principles. Molecular recognition implies the specific interaction between two or more molecules by means noncovalent (hydrogen bonding, metal coordination, hydrophobic forces, van der Waals forces, pi-pi interactions, electrostatic forces, etc.), and covalent bonds. For example, molecular recognition and molecular complementarity, exhibited by DNA, amino acids, and other biomolecules, can be significantly expanded utilizing organic and inorganic molecules. Stereochemistry studies the spatial arrangement of atoms, molecules, and molecular aggregates.

Most solid substances are crystalline in nature. Sometimes the particles of a sample of solid substance are themselves single crystals. Every crystal consists of atoms arranged in a 3D pattern that repeats itself regularly. The unit of this structure is the *unit cell*. For example, the unit cell of a cubic crystal is a cube. Crystals of many substances contain discrete groups of atoms, which are called *molecules*. The forces acting between atoms within a molecule are much stronger than those acting between molecules. At a low temperature, the molecules in a crystal lie rather quietly in their places. As the temperature increases, the molecules become more and more agitated. A molecule on the surface of the crystal is held to the crystal by the forces of attraction that its neighboring molecules exert on it. Forces of this kind are called *van der Waals* attractive forces.

Grouping means that larger objects are assembled out of smaller ones serving as their parts. It includes various procedures, such as clustering, class formation, and construction of strings. Typical example of grouping in biology is given below.

Example 1.11 *Cells form functional aggregated assemblies, which exhibit superior performance and capabilities. Living systems can be studied applying* **bottom-up** *hierarchies and* **top-down** *taxonomies.*

Specifically, many structural features of molecules and crystals that can be used in their interpretation in terms of computation are governed by *symmetry* and *grouping*. Symmetry operations, which leave one point in the space fixed, are called *point symmetry* operations. Point symmetry operations are *rotations* around an axis, *reflection* across a plane, and *inversion* through a point. The stacking of atoms or molecules side by side to build a crystal results in *translation* or *lattice* symmetry. The crystal lattice is the array of points at the corners of all of the unit cells in the crystal. The 3D array of symmetry elements itself is known as a *space group*.

Example 1.12 *Crystal growing is characterized by the grouping of molecules during solidification. A single crystal is the result of clusters gathered around a local center of crystallization. Crystals form their* **group hierarchies** *during crystal growth.*

An *aggregation* or *assembly* is formed an entity out of its parts. Each of the parts can also be obtained as a part of aggregation. *Self-assembly* is defined as the process by which an organized structure can be formed spontaneously from simple parts (molecules or various nanosize objects). It describes the assembly of natural structures such as crystals, DNA helices, and microtubules. The organization process is made into a desired structure via physical, chemical, or biochemical interactive processes involving, for example, electrostatic and surface forces. All these processes are very selective and reject defects so that the resulting structure is characterized by a high degree of perfection. Self-organization techniques are similar to the process of development of biological organisms.

Repeated duplication of a group of atoms by a screw axis produces a pattern called a *helix*. If the atoms are joined by chemical bonds in a continuous chain from one group to the next, the result is a helical molecule that extends the length of the crystal. It is possible to construct helical molecules by a symmetry operation similar to a screw axis, except that the angle of rotation from one group to the next is not an integral fraction of 360°. Some molecules of great biological importance have helical symmetry of this type, in particular, the α-helix of proteins and the helical backbone of the DNA molecule. Properties of helical symmetry can be encoded and interpreted in terms of computing Boolean functions.

The second law of thermodynamic states that in an isolated system, entropy can only decrease, not increase. Such system evolve to their state of maximum entropy, or thermodynamic equilibrium. The *thermodynamic* concept of entropy as the dissipation of heat is not very useful for computing systems. *Shannon entropy* is applicable to any system for which a state space can be defined. It expresses the degree of uncertainty about the state s of the system in terms of the probability distribution $P(s)$. In terms of Shannon entropy, the second law of thermodynamic can be expressed as "every system tends to its most probable state" [6]. At a molecular level, molecules are distributed homogeneously, and the most probable state of an isolated system is that of maximum entropy or thermodynamic equilibrium.

In molecular (nano) electronics, *self-assembly* is defined as a method of fabrication of the molecular computing structures that relies on chemicals, forming larger structures without centralized or external control. This is the spontaneous organization of molecules under thermodynamic equilibrium conditions into a structurally well-defined and rather stable arrangement. Self-

assembly in this system is associated with *bottom-up* design.

The key engineering principle for molecular self-assembly is to design molecular building blocks that are able to undergo spontaneous stepwise interactions. In this design, the instructions are incorporated into the structural framework of each molecular component. The running of these instructions is based on the specific interaction patterns, environment, and the intermediate stages of the assembly.

One of the main issues encountered in logic design based on self-assembling paradigm is the ability to match the components (logic primitives) into combinations that result in the correct output of logic network. The complementary molecular primitives (^{CM}primitives) are defined as a set of simplest molecular structures. Each molecular primitive can be interpreted in terms of switches, the simplest computing operation. Molecular primitive are used for designing the molecular logic gates. Various subsets of the molecular primitive can be derived from the ^{CM}primitives. These subsets form various libraries for design molecular logic gates often called *multiterminal* and *multifunctional* molecular devices utilizing various molecules (aromatic, cyclic, and other), polypeptides, and side groups.

Example 1.13 **Multiterminal** *solid molecular devices can be engineered as cyclic molecules arranged from atoms ensuring functionality. These devices includes switches (two-terminal device), two-input logic gates (three-terminal devices), three-input logic gates (four-terminal devices), and various multiple-input multiple-output computing networks.*

Logic design using the molecular primitives is based on self-assembling and results in random or partially-ordered computing networks. The controllable self-assembling and robust binding/pairing can be implemented using, for example, the *templates*. *Fractal* assemblies that can be interpreted in terms of Boolean functions carry the information about these functions by the labeled topological structures. These fractals are acceptable as templates.

Example 1.14 *An example of a structure that can be constructed using algorithmic self-assembly is a* **Sierpinski triangle**. *This structure is also known as a Sierpinski gasket, which is a kind of fractal structure. A Sierpinski gasket is the* **Pascal triangle** *modulo two, i.e., the EXOR operation is used instead of arithmetic addition while forming the Pascal triangle. There are* $(4^n + 2^n)/2$ *elements in total in Pascal triangle.*

Fractal-like templates are useful for systematic design for molecular systems

with logic processing capabilities. Methods of computer aided design (CAD) are classified into those for logic design based on logic primitives and those for designing molecular reactions.

1.3 Molecular computing devices

Synthetic chemistry allows one to synthesize a wide range of complex molecules from atoms linked by covalent bonds. Utilizing noncovalent and covalent intermolecular interactions, as well as precisely controlling spatial (structural) and temporal (dynamic) features, supramolecular chemistry provides methods to synthesize even more complex atomic aggregates.

Example 1.15 *The effective cell size of the envisioned microelectronic devices is projected to be* 500×500 *nm by the year 2025. Each of these devices will consist of billions of molecules. In contrast, molecular devices are expected to be synthesized from a couple of atoms or molecules.*

Molecular computing devices are comprised of

- Organic molecules
- Inorganic molecules
- Biomolecules

Molecular (nano) electronics focuses on fundamental/applied/experimental research and technology developments in the devising and implementation of novel, high-performance, enhanced-functionality, atomic/molecular devices, modules, and platforms (systems), as well as high-yield bottom-up fabrication. Molecular electronics centers on:

- Discovery of novel devices that are based on the new device physics
- Utilization of the exhibited unique phenomena, effects, and capabilities
- Devising of enabling topologies, organizations, and architectures
- Bottom-up, high-yield fabrication technologies

At the device level, the key differences between molecular and microelectronic devices are as follows:

The key differences between molecular and microelectronic devices

- Device physics and phenomena exhibited and utilized
- Performance, capabilities, and functionality achieved
- Topologies and organizations attained
- Fabrication processes, synthesis methods, and technologies used

The difference between microelectronic and molecular computing devices can be specified as follows. In microelectronic computing devices, individual molecules and atoms do not depict the overall device physics and do not define the device characteristics. In molecular computing devices, individual molecules and atoms explicitly define the overall device physics depicting the device performance, functionality, capabilities, and topologies.

There are fundamental differences at the system level. In particular, molecular electronics lead to novel organizations, advanced architectures, and the need for technology-centric, super-large-scale integration, novel interconnect, and interfacing. However, advanced techniques of logic design are the basis of the molecular electronics (Table 1.1).

Example 1.16 *In* [42]*, a molecular computing structure called* **nanocell** *have been introduced. A nanonocell is a 2D network of self-assembled molecules that act as reprogrammable switches. An array of nanocells implement the concept of the FPGA; the nanocells can be programmed and reprogrammed after fabrication to perform a specific functions. Genetic algorithm can be used for designing the logic network of molecular switches in nanocells.*

Interconnect and interfacing

The characteristics of nano scale devices include, in particular, unreliable device performance, transfer function, interconnect limitations (the inability to provide global interconnections), thermal power generation, and regularity of layout. Nanoscale devices need to be interconnected locally and patterned into 2D or 3D arrays of cells of various topologies.

Quantum cellular automata architecture (QCA) is a typical example of a regular and locally interconnected array of cells interacting with its neighbors, but there are no wires in the signal paths. In contrast to conventional silicon-based designs, where information is transferred between devices by electrical current, in QCA information is transferred by Coulomb interaction, which passes the state of one cell to its neighbors.

Example 1.17 *The wire in QCAs is a chain of* **quantum cells**. *The QCA is characterized by the ability to cross wires in the plane; different wires carrying different binary values can cross each other at the same level without any interferance or crosstalk.*

In locally connected structures, the range of interaction and the connection complexity of each cell are independent of the number of cells. Therefore,

these structure are scalable and massively parallel. Acceptable characteristics of reliability and robustness can be achieved by using special techniques. An example is a 2D cellular nonlinear network (CNN), which is an array of neuron-like cells [32, 39]. A cellular network can be mapped into a 3D topology [36].

Nanoscale design demand novel interconnect and interfacing solutions that ensure aggregation and assembly of molecular processing primitives in large-scale diverse modular modules. Molecular assemblies are comprised of functionalized aggregated molecules. In leaving organisms, biomolecules ultimately establish the biomolecular processing hardware. We focus on the *solid molecular devices*. In solid-state microelectronic devices, individual atoms and molecules have not been, and cannot be, utilized from the device physics prospective. The scaling down of microelectronic devices results in significant performance degradation due to quantum effects (interference, inelastic scattering, vortices, resonance, etc.), discrete impurities, and other features. In contrast, the molecular devices exhibit phenomena that can be uniquely utilized, ensuring device functionality and guaranteeing superior capabilities.

Organic synthesis is the collection of procedures for the preparation of specific molecules and molecular aggregates. In planning the syntheses of desired molecules, the *precursors* must be selected. One carries out the *retrosynthetic* analysis as

$$\texttt{Target molecule} \Longrightarrow \texttt{Precursor},$$

where the open arrow $\Longrightarrow$ denotes "is made from." Usually, more than one synthetic step is required. For example,

$$\texttt{Target molecule} \Longrightarrow \texttt{Precursor 1} \Longrightarrow \cdots \Longrightarrow \texttt{Precursor Z} \Longrightarrow \texttt{Starting molecule}$$

A linear synthesis, which is adequate for simple molecules, is a series of sequential steps to be performed, resulting in synthetic intermediates. For complex molecules, convergent or divergent synthesis is required. There are different procedures for synthesis of intermediates. For new synthetic intermediates, discovery, development, optimization, and implementation steps are needed.

In multiterminal solid molecular device, distinct quantum phenomena could be used to ensure the controlled characteristics. For example, quantum interaction, quantum interference, quantum transition, vibration, Coulomb effect, etc. The device physics, based on these and other phenomena and effects (electron spin, photon-electron-associated transitions, etc.), must be coherently complemented by the bottom-up synthesis of the molecular aggregates, that exhibit those phenomena.

Example 1.18 *Distinct solid molecular devices have been proposed, ranging from resistors to multiterminal devices* [5, 10, 17, 26, 33, 40, 43]. *These molecular devices are comprised of organic, inorganic, and bio-molecules. For example, Figure 1.1 shows, in particular, different molecules that were functionalized in order to perform acceptable characteristics for switching.*

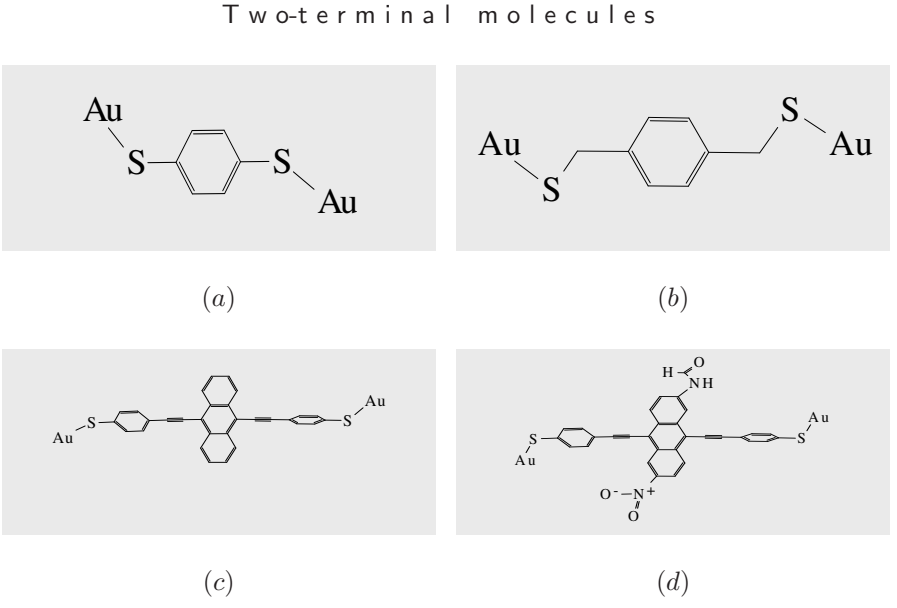


FIGURE 1.1
Molecules as potential two-terminal molecular devices: (a): 1,4-phenyledithiol molecule and functionalized 1,4-phenyledithiol molecule; (b): 1,4-phenylenedimethanethiol molecule; (c): 9,10-bis((2′-para-mercaptophenyl)-ethinyl)-anthracene molecule; (d): 1,4-bis((2′-para-mercaptophenyl)-ethinyl)-2-acetylamino-5-nitro-benzene molecule (Example 1.18).

Various problems associated with the devising, engineering, and analysis of functional molecular devices are reported in [26]. In order to depart from the symmetric organic molecular devices, asymmetric multiterminal carbon-centered molecular devices were proposed. These molecular devices are comprised from B, N, O, P, S, I and other atoms. To ensure synthesis feasibility and practicality, these molecular devices are engineered from

cyclic molecules and their derivatives. The reported multiterminal molecules ensure the desired asymmetry of the voltage-current characteristics, while the saturation region or peaks-and-deeps should be examined.

By applying the voltage to the control terminal, one varies the potential, regulates the charge and electromagnetic field, and varies the interactions, as well as changes the tunneling affecting the electron transport. Hence, the input-output characteristics can be controlled.

The aggregation and interconnect of input/control/output terminals can be accomplished within the carbon framework. For example, (a) the electron transport is predefined or significantly affected by X_i and side groups; (b) atomic structures of side groups can exhibit transitions or interactions under the external electromagnetic excitations and thermal gradient; (c) side groups can be utilized as electron donating and electron withdrawing substituent groups, as well as interacting or interconnect groups.

The functional molecular switches operate on one bit. This simplest computing is associated with switching (from an OFF state to an ON state, and vice versa), using stimuli such as voltage pulses. In multiterminal solid molecular device, quantum effects could be used to ensure the controlled voltage-current characteristics. Figure 1.2a shows the two-terminal molecular devices of various complexity.

Three-terminal cyclic molecules are utilized as molecular devices. The three-terminal molecular device is based on the quantum interaction and controlled electron transport. The inputs signals V_A and V_B are supplied to the input terminals, while the output signal is V_{out}. These molecular AND and NAND gates are designed using cyclic molecules within the carbon interconnecting framework. By applying the voltage to the control terminal, one varies the potential, regulates the charge and electromagnetic field, varies the interactions, as well as changes the tunneling affecting the electron transport. Hence, the input-output characteristics can be controlled.

Example 1.19 *Three-terminal molecular devices can perform Boolean functions AND, NAND, OR, and NOR of two variables (two-input) logic primitives (Figure 1.2).*

Example 1.20 *Molecular and biomolecular devices can operate with the estimated transition energy* 1×10^{-18} *J, discrete energy levels (ensuring multiple-valued logics and memory) and femtosecond transition dynamics. These guarantee exceptional device transition (switching) speed, low losses, unique functionality, and other features ensuring superior overall performance.*

The term *molecular electronics* covers a broad range of topics, in particular,

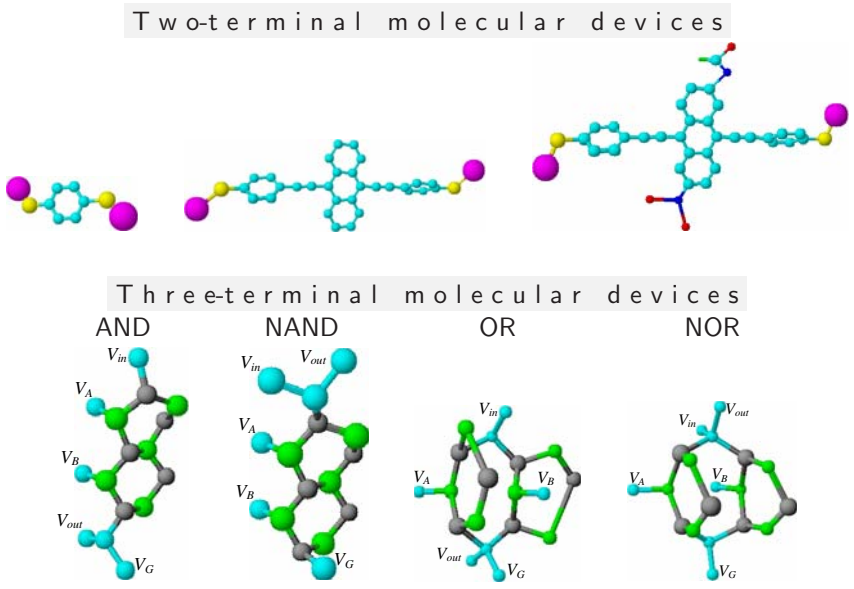


FIGURE 1.2
Molecular gates: AND, NAND, OR and NOR gates (Example 1.19).

molecular computing devices, device physics, and biophysics of biomolecular devices. Molecular materials for electronics deal with films or crystals that contain many trillions of molecules per functional unit. Molecular-scale electronics deal with one to few thousand molecules per device.

Molecular electronics involves the search for single molecules or small groups of molecules that can be used as the fundamental units for computing. The goal is to use these molecules to have specific properties and behaviors. Molecular devices for computing Boolean functions consist of (a) molecular terminals, and (b) molecular wires, which are materialized by means of interconnection phenomena.

Cross-talking

The computing units in molecular devices use different input signals; to assemble them into computing devices and systems of higher complexity, the standardization of the input/output signals are required. Most of the chemical reactions are of low specificity. That is, different reactions in one system may cause interference. This problem is called *cross-talking*. Cross-talking between the different reactions hinders assembling of chemical computing devices and systems.

Mixed silicon-based and molecular electronics

A hybrid between present silicon-based technology and technology based purely on molecular switches and molecular wires is a more viable path toward nano scale computing systems. A mixed paradigm presents opportunities for more tightly integrated mixed systems that utilize the advantage of the strong points of each technology.

Traditional digital logic design techniques can be used, and assemblies of switches and logic gates can be constructed. The resulting logic networks will operate in exactly the same manner as traditional silicon electronic-based circuits.

1.4 Fault tolerance

The stochastic nature of biomolecular systems can lead to random variation in the concentration of molecular species. Mutation or imperfect replication can alter the inserted gene sequences, possibly disabling them or causing them to operate in unforeseen ways. This type of computing is called *stochastic* computing. Stochastic computing achieves fault tolerance by employing statistical models in which deterministic logic signals are replaced with random variables. In this model, correct output signals are calculated with some probability. When a noise is accommodated, the Boolean function is replaced with a random function. The applications of various models for increasing the reliability of computing require small circuits or simple logic elements. Hence, efficient assembly and interconnects are critical for the implementation of stochastic models of computing. Conceptually, in stochastic computing, the problem of suppressing unwanted random effects is reformulated into the problem of efficient utilization of uncertainty.

The increase in complexity of the logic networks increases the probability of faults. Any architecture built from large numbers of these devices will contain a large number of defects, which fluctuate on time scale. The problem is to develop a computing structure, that is dynamically *defect tolerant*. That is, the problem is not only to test the correctness of logic networks but also to design networks resistant to faulty components. Biological systems are examples of complex fault-tolerant systems; these biological mechanisms can be used in the development of novel *self-test*, *self-repair*, and *self-replication* approaches. Self-replication means the capability of a machine to produce a copy of itself.

Fault detection is the analysis of errors to determine which components are faulty. Once the error is detected, the appropriate action must be taken. A property, required in all fault-tolerant computing techniques is that of *fault isolation*. Fault isolation aims to prevent a faulty unit from causing incorrect

behavior in a nonfaulty unit.

As devices increase in complexity, defect, and contamination, control becomes even more important since defect tolerance is very low. Nanoscale devices will have a high probability of failure, that is, they are characterized by a high and dynamic failure process. These failures can be occur both during fabrication and at a steady rate after fabrication.

There are several approaches to deal with nanodevice failure rates: (a) design for testability; in this approach, the design cycle of logic network is considered under conditions of the testability in all design phases; (b) design based on redundant logic; the key assumption of this approach is that the redundance can be incorporated at various level of logic network over a set of possible faults; and (c) design based on probabilistic computing paradigm; the key idea of this approach is the adaptation to errors.

Assuming reliable computing in the presence of faults is called *fault tolerance*. For example, approaches to reliable computing in networks with faulty nodes and/or interconnects can utilize the error correction codes and residual number systems. These approaches state that it is possible to correct a class of faults if a library of *reliable* logic nanocells for implementing the correction is available. Error-reduction/correction in biocomputing systems becomes extremely important as the complexity of numerous connected biochemical reactions increases. This is needed for scalability and fault tolerance of the computing structures *Self-repairing* electronics are the areas of fault-tolerant computing. Traditional approaches to preserve electronics incorporate radiation shielding, insulation and redundancy at the expense of power and weight. The *self-adaptive* system can autonomously recover the lost functionality of a reconfigurable array.

Example 1.21 *NASA uses various approaches to prevent radiation, and extreme-temperature, hardened electronics, required by space missions to survive the harsh environments beyond Earth's atmosphere* [21]. *The* **self-adaptive** *systems are operating in extreme temperatures (from* $120^{\circ}C$ *down to* $-180^{\circ}C$*). The reasonable implementation of a self-adaptive principle is based on FPGAs. Another approach, called* **tuning reconfigurable** *electronics, is related to an inexpensive, navigation grade, miniaturized inertial measurement units, which surpass the current state-of-the art technology in performance, compactness (both size and mass), and power efficiency. This approach used by all NASA missions. The* **self-tuning** *techniques for reconfigurable micro-electro-mechanical systems (MEMS) gyroscopes based on* **evolutionary** *computation.*

Example 1.22 *Fault tolerant computing array are often based on the principle of* **reconfiguration**. *This principle is implemented by integration of redundant cells. Fault cells and their locations are detected, and the computer structure reconfigures around these faulty cells.*

The drawback of this approach is that the reliability of fault detection logic must be guaranteed.

1.5 Computing in 3D

It has been justified in a number of papers that 3D topologies of computer elements are optimal, physically realistic models for scalable computers. In the contemporary logic network design, the use of the third dimension is motivated by decreasing the interconnect topology. The third dimension is thought of as *layering*. For example, in chips, networks are typically assembled from layers, where logic cells occupy the bottom layers and their interconnections are routed in upper metal layers.

Example 1.23 *A widely accepted* **2D placement** *model for conventional microelectronics is a square grid with all logic network elements of unit square size with their input/output connections in the center. Network elements are then placed, in checkerboard fasion onto the grid. The logic network to be put in place is represented by a hypergraph, where network elemebts form the set of nodes. The goodness of the placement is measured by its total wire length. A* **3D placement** *of FPGA, for instance, provides a significant reduction of wire-lengths.*

In this book, the third dimension is considered not because of layers, but as a dimension that relates to electrochemomechanical phenomena [46]. It includes the space orientations and 3D relationship between molecules, as the key factor for achieving the desired functional logical properties. This is the motivation to search for adequate spatial computing models. A 3D directly interconnected molecular electronics concept utilizes a direct device-to-device aggregation based on molecular recognition, complementarity, and aggregation.

1.6 Multivalued processing

Multivalued algebra is a generalization of Boolean algebra, based upon a set of m elements $M = \{0, 1, 2, \ldots, m\}$, corresponding to multilevel signals and the corresponding operations. Multivalued logic has been proposed as the means to [7, 18, 19]

- Improve performance
- Increase the packing density of VLSI circuits
- Improve fault tolerance
- Reduce power and power dissipation
- Improve testability

In multivalued logic networks, more than two levels of a signal are used. There are many motivated examples for considering the processing of *multilevel* signals as biologically inspired processing.

Interconnections. One of the most promising approaches to solving the interconnection problems is the use of multivalued logic. The number of interconnections can be reduced with multilevel signal representation. The reduced complexity of interconnections makes the chip area and delay much smaller.

Packing density. The chip area can be evaluated using the interconnections. The number of interconnections for the networks that use m-level logic signals can be estimated as $1/\log_2 m$, compared with two-level ($m = 2$) logic signals, $1/\log_2 2$. In the case of 2D topology, the reduction becomes $1/(\log_2 m)^2$ [18]. The total area of interconnections can be calculated using the number of interconnections, topology of interconnections, and the length of interconnections.

Pads. The number of bonding pads in a chip can be reduced by using multilevel signals.

Testability can be improved because access to internal components of a logic network is available using extra pads.

Performance can be improved because of the decrease of the total length of interconnections and interconnections delay (the switching time of gates is much less than interconnection delay). Special encoding methods, such as radix-r signed-digit number system and residue symmetrical number system, as well as parallel hardware algorithms using multivalued logic, enable local computing. These are additional resources for increasing the performance.

Power dissipation can be decreased because the dynamic power dissipation is determined mainly by the interconnections.

Fault tolerance. Cross-talk noise is increased because of extremely small distances between wires. Special encoding methods, using multivalued logic enable reduction of cross-talk noise.

Sequential logic networks. Multivalued logic is the base for development of the racing-free asynchronous sequential elements and networks.

Special encoding methods for local computing. In signed-digit arithmetic, the carry propagation in arithmetic operations such as addition and substraction is localized by one digit position; that is, massive parallelism of the computation can be achieved. In the residue number system, addition and multiplication are inherently carry-free; this number system is suitable for massively parallel arithmetic operations. A signed-digit arithmetic and residue number system can be implemented by binary logic networks. However, multivalued logic provides them more efficiency compared with binary logic.

In practice, multivalued logic networks are modeled by a multilevel encoding of information.

1.7 Further study

[1] Adamatzky A. Computing with waves in chemical media: Massively parallel reaction-diffusion processors, *IEICE Trans. Electron.*, E87-C(11):1748-1756, 2004.

[2] Adleman LM. Molecular computation of solutions to combinatorial problems. *Science*, 226, November, pp. 1021–1024, 1994.

[3] Aguirre AH and Coello CAC. Evolutionary synthesis of logic circuits using information theory. In Yanushkevich SN, Ed., *Artificial Intelligence in Logic Design*, pp. 285–311, Kluwer, Dordrecht, 2004.

[4] Aoki T, Homma N, and Higuchi T. Evolutionary synthesis of arithmetic circuit structures. In Yanushkevich SN, Ed., *Artificial Intelligence in Logic Design*, pp. 39–72, Kluwer, Dordrecht, 2004.

[5] Aviram A and Ratner MA. Molecular rectifiers. *Chem. Phys. Letters*, 29:277–283, 1974.

[6] Beer S. *Decision and Control: The Meaning of Operational Research and Management Cybernetics.* John Wiley & Sons, New York, 1966.

[7] Butler JT. Multiple-valued logic. *IEEE Potentials,* 14(2):11–14, 1995.

[8] Carbone A and Seeman NC. Circuits and programmable self-assembling DNA structures. In *Proceedings of National Academy of Sciences*, 99(20):12577–12582, 2002.

[9] Chakradhar ST, Agrawal VD, and Bushnell ML. *Neural Models and Algorithms for Digital Testing*. Kluwer, Dordrecht, 1991.

[10] Chen J, Lee T, Su J, Wang W, Reed MA, Rawlett AM, Kozaki M, Yao Y, Jagessar RC, Dirk SM, Price DW, Tour JM, Grubisha DS, and Bennett DW. Molecular electronic devices, In Reed MA and Lee L, Eds., *Handbook of Molecular Nanoelectronics*, American Science Publishers, New York, 2003.

[11] Collier CP, Wong EW, Belohradsk M. Raymo FM, Stoddart JF, Kuekes PJ, Williams RS, and Heath JR. Electronically configurable molecular-based logic gates. *Science*, 285:391–394, July, 1999.

[12] Crawley D, Nikolić K, and Forshaw M, Eds., *3D Nanoelectronic Computer Architecture and Implementation*. Institute of Physics Publishing, UK, 2005.

[13] Das B and Abe S. Modeling molecular switches: A flexible molecule anchored to a surface. In Seminario JM, Ed., *Molecular and Nano Electronics: Analysis, Design and Simulation*, pp. 141–162, Elsevier, Amsterdam, 2007.

[14] de Castro LN. *Fundamentals of Natural Computing: Basic Concepts, Algorithms, and Applications*. Chapman & Hall/CRC Taylor & Francis Group, Boca Raton, FL, 2006.

[15] DeHon A. Array-based architecture for FET-based, nanoscale electronics. *IEEE Trans. Nanotechnology*, 2(1):23–32, 2003.

[16] Fredkin E and Toffoli T. Conservative logic. *Int. J. Theor. Phys.*, pp. 171–182, 2003.

[17] Ellenbogen JC and Love JC. Architectures for molecular electronic computers: Logic structures and an adder designed from molecular electronic diodes. *Proceedings IEEE*, 88(3):386–426, 2000.

[18] Hanyu T, Kameyama M, and Higuchi T. Prospects of multiple-valued VLSI processors. *IEICE Trans. Nanotechnology*, E76-C(3):383–392, 1993.

[19] Hanyu T. Challenge of a multiple-valued technology in recent deep-submicron VLSI. In *Proc. 31st IEEE Int. Symp. on Multiple-Valued Logic,* pp. 241–247, 2001.

[20] Karpovsky MG, Stanković RS, and Astola JT. *Spectral Logic and Its Applications for the Design of Digital Devices*. John Wiley & Sons, Hoboken, NJ, 2008.

[21] Keymeulen D. Self-repairing and tuning reconfigurable electronics: real world applications. In *Proc. NASA/ESA Conf. on Adaptive Hardware and Systems,* 2008.

[22] Kumar VKP and Tsai Y-C. Designing linear systolic arrays. *J. Parallel and Distributed Computing*, 7:441–463, 1989.

[23] Kung HT and Leiserson CE. Systolic arrays (for VLSI). In *Sparse Matrix Proceedings*. SIAM, Philadelphia, pp. 256–282, 1978.

[24] Kung SY. *VLSI Array Processors*. Prentice Hall, Englewood Cliffs, New York, 1988.

[25] Lent CS, Tougaw PD, Porod W, Bernstein GH. Quantum cellular automata. *Nanotechnology*, 4:49–57, 1993.

[26] Lyshevski SE. *Molecular Electronics, Circuits, and Processing Platforms*. CRC Press, Boca Raton, FL, 2007.

[27] Lyshevski SE. 3D multi-valued design in nanoscale integrated circuits. In *Proc. 35st IEEE Int. Symp. on Multiple-Valued Logic,* pp. 82–87, 2005.

[28] Macii E and Poncino M. An application of Hopfield networks to worst-case power analysis of RT-level VLSI systems. *Int. J. Sci.*, 35(8):783–792, 1997.

[29] Ma Y and Seminario JM. Analysis of programmable molecular electronic systems. In Seminario JM, Ed., *Molecular and Nano Electronics: Analysis, Design and Simulation*, pp. 96–140, Elsevier, Amsterdam, 2007.

[30] Negrini R and Sami MG. *Fault Tolerance Trough Reconfiguration in VLSI and WSI Arrays*. The MIT Press, Cambridge, MA, 1989.

[31] Peper F, Lee J, Abo F, Isokawa T, Adachi S, Matsui N, and Mashiko S. Fault-tolerance in nanocomputers: a cellular array approach. *IEEE Trans. Nanotechnology*, 3(1):187–201, 2004.

[32] Porod W, Lent CS, Toth G, Csurgay A, Huang Y-F, and Liu RW. *IEEE Abstracts*, p. 745, 1997.

[33] Reichert J, Ochs R, Beckmann D, Weber HB, Mayor M, and Lohneysen HV. Driving current through single organic molecules. *Physical Review Letters*, 88(17), 2002.

[34] Rosenblatt F. *Principles of Neurodynamics*. Spartan, New York, 1962.

[35] Rothemund PWK, Paradakis N, and Winfree E. Algorithmic self-assembly of DNA Sierpinski triangles. *PloS Biology* — www.plosbiology.org, 2(12,e424):2041–2053, 2004.

[36] Shmerko VP and Yanushkevich SN. Three-dimensional feedforward neural networks and their realization by nano-devices. *Artificial Intelligence Review, An Int. Science and Eng. J.* (UK), Special Issue on *Artificial Intelligence in Logic Design*, 20(3-4):473–494, 2004.

[37] Shukla SP and Bahar RI, Eds., *Nano, Quantum and Molecular Computing*. Kluwer, Dordrecht, 2004.

[38] De Silava AP, McClenaghan ND, and McCoy CP. Molecular logic systems. In Feringa BL, Ed., *Molecular Switches*, pp. 339–361, Wiley-VCH, Weinheim, Germany, 2001.

[39] Toth G, Lent CS, Tougaw PD, Brazhnik Y, Weng WW, Porod W, Liu RW, and Huang YF. Quantum cellular neural networks. *Superlattices and Microstructures*, 20,:473–478, 1996.

[40] Tour JM and James DK. Molecular electronic computing architectures. In Goddard WA, Brenner DW, Lyshevski SE, and Iafrate GJ, Eds., *Handbook of Nanoscience, Engineering and Technology*, pp. 4.1–4.28, CRC Press, Boca Raton, FL, 2003.

[41] Tour JM. *Molecular Electronics: Commercial Insights, Chemistry, Devices, Architecture and Programming*. World Scientific, Hackensack, NJ, 2003.

[42] Tour JM, Zandt WLV, Husband CP, Husband SM, Wilson LS, Franzon PD, and Nackashi DP. Nanocell logic gates for molecular computing. *IEEE Trans. Nanotechnology*, 1(2):100–109, 2002.

[43] Wang W, Lee T, Kretzschmar I, and Reed MA. Inelastic electron tunneling spectroscopy of an alkanedithiol self-assembled monolayer. *Nano Letters*, 4(4):643–646, 2004.

[44] Yao X and Higuchi T. Promises and challenges of evolutionary hardware. *IEEE Trans. Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 29(1):87–97, 1999.

[45] Yanushkevich SN, Shmerko VP, and Lyshevski SE. *Logic Design of Nano ICs*, CRC Press, Boca Raton, FL, 2004.

[46] Yanushkevich SN. Logic design of computational nanostructures. *J. Computational and Theoretical Nanoscience*, 4(3):384–407, 2007.

[47] Yanushkevich SN. Spatial systolic array design for predictable nanotechnologies. *J. Computational and Theoretical Nanoscience*, 4(3):467–481, 2007.

[48] Yanushkevich SN, Shmerko VP, and Steinbach B. Spatial interconnect analysis for predictable nanotecnologies. *J. Computational and Theoretical Nanoscience*, 5(1):56–69, 2008.

[49] Yanushkevich SN, Miller DM, Shmerko VP, and Stanković RS. *Decision Diagram Techniques for Micro- and Nanoelectronic Design*, Taylor & Francis/CRC Press, Boca Raton, FL, 2006.

[50] Ziegler MM and Stan MR. CMOS/nano co-design for crossbar-based molecular electronic systems. *IEEE Trans. Nanotechnology*, 2(4):217–230, 2003.

2

Computational Nanostructures

2.1 Introduction

In this chapter, the basic principles for design of a computational structure are introduced. These principles, such as analysis and synthesis, assembling, top-down and bottom-up methodology, design styles, simulation, and modeling, can be adopted in nanodevices design, except for some particular cases when underlying physical and chemical phenomena address nontraditional approaches. This chapter also briefly introduces the computational models such as switch-based computing and homogeneous structures that are the focus of further chapters in this book.

The crucial point of computing nanostructure design is an approach that provides the delegation of computing abilities from initial data structure to the nanostructure. This approach is based on the specification of the form of graphical data structures, such as decision trees and diagrams. These data structures are characterized by computational properties that can be embodied in various spatial structures specified by nanotechnology. This chapter provides a brief introduction of this problem. Another aspect is that hybrid technologies can be used in computing structure design. This aspect requires using the unified approaches in design methodologies.

A *discrete system* is a combination of logic networks and discrete devices that is assembled to accomplish a desired result, such as the computing and transferring of data. Digital logic networks are used in all devices that process information in digital form. *Information* can be defined as recorded or communicated facts, or *data*. Information takes a variety of physical forms when being stored, communicated, or manipulated. Information on the nature of a physical phenomenon is conveyed by signals that assume a finite number of discrete values, that is, it is expressed as a finite sequence of symbols. A *signal* is defined as a function of one or more variables, and a *system* is defined as an entity that manipulates one or more signals to accomplish a function, thereby yielding new signals.

In this chapter, each signal is assumed to have only one of two values, denoted by the symbols 0 and 1. If the signals are constrained to only two values, the system is *binary*.

2.2 Theoretical background

Boolean algebra was introduced by George Boole in 1854 and applied by C. Shannon in 1938 to relay-contact networks, the first switching circuits. This theory is called *switching theory* and has been used ever since in the design of digital *logic networks* or *logic circuits*. Since then the technology has gone from relay-contacts through diode gates, transistor gates, and integrated circuits, to future nanotechnologies, and still Boolean algebra is its fundamental and unchanging basis. Modern logic design includes methods and techniques from various fields. In particular, digital signal processing is adopted in logic design for efficient manipulation of data; communication theory solves communication problems between computing components in logic networks; artificial intelligence methods and techniques are used for optimization at logic and physical levels of logic network design.

As far as predictable technologies are concerned, nontraditional computing paradigms that are based on various physical and chemical phenomena are studied. The assumed stochastic nature of many processes at nano scale implies that *random* signals must be used instead of *deterministic* signals. Random signals take on random values at any given moment in time and must be modeled probabilistically, while deterministic signals can be modeled as completely specified functions of time. The theoretical base of probabilistic logic signals is called *probabilistic logic*.

A unit for computing an elementary Boolean function (such as an AND, OR, NAND, NOR, EXOR, threshold function, or a special case of it, the majority function) at nano scale is defined as a *logic nanocell*. The elementary logic functions and arbitrary logic networks can be implemented by using only threshold (neuron) cells. A *switch* is the simplest logic operation that operates on a single bit. An arbitrary Boolean function (and, therefore, an arbitrary logic network) can be implemented using, for example, only NAND or only NOR nanocells, or only switches. If nanocells are affected by noise that influences their dynamic and steady-state behavior, they are referred to as *probabilistic nanocells*.

The classic computing paradigm implemented in most of today's computers is known as *von Neumann* architecture. In the computing paradigms based on the principle of *distributed* processing, information is encoded into the different components of a computing network. In *fault-tolerant* computing, the effects of faults are mitigated, and correct computations are guaranteed with a certain level of reliability. *Redundancy* is one of the fault-tolerant techniques that can be used if additional resources are available. Redundancy in distributed models cannot be achieved by integration of extra cell copies over the appropriate permutation and decision profile, but only by sharing of carriers of information between many components of the network. Assuming reliable computing in the presence of faults is called *fault-tolerance*; for

example, approaches to reliable computing in networks with faulty nodes and/or interconnects, including error correction codes and residual number systems. These approaches state that it is possible to correct a class of faults if a library of *reliable* logic nanocells for implementing the correction is available.

2.3 Analysis and synthesis

The *specification* of a system is defined as a description of its function. The *implementation* of a system refers to the construction of a system. The *analysis* of a system has its objective to determine its specification from the implementation:

$$\underbrace{\texttt{Logic network} \longrightarrow \texttt{Specification}}_{Analysis}$$

Synthesis, or *design*, consists of obtaining an implementation that satisfies the specification of the system:

$$\underbrace{\texttt{Specification} \longrightarrow \texttt{Logic network}}_{Synthesis}$$

The central task in logic synthesis is to optimize the representation of a logic function with respect to various criteria.

Complex systems are specified at various levels of detail. At each level, the units of complexity are specified as *components* or a *subsystem* of the components, and a *basic element, or primitive* is defined as a component that has no internal structure.

A system can be examined at various levels of abstraction. Each such level is called a *design level* (Figure 2.1a). The following design levels are distinguished:

- The top design level is called the *architectural* or *system* level.
- The intermediate design level is called the *logic* level; this level is the subject of the present book.
- The bottom design level is called the *physical* level; this level is concerned with the details needed to manufacture or assemble the system.

At the physical level, a system is implemented by a complex interconnection of simplest elements (primitives). Because of high complexity, it is impractical to perform design and optimization at this level, motivating a move to the

intermediate level of design. At the intermediate level, a modular structure provides a reasonable simplification of design. *Libraries* of standard modules significantly simplify the design of different systems. Assembling modules of increasing complexity into higher hierarchical blocks is achieved at the system level.

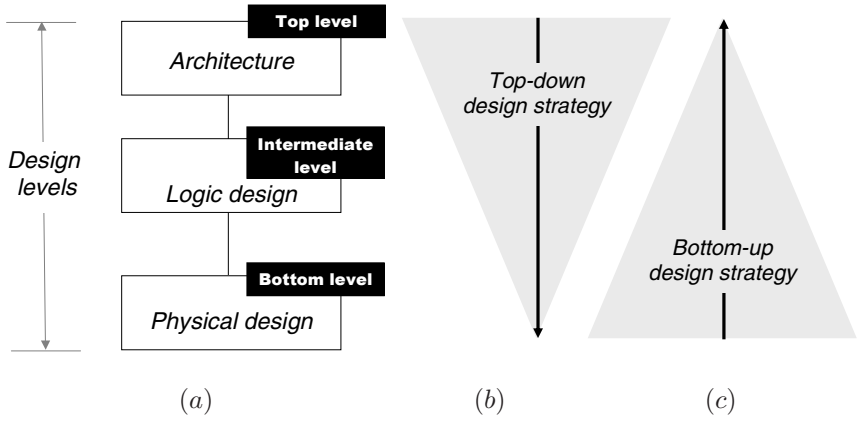


FIGURE 2.1
Design hierarchy (a), top-down (b), and bottom-up (c) design strategies.

2.3.1 Design hierarchy

A *hierarchical* approach to digital system design aims at reducing the cost of the design of a system, and improving the quality of the obtained solutions. The hierarchical approach to design makes a large system more manageable by reducing complexity and introducing a rational *partitioning* of the design processes. They can be designed, tested, and manufactured separately. This is the basis for *standardization*. The specified components can be mass produced at relatively low cost. In the design process, these components can be composed in standard *libraries* and reused with minor modifications. This facilitates the reduction of overall design time and cost. The *robustness* of the hierarchical approach provides many possibilities for avoiding design errors, design corrections, and repairs after manufacture.

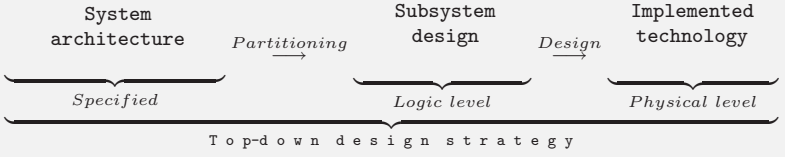
Any design process includes a *design loop* that provides the possibility to carry out a *redesign* if errors are detected in simulation. This loop is repeated until the simulation indicates a successful design.

Example 2.1 *The use of standard cells is a classical example of how a restriction in the design space (a limited library of constrained cells) makes it possible to use intellectual and computational capabilities to find high-quality acceptable solutions through synthesis.*

2.3.2 Top-down design methodology

Design methodology is a system of ways of obtaining the implementation of a specified design.

A design that evolves from a generalized or abstract point of view and proceeds in steps to specific components is referred to as a *top-down design methodology* (Figure 2.1b):



In this approach, the hierarchy tree is traversed from top to bottom. The system architecture is specified at the highest level first. The disadvantage of this approach is that no systematic procedure exists for optimization of the final implementation; that is, optimization at one particular level does not guarantee an optimal final solution. The success of the approach depends mainly on the experience and professional skills of the designer.

Example 2.2 *Given the architecture of a system, a designer may begin to detail subsystems in terms of small black boxes, each of which consists of an electronic circuit developed by "circuit designers." The designer has to know, in particular, (a) the logical function of each block, have a means be able of expressing those functions, and (b) the logic function the interconnected blocks provide.*

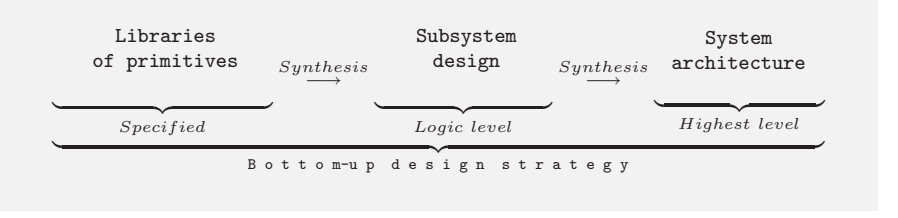
The top-down approach is currently used in the silicon industry, wherein small features such as transistors are etched into silicon using resisters and light. This hierarchy starts at the highest level of abstraction, the architecture level. Then it descends to the level of its component circuits, and finally to the level of the component switch and interconnect devices.

An example of an advanced top-down methodology is so-called *platform-based design*. A platform is defined as a family of the designs and not a single design. In this top-down process, the constraints that accompany the specification are mapped into constraints on the components of the platform.

Most artificial intelligence techniques based upon the top-down design paradigm are known as knowledge-based or expert systems.

2.3.3 Bottom-up design methodology

An alternative process to the top-down approach is a *bottom-up design methodology* (Figure 2.1c):



This is the reverse of the top-down design process. One starts with specific components in mind and proceeds by interconnecting these components into a generalized system. In a bottom-up design approach, the components at or near the lowest design level of the hierarchy tree are designed first. The architecture of the entire system is not specified until the top of the tree is reached. Unfortunately, in general, there is no systematic technique that results in correct system specification.

Bottom-up systems and models are those in which the global behavior emerges from the aggregative behavior of ensemble of relatively simple elements acting solely on the basis of local information.

Example 2.3 *Natural computing inspired or based on biology is more rooted in* **bottom-up** *approaches. An example of bottom-up design is artificial neural networks. Their structure is not defined a priori; they are a result of the network interactions with the environment. For example, if the initial neural network has a single neuron, and more neurons are can be added until the network is capable of appropriately solving the problem.*

The bottom-up approach implies the construction of functionality into small features, such as molecules, with the opportunity to have the molecules further self-assemble into higher-ordered structural units such as transistors. Bottom-up methodologies are quite natural in that all systems in nature are constructed bottom-up.

Example 2.4 *Molecules with specific features assemble to form higher-order structures such as lipid bilayers.*

Molecular electronics proponents believe purposeful bottom-up design will be more efficient than the top-down method.

Example 2.5 *The device physics, based on quantum interaction, quantum interference, quantum transition, vibration, Coulomb effect, and other phenomena and effects (electron spin, photon-electron-assisted transitions, etc.), must be coherently complemented by the bottom-up synthesis of the molecular aggregates that exhibit those phenomena.*

However, in practice, there are many specific-area application devices and systems where top-down and bottom-up design methodologies can be efficiently used separately or in various combinations. This provides the possibility for designing simultaneously in several levels.

2.3.4 Design styles

In a general sense, the following design styles are distinguished:

Full custom design: This style provides freedom to the designer and is characterized by great flexibility; however, this style is not acceptable for the design of large systems.

Semi-custom design: Provides more possibilities for automation using, in particular, standardization; such as a *library* of standard cells.

Mixed design styles: Often provide an acceptable reduction to the flexibility of full-custom style, while opening up possibilities for the automation and optimization of semi-custom style.

Gate-array design meets the requirements of fabrication and simplifies the optimization problem; this style results in regular structures within the chip, that is, connected cells that are placed on a chip in a regular way.

These design styles are adopted in various technologies.

2.3.5 Modeling and simulation

A unit for computing elementary logic (Boolean) functions (such as an AND, OR, NAND, NOR, EXOR, threshold function, or the majority function) at nano scale is defined as a *logic nanocell*. The elementary logic functions and arbitrary logic networks can be implemented by using only threshold (neuron) cells. A *switch* is the simplest logic operation that operates on a single bit.

An arbitrary elementary logic function (and, therefore, an arbitrary logic network) can be implemented using, for example, only NAND or only NOR nanocells, or only switches. If nanocells are affected by noise that influence the dynamic and steady-state behavior, they are referred to as *probabilistic nanocells*.

The classic computing paradigm implemented in most of today's computers is known as *von Neumann* architecture. In the computing paradigms based on the principle of *distributed* processing, information is encoded into the different components of a computing network. In *fault-tolerant* computing paradigms, the effects of faults are mitigated, and correct computations are guaranteed with a certain level of reliability. *Redundancy* is one of the fault-tolerant techniques that can be used if additional resources are available. Redundancy in distributed models cannot be achieved by integration of extra cell copies over the appropriate permutation and decision profile, but only by sharing of carriers of information between many components of the network. Assuming reliable computing in the presence of faults is called *fault-tolerance*, which includes approaches to reliable computing in networks with faulty nodes and/or interconnects, error correction codes, and residual number systems, etc. These approaches state that it is possible to correct a class of faults if a library of *reliable* logic nanocells for implementing the correction is available.

Recursive stochastic models are based on the property of *relaxation* that is, an ability to relax into a stable state in the absence of external input. Relaxation is referred to as the *embedding* of a correct solution into a network of logic nanocells (neurons). In models based on *stochastic pulse stream encoding*, information is encoded in the average pulse rate or primary statistics of this stream.

The use of software simulation is an important part of any modern design process. The primary uses of a simulator are to check a design for functional correctness and to evaluate its performance. Simulators are key tools in determining whether design goals have been met and whether redesign is necessary.

2.4 Implementation technologies

The scaling of microelectronics down to nanoelectronics is the inevitable result of technological evolution (Figure 2.2). The most general classification of the trends in technology is based on grouping computers into *generations*. Using this criterion, five generations of computers are distinguished (Table 2.1). Each computer generation is 8 to 10 years in length.

The following can be compared against this scale:

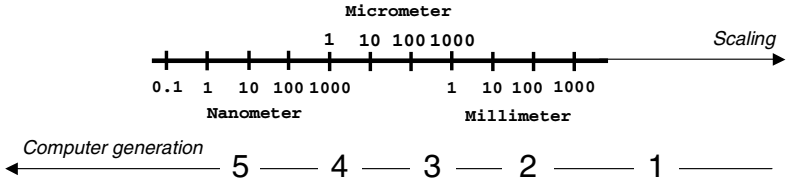


FIGURE 2.2
Progress from micro- to nanosize in computing devices.

TABLE 2.1
Computer generations are determined by the change in the dominant technology

Generation	Dates	Technology
1	1950–1964	Vacuum tubes (zero-scale integration)
2	1965–1969	Transistors (small-scale integration)
3	1970–1979	Integrated circuits (medium-scale integration)
4	1980–2004	Large, very large, ultra large-scale integration
5	2005–	Nanotechnology (giga-scale integration)

The scaling of microelectronics down to nanoelectronics

- The size of an atom is approximately 10^{-10} m. Atoms are composed of subatomic particles; e.g., protons, neutrons and electrons. Protons and neutrons form the nucleus, with a diameter of approximately 10^{-15} m.
- 2D molecular assembly (1 nm).
- 3D functional nanoICs topology with doped carbon molecules ($2 \times 2 \times 2$ nm).
- 3D nanobioICs (10 nm).
- *E.coli* bacteria (2 mm) and ants (5 mm) have complex and high-performance integrated nanobiocircuitry.
- 1.5×1.5 cm 478-pin Intel® Pentium® processor with millions of transistors, and Intel 4004 Microprocessor (made in 1971 with 2,250 transistors).

Binary states are encountered in many different physical forms in various

technologies. The standard approach is to use the digit symbols 0 and 1 to represent the two possible values of binary quantity. These symbols are referred to as *bits*.

Binary arithmetic has the following advantages:

(*a*) It can be implemented using on-off switches, the simplest binary devices.

(*b*) It provides for the simplest decision-making such as YES (1) and NO (0).

(*c*) Binary signals are more reliable than those formed by more than two quantization levels.

Significant evolutionary progress has been achieved in microelectronics. This progress (miniaturization, optimal design and technology enhancement) has been achieved by scaling down microdevices, approaching 45 nm sizing features for structures, while increasing the integration level (Figure 2.3). Complementary metal-oxide semiconductor (CMOS) technology is being enhanced, as nanolithography, advanced etching, enhanced deposition, novel materials, and modified processes are all currently used to fabricate ICs.

Example 2.6 *The channel length of metal-oxide-semiconductor field effect transistors (MOSFETs) has decreased from*

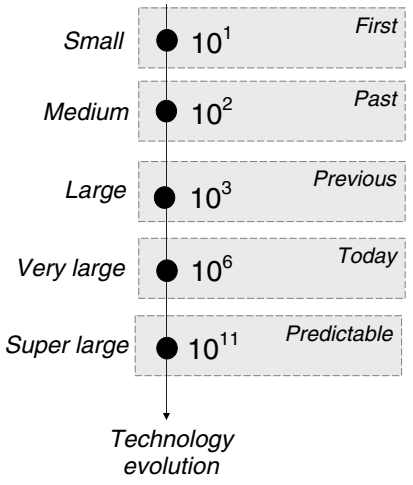
- *50 μm in 1960, to*
- *1 μm in 1990, and to*
- *130 nm in 2004, 65 nm in 2006, and 45 nm in 2007.*

Example 2.7 *The progress in miniaturization and integration can be observed, for example, on Intel processors:*

1971–1982: From Intel 4004 (1971, 2,250 transistors), to Intel 286 (1982, 120,000 transistors),

1993–2007: From Pentium (1993, 3,100,000 transistors), to Pentium 4 (2000, 42,000,000 transistors), to Itanium™ 2 processor (2002), Pentium® M processor (2003) with hundreds of millions of transistors, and Pentium Dual-Core in 2007.

The increases in packing density of the circuitry are achieved by shrinking the linewidths of the metal interconnects, by decreasing the size of other features, and by producing thinner layers in the multilevel device structures.



Small-scale integration (SSI):

- *1960s, dozens of gates in a package*

Medium-scale integration (MSI):

- *1970s, hundreds of gates in a package*

Large-scale integration (LSI):

- *1980s, thousands of gates in a package*

Very large-scale integration (VLSI):

- *1980s* and *1990s, hundreds of thousands of gates in a package*

Giga-scale integration:

- *Today, millions of gates in a package*

Tera-scale integration:

- *Expected, hundred millions of gates in a package*

FIGURE 2.3
Evolution of technologies: levels of integration of chips.

Example 2.8 *Commercial metal interconnect linewidths have decreased to 0.1 mm.*

Typical signal integrity effects include interconnect delay, cross-talk (in closely coupled lines the phenomenon of cross-talk can be observed), power supply integrity, and noise-on-delay effects. In the early days of very large-scale integration (VLSI) design, these effects were negligible because of relatively slow chip speed and low integration density. However, with the introduction of technological generations working at about 0.25 μm scale and below, there have been many significant changes in wiring and electrical performance.

As the number of computational and functional units on a single chip increases, the need for communication between those units also increases. Interconnection has started to become a dominant factor in chip performance. As chip speed continually increases, the increasing inductance of interconnections affects the signal parameters. The length of interconnect lines when measured in units of wire pitch increases dramatically.

Noise is a deviation of a signal from its intended or ideal value. Most noise in VLSI circuits is created by the system itself. Electromagnetic interference is an external noise source between subsystems. In deep submicron circuits, the noise created by parasitic components with digital switching exhibits the strongest effect on system performance. Noise has two deleterious effects on

circuit performance: (a) when noise acts against a normal static signal, it can destroy the local information carried by the static node in the circuit and ultimately result in incorrect machine-state stored in a latch, and (b) when noise acts simultaneously with a switching node, this is manifest as a change in the timing of a transient.

2.5 Predictable technologies

The key to the predictable technologies of the future is changing the *computing paradigm*, that is, the model of computation and the physical effects for the realization of this model:

$$\underbrace{\texttt{Computing model}}_{Design} \overset{Implementation}{\longrightarrow} \underbrace{\texttt{Physical effects}}_{Technology}$$

Silicon loses its original band structure when it is restricted to very small sizes. The lithography techniques used to create the circuitry on the wafers also neared its technological limits.

There are fundamental technological differences among

- Nanoelectronic devices versus microelectronic ones (which can even be nanometers in size)
- Nanoelectronics versus microelectronics, for example, nanointegrated circuits versus integrated circuits

These enormous differences are due to differences in basic physics and other phenomena. The dimensions of nanodevices that have been made and characterized are a hundred times less than even newly designed microelectronic devices (including nanoFETs with 10 nm gate length). Nanoelectronics sizing leads to volume reduction by a factor of 1,000,000 in packaging, not to mention revolutionary enhanced functionality due to multiterminal and spatial features. For example, molecular electronics focuses on the development of electronic devices using small molecules with feature sizes on the order of a few nanometers.

Superconductive solid-state and molecular electronics focuses on the development of electronic devices using small particles and molecules with feature sizes on the order of a few nanometers. The ultimate goal of miniaturization is to use the minimum amount of atoms per electronic function.

Example 2.9 *A contemporary computer utilizes* $\sim 10^{10}$ *silicon-based devices, whereas the scaling factor gained from molecular-scale technology* $\sim 10^{23}$ *devices in a single beaker using routine chemical syntheses.*

An additional driving factor is the potential to utilize thermodynamically driven directed self-assembly of components such as chemically synthesized interconnects, active devices, and circuits.

In order to take advantage of the ultra-small size of molecules, one ideally needs an interconnect technology that (a) scales from the molecular dimensions, (b) can be structured to permit the formation of the molecular equivalent of large-scale diverse modular logic blocks as found in VLSI interconnect architectures, and (c) can be selectively connected to mesoscopically (100 nm scale) defined input/output (I/O).

Nanoscale devices must obey the fundamental constraints that the laws of physics impose on the manipulation of information. Some physical constraints, such as the fact that the speed at which information can travel optically through free space is upperbounded by a constant (the speed of light, 299 792 458 m/s), are still present in nanodimensions. The electrical transmission of signals along wires is slower than light, so the current propagation delays along dissipative wires are significant. The time of transmission is proportional to the square of the distance, unless the signals are periodically regenerated by additional buffering logic.

A wide variety of factors, such as voltage scaling and thermal noise, dramatically reduce the reliability of integrated systems. In nanotechnologies, process variations and quantum fluctuations occur in the operations of very-deep submicron transistors. Any computer with nano scale components will contain a significant number of defects, as well as massive numbers of wires and switches for communication purposes. It therefore makes sense to consider architectural issues and defect tolerance early in the development of a new paradigm.

Two main aspects are critical to the design of nanodevices. First is *the probabilistic behavior* of nanodevices (electrons, molecules), which means that a valid Boolean function can be calculated with some probability. Second is the *high defect rates* of nanodevices, meaning that because many of the fabricated devices have defects, their logic correctness is distorted.

There are two types of fault tolerances exhibited by a nanosystem: fault tolerance with respect to (a) data that are noisy, distorted, or incomplete, which results from the manner in which data are organized and represented in the nanosystem, and (b) physical degradation of the nanosystem itself. If certain nanodevices or parts of a network are destroyed, the network will continue to function properly. When the damage becomes extensive, the network will only affect the system's performance, as opposed to causing a complete failure. Self-assembling nanosystems are capable of this type of fault

tolerance because they store information in a distributed (redundant) manner, in contrast to traditional storage of data in a specific memory location in which data can be lost in the case of a hardware fault.

The methods of stochastic computing provide another approach to overcoming the problem of the design of reliable computers from unreliable elements, i.e., nanodevices.

Example 2.10 *A signal may be represented by the probability that a logic level has a value of 1 or 0 at a clock pulse. In this way, random noise is deliberately introduced into the data. A quantity is represented by a clocked sequence of logic levels generated by a random process. Operations are performed via completely random data.*

2.6 Nanoelectronic networks

The specifications of nanoelectronic devices and networks are based on data structures adopted from classical logic design. This adaptation is done based on the premises of nano scale implementation structures.

2.6.1 CMOS-molecular electronics

The chemically directed self-assembly of molecular single-electron devices on prefabricated nanowires is considered as a promising way toward integrated circuits using, for example, single-electron tunneling.

2.6.2 Neuromorphic computing paradigm

A *Hopfield* network, also known as *neuromorphic* computing,* is based on distributed architecture principles, and can be implemented by various models [25]. Using a Hopfield network, an arbitrary logic function can be computed via a process similar to the gradual cooling process of metal. A value of a logic function given an assignment of Boolean variables is computed through the relaxation of neuron cells. Hopfield networks are capable of reliable computing, despite imperfect neuron cells and degradation of their performance. This is because of degraded neuron cells store information (in weights, thresholds, and topology) in a *distributed* (or redundant)

*The term "neuromorphic systems" was used for the first time by C. Mead [32] to define an interdisciplinary approach to the design of biologically inspired information processing systems.

manner. A single "portion" of information is not encoded in a single neuron cell, but is rather spread over many. This contrasts with the traditional computing paradigm (von Neumann architecture), in which data is stored in a specific memory location. The Hopfield computing paradigm is based on the concept of *minimization of energy* in a stochastic system. This concept is implemented using *McCulloch–Pitts* neuron cells for elementary logic functions, the *interconnections* between nanocells given a logic function, the *weights* assigned to the links between nanocells, and an *objective* function given the number of neuron cells, their thresholds, interconnections, and weights. The carriers of information in the Hopfield network are a particular topology of connections between neuron cells, the weights of the links, and the neuron thresholds.

2.6.3 Interconnect

An *interconnect* is considered with respect to various criteria, for example, number of switches, signal delays, total length, power dissipation, and cost. The traditional understanding of the interconnect as a wire in a voltage-controlled network does not work for certain nanotechnologies, especially for those where the carrier of charge (for example, single electron) is considered as a messenger that travels through the network of nanowires via multiplexing, or a signal propagated by the intractability of cells. In such a network, control signals (input variables) are separated from the messenger signal (function), and the problem of interconnection becomes the problem of compatibility of inputs and outputs. Below we consider switches on multiplexers as the most suitable candidates for interconnect nodes.

The components of biomolecular networks are not connected by physical wires that direct a signal to a precise location; molecules that are the inputs and outputs of these processes share a physical space and can commingle throughout the cell. It requires the special tools to isolate signals and prevent cross-talk, in which signals intended for one recipient are received by another. Unlike electronics, which typically rely on a clock to precisely synchronize signals, these biomolecular signals are asynchronous.

Nanoscale devices must obey the fundamental constraints that the laws of physics impose on the manipulation of information. Some physical constraints, such as the fact that the speed at which information can travel optically through free space is upperbounded by a constant (the speed of light, 299 792 458 m/s), are still present in nanodimensions. The electrical transmission of signals along wires is slower than light, so the current propagation delays along dissipative wires are significant. The time of transmission is proportional to the square of the distance, unless the signals are periodically regenerated by additional buffering logic.

The vulnerability of nanocircuits to defects and faults arising from instability and noise-proneness at nanometer scales [43] leads to unreliable and undesirable results of computation. Robustness to errors is an important

design consideration for nanocomputers in the light of the noise and instabilities that affect the reliability of nanometer-scale devices.

2.6.4 Carbon-nanotube-based logic devices

Carbon nanotubes are sheets of graphite rolled into tubes. Depending on the direction in which the graphite sheet is rolled, the single-walled carbon nanotubes can be metallic or semiconductor. Semiconducting nanotubes are used for fabrication switches and transistors known as carbon-nanotube field-effect transistors (CNFETs). The CNFETs are considered as possible successors to silicon MOSFETs because CNFETs inherit current-voltage characteristics that are qualitatively to silicon MOSFETs. That is, MOS-based logic networks can be transferred to a CNFET-based design.

Storage elements are implemented as the molecular scale junction switches formed at the crosspoints of the wires. Addressing memory array (random access memory) can be designed based on the crosspoint array by using a decoder.

State-of-the art defect-tolerant techniques can be applied to the crossbar memory. The crossbar memory does not operate correctly if defects occurred during the fabrication. These defect can be located and identified. Using redundant rows and columns, the faults caused by the identified defects can be eliminated.

Example 2.11 *The following faults caused by the defects can be used in defect tolerance techniques: (a) broken nanowire, (b) stuck-at-low and stuck-at-high resistance state, (c) short at a crosspoint, and (d) open at a crosspoint.*

2.6.5 Crossbar-based computing structures

The *crossbar* structures can be designed using the bottom-up approach. Various physical phenomena can be utilized in crossbar structure design.

Example 2.12 *The following physical implementation can be utilized in crossbar structure design:*

- *Self-assembled nanowires*
- *Nano-imprinted wires*
- *Nanotubes*

Computing properties are delegated in crossbar structure using (a) crosspoint function and (b) topological properties such as distribution of this

function in a rectangular array. The simplest function of the crosspoint is switch.

Example 2.13 *Functions of a crosspoint in a crossbar structure can be switches. The advantage of this array is that an arbitrary Boolean function can be directly assembled. The drawback is that the signal is degradated with respect to the number of active switches.*

The *crossbar* computing structures utilize the common paradigm for molecular electronics. The crossbar structure has a simple interconnection topology and is therefore attractive for fabrication. Different types of crossbar address different types of elements such as resistor, diode, or transistor. These elements can be combined to perform larger functions.

Example 2.14 *The basic crossbar structure consists of the combination of planes of parallel wires laid out in orthogonal directions.*

Each junction or crosspoint within the crossbar can be independently configured. After fabrication, the bistable junctions at the wire crossing can be customized, that is programmed to perform the designed Boolean function.

2.6.6 Noise

The term *noise* means undesirable variations of voltages and currents in a circuit. Noise is an unpredictable and random phenomenon. The noise become significant in a logic network when the noise signal causes incorrect logic functions. Noise in integrated circuits is the interference signal inducted by signals on neighboring wires or from the substrate. This noise is different from the intrinsic noise generated by active device components. Most CMOS digital circuits have a relatively high immunity to noise. However, as power supply levels have decreased, this property has diminished. Both the power supply distribution and signal distortion along coupled interconnect lines are interconnect-related problems. Interconnections in CMOS integrated circuits are multiconductor lines existing on different physical planes. The parasitic impedances of these conductor lines can be extracted from the geometric layout. The coupling capacitance is physically a fringing capacitance between neighboring interconnect lines, and strongly depends upon the physical structure of the adjacent interconnections.

Example 2.15 *For parallel metal lines on the same layers, the fringing capacitance will increase as the spacing between the interconnections and the thickness-to-width aspect ratio of the interconnection increases.*

Due to the screening effect of low-level interconnect, the metal-to-metal coupling capacitance among different layers also contributes to the total coupling capacitance. Coupling effects become more significant as the feature size is decreased. The induced noise voltage may cause extra power to be dissipated on the quiet line due to momentary glitches at the output of the logic gates.

Example 2.16 *At gigahertz operating frequencies and high integration densities, power dissipation densities are expected to approach $20W/cm^2$, a power density limit for an aircooled package devices. Such a power density is equivalent to 16.67 ampers of current for 1.2V power supply in a 0.1μm CMOS technology.*

2.7 Switch-based computing structures

Decision trees and diagrams are the graphical data structures, which correspond to the networks of switches. Therefore, they are called *switch-based computing structures*. This data structures are useful for two classes of problems: (a) for computing Boolean functions and (b) for embedding computational properties into spatial structures.

2.7.1 Switches

A *switch* is any device by means of which two or more conductors (electrical, electrochemical) can be conveniently connected or disconnected. The status of the contact, which can be opened or closed, can be represented by a variable x_i. A switch is the simplest computing device that operates on a single bit. Computations with n bits can be realized as compositions of such one-bit switching changes. Switches are the basic elements for construction of logic gates.

Switches have two states: *operated* or *active*, and *released* or or *inactive*. There are many different types of switches, but they fall into two general classes: *nonlocking* switches, which return to a normal state when released, and *locking* switches, which retain their state after the controlling level is released.

Example 2.17 *A NOT gate can be implemented as a single-bit switch.*

A switch whose level is operated by a molecular phenomenon is called a *molecular switch*. The state of the molecular switch "contacts" when the switch is not energized (activated), is called the *normal state* of the switch. Thus, a molecular switch can have both *normally open* and *normally closed* states.

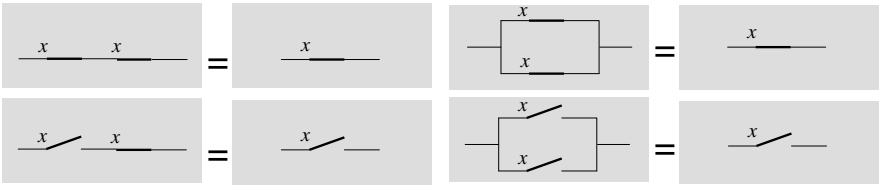
Switches might be a clue to the resolving of the poor input–output gain in the device-centered (based on the library of logic gates) design considered above. Many nanocircuits, such as arrays of nanowires, charge state logic, and other devices, are nonuniliteral, i.e., they have no clearly distinguished input and output voltage. In terms of logic design, these circuits act as decision diagram-based pass-gate logic, or switch (simplest multiplexer)-based devices.

A Boolean function realized by a switch-based network has the value of 1 if there is closed path between two specified terminals of the logic network. The switch themselves are assumed to be operated by the binary variables of the Boolean function. A variable x_i and $\overline{x}_i$ may be associated with a switch that is closed if and only if $x_i = 1$ and $x_i = 0$, respectively. Switches associated with uncomplemented and complemented variables are assumed to be normally open ($x_i = 0$) and closed ($x_i = 1$), respectively.

Example 2.18 *When two switches are connected in series, the path through them is closed only when both of them are closed. This condition can be described by the equation* $x_1 \times x_2 = 1$, *where* x_1 *and* x_2 *are the binary variables associated with these two switches. In the parallel connection of switches, there is a closed path between the terminals if either switch is closed. Figure 2.4 shows the series and parallel connections of switches.*

Probabilistic switching paradigm is the base of models for the energy saving computing in contemporary logic design of discrete devices. Based on this model, probabilistic logic gates can be constructed. In these models, noise is considered as a resource to achieving the accurate energy and power characteristics of discrete devices.

A switch operates in a single bit. It is assumed that the minimum energy needed to compute a bit with error of $1-p$ by idealized probabilistic switch is $kt \ln 2p$ Joules, where k is the Boltzmann's constant, and t is the temperature of the thermodynamic system.



(*a*) (*b*)

FIGURE 2.4
Two switches are connected in series; the closed path through them is closed only when both of them are closed (a). Two switches are connected in a parallel connection of switches; the path between the terminals is closed if either switch is closed (b) (Example 2.18).

2.7.2 Switch-based networks represented by decision diagrams

A decision diagram is a rooted directed acyclic graph consisting of

- The root node
- A set of terminal (constant) nodes
- A set of nonterminal nodes

connected with directed edges (links) (Figure 2.5). These components make up a *topology* of decision trees and diagrams. The topology is characterized by a set of topological parameters such as size, number of nonterminal nodes, number of levels, and number of links.

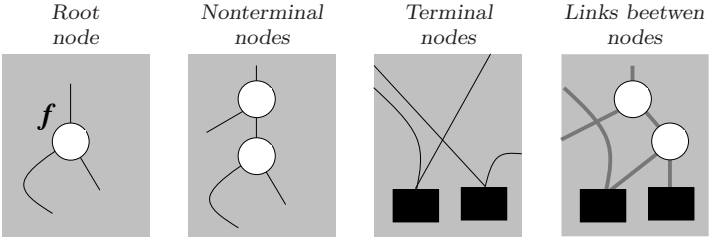


FIGURE 2.5
Components of a decision diagram.

Decision diagrams are graph-based data structures that represent algebraic ones. The algebraic data structure can be mapped into a decision tree or diagram, and vice versa.

Example 2.19 *Figure 2.6 shows the implementation of decision diagrams for the AND Boolean function using multiplexers.*

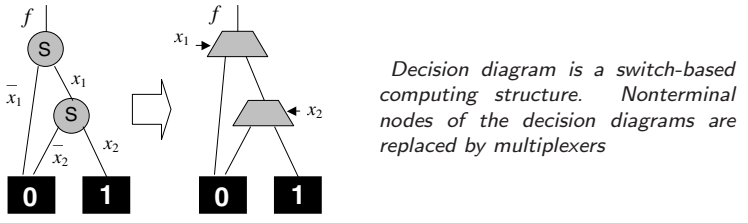


FIGURE 2.6
Mapping of decision diagram into the multiplexer-based logic network (Example 2.19).

Manipulation of algebraic representations is based on the rules and axioms of Boolean algebra. These are aimed at simplification, factoring, composition, and decomposition. Manipulation of graphical data structures is based on the rules for reduction of nodes and links in decision trees and diagrams. There is a mapping between these representations. Algebraic expressions are relevant to the corresponding graphical data structures (hypercubes and hypercube-like structures, decision trees and diagrams, etc.)

Example 2.20 *Figure 2.7 shows the relationship between algebraic and graphical data structures for a 3-input NOR function. The algebraic form (a) is mapped into decision tree (b), which is reduced to the decision diagram (c).*

If the relationship between the basic rules of algebraic and graphical data structures established, an arbitrary logic function can be mapped into a corresponding decision diagram and vice versa. Because of the variety of algebraic structures and techniques for their synthesis, a wide variety of diagram types have been developed in the last four decades.

The implementation costs of decision trees and diagrams can be estimated directly from their topology. The other important properties, which can be evaluated on the diagram or tree, are a probability distribution that can be specified on the variables' spaces (which can be assumed uniform if not otherwise known) and information-theoretic measures that also use probability distributions. In Figure 2.8, the information-theoretic model of decision diagrams is given.

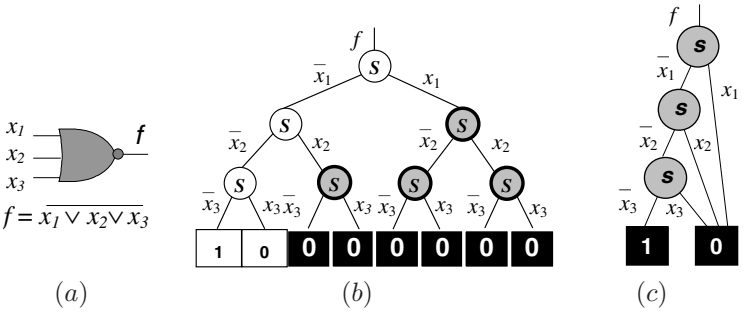
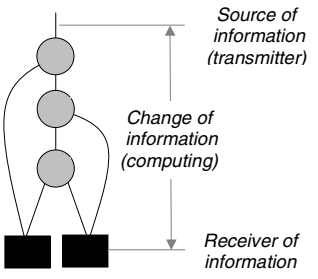


FIGURE 2.7
The 3-input NOR function (a), decision tree (b), and decision diagram (c) (Example 2.20).



The information-theoretic model includes:

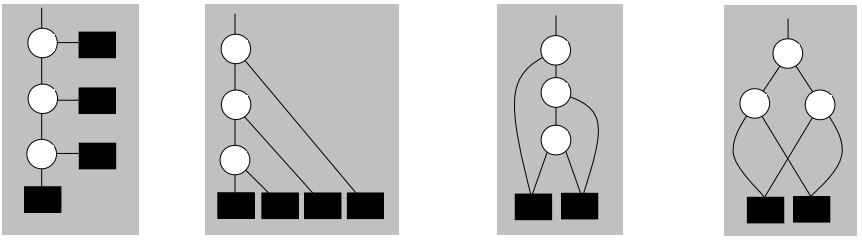
- *The source or transmitter of information, which corresponds to the input of the root node*
- *The channel where the information is transmitted. In a decision diagram, this corresponds to processing (changing) of information by nonterminal nodes*
- *The receiver of information, which corresponds to the terminal nodes of a decision diagram*

FIGURE 2.8
Information-theoretic model of a decision diagram.

The topology of decision trees and diagrams can be changed by the manipulation of the algebraic representation or by direct manipulation of the graphical structure.

Example 2.21 *Manipulation of algebraic expression and graphical structure implies various topologies of decision diagrams (Figure 2.9). The topologies shown in Figure 2.9a,b,c are classified as planar topologies of decision diagrams (useful in the design of networks without crossings).*

Reduction of decision trees is aimed at simplification with respect to various criteria, in particular, reduction of the number of nonterminal nodes, satisfaction of topological constraints to meet the requirements of implementation, and reduction of the length of paths from root to terminal nodes.



(*a*) (*b*) (*c*) (*d*)

FIGURE 2.9
Various topologies of decision diagrams: (a) the topology is achieved by linearization of an algebraic expression of a Boolean function; (b) the topology is derived by linearization of an algebraic expression and manipulation (composition) of graphical elements; (c,d) the topologies are achieved by using various representations of Boolean functions and their symmetric properties (Example 2.21).

A path in the decision tree consists of the edges connecting a nonterminal node with a terminal node. A complete path connects the root node with a terminal node. Decision trees and diagrams are graphical representations of functional expressions. For instance, each complete path in a binary decision tree or diagram corresponds to a term in sum-of-products expression of a function f.

2.8 Spatial computational nanostructures

It has been justified in a number of papers that 3D topologies of computer elements are optimal, physically realistic models for scalable computers. In the contemporary logic network design, the use of the third dimension is motivated by decreasing of the interconnect topology. The third dimension is thought of as *layering*. For example, in chips, networks are typically assembled from about ten layers. In our study, the third dimension is considered not because of layers but as a dimension that relates to electrochemomechanical phenomena. It includes the space orientations, that is, the 3D relationship between molecules, as the key factor for achieving the desired functional logical properties.

2.8.1 Graph embedding problem

There are several documented approaches in computing in 3D. The first approach is based on the idea that 2D computational structures can be layered. An example is the computer structures that are used in the contemporary logic network fabrication: logic network (for instance, a 2D silicon-based substrate) is layered and interconnected (between layers) to achieve the desired functionality. The interconnected layers form the third dimension. In this approach, the third dimension does not carry functional information about the implemented logic functions. For example, this computational 3D model is not adequate to the natural 3D neural structure.

The second approach is based on the mapping of a logic function into a 3D system (a system with three coordinates) [3]. This approach requires complicated transformations of logic functions with respect to each dimension. The main drawback is that the known techniques of logic network design can not be used in the representation of logic functions in the 3D space. That is, each logic function requires a 3D expansion. This approach is nonacceptable in practice because it requires revision of the previously obtained techniques and tools. The third approach is related to the 3D cellular arrays and their particular case, 3D systolic arrays [29]. These structures are very complicated in design and control, and they are have not been completely implemented in practice. However, linear systolic arrays, which are reasonable for practice, can be used in the 3D computing structure design.

The following characteristics are critical in 3D computing: (a) embedding capabilities, (b) ability to extend the size of structure with minimal changes to the existing configuration, (c) ability to increase reliability and fault tolerance with minimal changes to the existing configuration, and (d) flexibility regarding technology-dependence and technology-independence.

There are two formulations of the embedding problem (Figure 2.10):

(a) Given a guest data structure, find an appropriate topology for its representation; this is a *direct* problem.

(b) Given a topological structure, find a corresponding data structure which is suitably represented by this topology; this is an *inverse* problem (using the properties of the host representation, specify possible data structures that satisfy certain criteria).

The direct and inverse problems of embedding address different problems of nanocomputing. In a direct problems, the topology is not specified, and the designer can map a representation of a logic function into spatial dimensions without strong topological limitations. In this way, a high efficiency in embedding can be achieved. The inverse formulation of the problem assumes that the topology is constrained by technology. That is, the question is how to use a given topology in the efficient representation of logic functions.

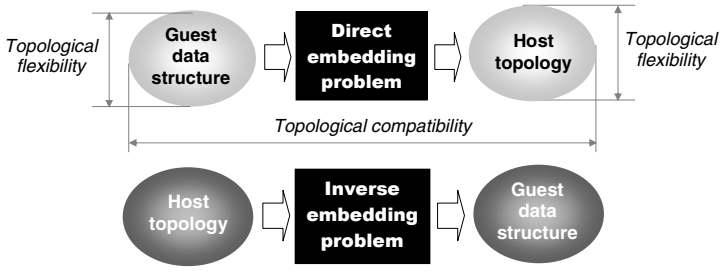


FIGURE 2.10
A formulation of direct and inverse embedding problems.

2.8.2 Embedding decision tree into spatial dimensions

There are a number of particular considerations for representing Boolean functions in nanospace:

(*a*) The Boolean functions have to be represented by a spatial data structure in which information about the function satisfies the requirements of the implementation technology.
(*b*) Information flow has to comply with the implementation topology.
(*c*) Data structure and information flow must be effectively embedded into the 3D topological structure.

In embedding problem, the initial (guest) structure must be specified with respect to computational abilities. A data structure becomes a computing network if the properties of data structures for the representation of Boolean functions are delegated. We utilize the fact that decision tree is topologically isomorphic to the H-tree, which can be embedded into a 3D hypercube. Hence, computational abilities are delegated to the H-tree. The 3D H-tree topology is constructed recursively from 2D elementary H-clusters. An H-tree embedded into a hypercube is called an N-hypercube topology.

The N-hypercube topology is the most suitable data structures for the representation of Boolean functions for the following reasons:

(*a*) They are 3D, and thus, meet the requirements of distributed spatial topology.
(*b*) They correspond to functional (Shannon expansion) and dataflow organization (information relation of variables and function values) requirements, as data structures such as decision trees can be embedded into hypercubes.
(*c*) They meet the requirements of certain nanotechnologies with local quantum effects and charge-state logic.

Example 2.22 *Given a Boolean function* $f = \overline{x}_1 x_2 \vee x_1 \overline{x}_2 \vee x_1 x_2 x_3$. *Its representation by a hypercube given in Figure 2.11a. Another representation is the hypercube-like structure obtained by embedding a decision tree of a function f into 3D H-tree (Figure 2.11(b)).*

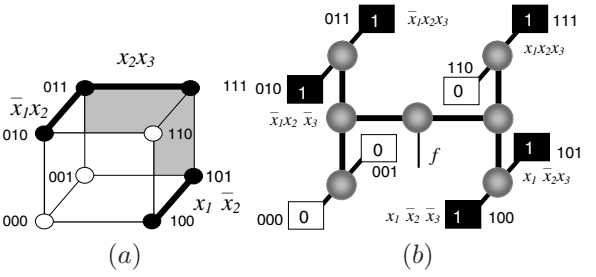


FIGURE 2.11
Representation of Boolean function $f = \overline{x}_1 x_2 \vee x_1 \overline{x}_2 \vee x_1 x_2 x_3$ by the classic hypercube (a) and hypercube-like structure (b) (Example 2.22).

2.9 Further study

Advanced topics of computational nanostructures

Topic 1: The fundamental theoretical sources are adopted from computer science, graph theory, artificial intelligence, and information theory. These techniques are the base of a 3D computing paradigm, in particular:

Graph theory provides techniques for delegation (embedding) of computing properties from 2D to 3D structures for the representation, manipulation, and optimization of logic functions using decision trees and diagrams. Graphical data structures become efficient tools in combination with techniques of logic function theory and information-theoretic approaches.

Logic function theory provides various techniques for manipulation of logic functions, including probabilistic Boolean and multivalued logic functions. These techniques address such problems as reliable computing using unreliable elements, fault-tolerant computing, and optimization techniques.

Information theory provides various techniques for manipulation and

optimization of data structures using criteria of entropy; a topic of particular interest is the relationship of thermodynamic entropy (useful measure in molecular structures) and communication entropy (Shannon entropy).

Artificial intelligence provides the basis of neural network modeling.

Topic 2: The measure of computing structure. The five classic components of a computer structure are *input*, *output*, *memory*, *datapath*, and *control*. These are a technology-independent components. Description of this computer structure is based on a principle of abstraction, that is, each lower layer in the hierarchy of layers hiding details from the level above. The interface between the levels of abstraction is called the *instruction set*. The measure of computing structure is CPU execution time (in seconds):

$$\texttt{CPU execution time} = \underbrace{\frac{\texttt{Instructions}}{\texttt{Program}}}_{\textit{Instruction count}} \times \underbrace{\frac{\texttt{Clock cycles}}{\texttt{Instruction}}}_{\textit{CPI}} \times \underbrace{\frac{\texttt{Seconds}}{\texttt{Clock cycle}}}_{\textit{Clock cycle time}}$$

where CPI is a clock cycles per instruction.

In computer structure, data are represented by bit patterns. These patterns can represent numbers (signed and unsigned integers, floating-point numbers), instructions, and other encoded parameters. Special instructions specify the meaning of the bit pattern.

Topic 3: Collision-based computing. The basic principle of collision-based computing is self-localizations [1, 56]. Truth values of logical variables are represented by the absence or presence of the traveling information quanta.

Further reading

[1] Adamatzky A. Computing with waves in chemical media: Massively parallel reaction-diffusion processors, *IEICE Trans. Electron.*, E87-C(11):1748-1756, 2004.

[2] Agraval V. An information theoretic approach to digital fault testing. *IEEE Trans. Comput.*, 30(8):582–587, 1981.

[3] Al-Rabady A and Perkowski M. Shannon and Davio sets of new lattice structures for logic synthesis in three-dimensional space. In *Proc. 5th Int. Workshop on Applications of the Reed–Muller Expansion in Circuit Design*, Mississippi State University, pp.165–184, 2001.

[4] Akers SB. Binary decision diagrams. *IEEE Trans. Comput.,* 27(6):509–516, 1978.

[5] Alexander MJ, Cohoon JP, Colflesh JL, Karro J, Peters EL, and Robins G. Placment and roting for three-dimensional FPGAs. In *Proc. 4th Canadian Workshop on Field-Programmable Devices*, pp. 11–18, 1996.

[6] Alspector J and Allen RB. A neuromorphic VLSI learning system. In Losleben P, Ed., *Advanced Research in VLSI: Proc. 1987 Stanford Conf.*, The MIT Press, Cambridge, MA, pp. 313–349, 1987.

[7] Ancona MG. Systolic processor design using single-electron digital circuits. *Superlattices and Microstructures*, 20(4):461–472, 1996.

[8] Asahi N, Akazawa M, and Amemiya Y. Single-electron logic device based on the binary decision diagram. *IEEE Trans. Electron Devices*, 44(7):1109–1116, 1997.

[9] Bachtold A, Hadley P, Nakanishi T, and Dekker C. Logic circuits with carbon nanotube transistors. *Science*, 294:1317–1320, 2001.

[10] Beiu V, Quintana JM, and Avedillo MJ. VLSI implementation of threshold logic – A comprehensive survay. *IEEE Neural Networks*, 14(5):1217–1243, 2003.

[11] Bryant RE. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Comput.*, 35(6):677–691, 1986.

[12] Brown BD and Card HC. Stochastic neural computing I: Computational elements. *IEEE Trans. Comput.*, 50(9):891–905, 2001.

[13] Chakradhar ST, Agrawal VD, and Bushnell ML. *Neural Models and Algorithms for Digital Testing*, Kluwer, Dordrecht, 1991.

[14] Crawley D, Nikolić K, Forshaw M, Ackermann J, Videlot C, Nguyen TN. Wang L, and Sarro PM. 3D molecular interconnection technology, *J. Micromechanics and Microengineering*, 13(5):655–662, 2003.

[15] Ellenbogen JC and Love JC. Architectures for molecular electronic computers: Logic structures and an adder designed from molecular electronic diodes. *Proceedings IEEE*, 88(3):386–426, 2000.

[16] Evans WS and Schulman LJ. Signal propagation and noisy circuits. *IEEE Trans. Information Theory*, 45(7):2367–2373, 1999.

[17] Feringa BL, Ed., *Molecular Switches*, Wiley-VCH, Weinheim, Germany, 2001.

[18] Frank MP and Knight TF, Jr. Ultimate theoretical models of nanocomputers. *Nanotechnology*, 9:162–176, 1998.

[19] Frank MP. Approaching the physical limits of computing. In *Proc. 35th IEEE Int. Symp. Multiple-Valued Logic*, pp. 168–185, 2005.

[20] Gaines BR. Stochastic computing systems. In Tou JT, Ed., *Advances in Information Systems Science*, Plenum, New York, vol. 2, chap. 2, pp. 37–172, 1969.

[21] Gershenfeld N. Signal entropy and the thermodynamics of computation. *IBM Systems J.*, 35:577–586, 1996.

[22] Gerousis C, Goodnick SM, and Porod W. Toward nanoelectronic cellular neural networks. *Int. J. Circuits Theory Appl.* 28(6):523-535, 2000.

[23] Han J and Jonker P. A system archtecture solution for unreliable nanoelectronic devices. *IEEE Trans. Nanotechnology*, 1(4):201–208, 2002.

[24] Han J, Jonker P, Qi Y, and Fortes JAB. Toward hardware-redundant, fault-tolerant logic for nanoelectronics. *IEEE Design and Test Comput.*, July-August, pp. 328–339, 2005.

[25] Hopfield JJ. Neural networks and physical systems with emergent collective computational abilities. *Proceedings National Academy of Sciences*, 79:2554–2558, 1982.

[26] Hopfield JJ. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings National Academy of Sciences*, 81(10):3088–3092, 1984.

[27] Hopfield JJ. Pattern recognition computation using action potential timing for stimulus representation. *Nature*, 376:3–36, 1995.

[28] Janer CL, Quero JM, Ortega JG, and Franquelo LG. Fully parallel stochastic computation architecture. *IEEE Trans. Signal Processing*, 44(8):2110–2117, 1996.

[29] Kung SY. *VLSI Array Processors*, Prentice Hall, Englewood Cliffs, NJ, 1988.

[30] McCulloch WS and Pitts WH. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–137, 1943.

[31] Majumdar A and Vrudhula SBK. Analysis of signal probability in logic circuits using stochastic models. *IEEE Trans. VLSI*, 1(3):365–379, 1993.

[32] Mead C. Neuromorphic electronic systems. *Proceedings IEEE*, 78(10):1629–1639, 1990.

[33] Minato S. *Binary Decision Diagrams and Applications for VLSI CAD.* Kluwer, Dordrecht, 1996.

[34] Moore EF and Shannon CE. Reliable circuits using less reliable relays. *J. Franklin Institute*, 262:191–208, Sept, 1956, and 262:281–297, Oct. 1956.

[35] Mullin AA. Stochastic combinational relay switching circuits and reliability. *IRE Trans. Circuit Theory*, 6(1):131–133, 1959.

[36] Negrini R and Sami MG. *Fault Tolerance Trough Reconfiguration in VLSI and WSI Arrays*. The MIT Press, Cambridge, MA, 1989.

[37] Palem KV. Energy aware computing through probabilistic switching: a study of limits. *IEEE Trans. Comput.*, 54(9):1123–1137, 2005.

[38] Parker KP and McCluskey EJ. Probabilistic treatment of general combinational networks. *IEEE Trans. Comput.*, 24(6):668–670, 1981.

[39] Peper F, Lee J, Abo F, Isokawa T, Adachi S, Matsui N, and Mashiko S. Fault-tolerance in nanocomputers: a cellular array approach. *IEEE Trans. Nanotechnology*, 3(1):187–201, 2004.

[40] Pierce WH. *Failure-Tolerant Computer Design*, Academic Press, San Diego, CA, 1965.

[41] Poole CP Jr. and Owens FJ. *Introduction to Nanotechnology*, John Wiley & Sons, New York, 2003.

[42] Reed MA and Tour JM. Computing with Molecules, *Scientific American*, pp. 86–93, June 2000.

[43] Sadek AS, Nikolić K, and Forshaw M. Parallel information and compuation with restriction for noise-tolerant nanoscale logic networks. *Nanotechnology*, 15:192–210, 2004.

[44] Shannon C. Reliable machines from unreliable components. *Notes by W. W. Peterson of Seminar at MIT*, March, 1956.

[45] Shmerko VP and Yanushkevich SN. Three-dimensional feedforward neural networks and their realization by nano-devices. *Artificial Intelligence Review,*

An Int. Science and Eng. J. (UK), Special Issue on *Artificial Intelligence in Logic Design*, 20(3-4):473–494, 2004.

[46] Siu KY, Roychowdhury VP, and Kailath T. Depth-size tradeoffs for neural computation. *IEEE Trans. Comput.*, 40(12):1402–1411, 1991.

[47] Shukla S and Bahar RI, Eds., *Nano, Quantum and Molecular Computing: Implications to High Level Design and Validation* Kluwer, Dordrecht, 2004.

[48] Tour JM, Zandt WLV, Husband CP, Husband SM, Wilson LS, Franzon PD, and Nackashi DP. Nanocell logic gates for molecular computing. *IEEE Trans. Nanotechnology*, 1(2):100–109, 2002.

[49] Tryon JG. Quadded logic. In [53], pp. 205–228.

[50] Von Neumann J. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In Shannon CE and McCarthy J, Eds., *Automata Studies*, Princeton University Press, Princeton, NJ, pp. 43–98, 1956.

[51] Webster J, Ed., *Encyclopedia of Electrical and Electronics Engineering. Threshold Logic*, Vol. 22, John Wiley & Sons, New York, pp. 178–190, 1999.

[52] Wicker SW. *Error Control Systems for Digital Communication and Storage*. Prentice-Hall, New York, 1995.

[53] Wilcox RH and Mann WC, Eds., *Redundancy Techniques for Computing Systems*. Spartan Books, Washington, 1962.

[54] Winograd S and Cowan JD. *Reliable Computation in the Presence of Noise*. The MIT Press, Cambridge, MA, 1963.

[55] Yakovlev VV and Fedorov RF. *Stochastic Computing*. Engineering Industry Publishers, Moscow, 1974 (in Russian).

[56] Yamada K, Asai T, Hirose T, and Amemia Y. On digital LSI circuits exploiting collision-based fusion gates. *Int. J. of Unconventional Computing*, 4:45–59, 2007.

3

Binary Arithmetic

3.1 Introduction

The binary number system is the most important number system in digital design. This is because it is suited to the binary nature of the phenomena used in dominant microelectronic technology. Even in situations where the binary number system is not used as such, binary codes are employed to represent information at the signal level. However, other number systems can be useful in computing nanosystems. For example, 4,8,16, and 32 number systems can be used for memory devices. These systems address a multivalued logic. Multivalued logic values are often encoded using binary representations. However, humans prefer decimal numbers; that is, binary numbers must be converted into decimal numbers.

In this chapter, binary arithmetic is introduced. The binary number system is fundamental to computers and to digital electronics in general. This arithmetic operate with two values, 0s and 1s, and is the base of all computer devices. Binary arithmetic is used in computing nanodevices design because various chemical and physical phenomena can be encoded by 0s and 1s. Binary arithmetic can be also considered as a fundamental basis for hybrid computing devices design, in which the computing components are implemented using various technologies.

We introduce the techniques for the manipulation of binary numbers and show their relationship to other number systems such as decimal, hexadecimal, octal, and others. Binary data can be logically combined and computed by using theorems of Boolean algebra. Various number systems are examined that are used in digital data structures. These number systems, such as octal and hexadecimal, are used to simplify the manipulation of binary numbers.

3.2 Positional numbers

A number system is defined by its basic symbols, called *digits* or *numbers*, and the ways in which the digits can be combined to represent the full range of numbers we need. In a *positional* number system there is a finite set of digits. Each digit represents a nonnegative integer quantity. The number of distinct digits in the number system defines the *base* or *radix* of the number system. The numerical symbols are used to encode information. The encoding process can result in encoded information having desirable properties for the implementation.

3.2.1 The decimal system

The decimal number system is an example of a *positional* number system. The ten digits $0, 1, 2, \ldots, 9$ can be combined in various ways to represent any number. The fundamental method of constructing a number is to form a *sequence* or *string* of digits or *coefficients*

$$\underbrace{\underbrace{d_{n-1}\cdots d_1 d_0}_{Integer\ part} \overset{Decimal\ point}{\overset{\downarrow}{\bullet}} \underbrace{d_{-1}d_{-2}\cdots d_{-m}}_{Fractional\ part}}_{String\ of\ digits\ or\ coefficients}$$

where integer and fractional parts are represented by n and m digits to the left and to the right of the *decimal point*, respectively. The subscript $i = -m, m-1, \ldots, 0, 1, \ldots, n$ gives the position of the digit. Depending on the position of digits in the string, each digit has an associated value of an integer raised to the power of 10 as follows:

$$\begin{aligned} N &= \underbrace{d_{n-1}d_{n-2}\cdots d_1 d_0 \overset{Decimal\ point}{\overset{\downarrow}{\bullet}} d_{-1}d_{-2}\cdots d_{-m}}_{String\ of\ coefficients} \\ &= \underbrace{d_{n-1}\times 10^{n-1} + d_{n-2}\times 10^{n-2} + \cdots + d_1\times 10^1 + d_0\times 10^0}_{Computing\ the\ integer\ part} \\ &= \underbrace{d_{-1}\times 10^{-1} + d_{-2}\times 10^{-2} + \cdots + d_{-m}\times 10^{-m}}_{Computing\ the\ fractional\ part} = \sum_{i=-m}^{n-1} d_i 10^i \end{aligned}$$

This method of representing numbers is called the *decimal system*. In the positional representation of digits: (a) each digit has a fixed value, or *weight*,

determined by its position; (b) all the weights used in the decimal number system are powers of 10; (c) each decimal digit d_i ranges between 0 and 9; (d) the weighting of the digits is defined relative to the *decimal point*; this symbol means that digits to the left are weighted by positive powers of 10, giving integer values, while digits to the right are weighted by negative powers of 10, giving fractional values; and (e) fractions are denoted by sequences of digits whose weights are negative powers of 10.

Example 3.1 *The four digits in the number 2008 represent, from left to right, thousands (digit 2), hundreds (number 0), tens (number 0), and ones (number 8). Hence, this four-digit number can be represented in the following form:*

$$2008 = \sum_{i=0}^{3} 10^i \times d_i$$
$$= 2 \times 10^3 + 0 \times 10^2 + 0 \times 10^1 + 8 \times 10^0$$

The decimal number 747 in positional form is $747 = 7 \times 10^2 + 4 \times 10^1 + 7 \times 10^0$.

Example 3.2 *The decimal number 12.3456 consists of an integer part (12) and a fractional part (3456) separated by the decimal point. Thus, this number can be represented in the following form:*

$$12.3456 = \sum_{i=-4}^{1} 10^i \times d_i = \underbrace{1 \times 10^1 + 2 \times 10^0}_{Integer\ part}$$
$$+ \underbrace{3 \times 10^{-1} + 4 \times 10^{-2} + 5 \times 10^{-3} + 5 \times 10^{-4}}_{Fractional\ part}$$

The number 0.34_{10} *is represented as*

$$0.34_{10} = \sum_{i=-2}^{-1} 10^i \times d_i = 3 \times 10^{-1} + 4 \times 10^{-2} = {}^{34}/_{100}$$

3.2.2 Number radix

In general, an n-digit number in radix r consists of n digits, each taking one of r values: $\underbrace{0,\ 1,\ 2,\ \ldots,\ r-1}_{Radix\ r\ system}$. A general number N in a positional number system is represented by the following formula: