0.0in

NEW CANONICAL FORMS FOR FOUR-VALUED LOGIC

Marek A. Perkowski, and Edmund Pierzchala

Department of Electrical Engineering, Portland State University, Portland, OR 97207
P.O. Box 751, Portland, Oregon, 97207–0751
tel: (503) 725–3823

e–mail: edmundp@ee.pdx.edu mperkows@ee.pdx.edu
17 November 1993.

Contents

List of Figures

Abstract

In this report we extend our general approach to design of novel programmable devices that we outlined first in [42, 40] and that is further extended in [41]. There, we proposed the general architecture of an FPGA for the realization of continuous, fuzzy, and multi-valued logic (MVL) circuits. We demonstrated how a general-purpose field programmable analog array (FPAA), with cells realizing simple arithmetic operations on signals, can be used for many other purposes in digital and analog worlds.

An important point to make is that our goal is to design the best architecture for **very high speed** (high-frequency) field programmable analog array (FPAA), rather than inexpensive "glue-analog" circuit. Thus, it is aimed not at the near future "glue-analog" market (filters, amplifiers) but it is intended for the emerging in future **"analog supercomputers"** market. In such circuits, designed in sub-micron technologies and possibly quantum technologies, the regularity of structures and short connections will become the most important. We assume that such computers will be mixed, analog-digital designs in which logic will be binary or multi-valued, most probably, quaternary.

We introduce here a new concept in VLSI layout which can find applications in submicron design, quantum and optical devices, and designing new fine-grain analog, digital and mixed FPGAs. This concept is a **regular arrangement of cells**, every cell is connected to 4, 6 or 8 neighbors and to vertical, horizontal and diagonal buses. Methods for expanding arbitrary binary, multi-valued and continuous combinational functions to this layout will be illustrated. Some of our expansion ideas are based on orthogonal logic [38, 39]. Two-dimensional arrays [47] similar to those from Concurrent Logic Corporation [5] are assumed as a background, as well as two-level PLAs and multi-level Generalized PLAs [63]. In some more general sense our structures are canonical, it means, when some more agreements about orders of variables or expansion types are first done.

Because this is only an initial report, a base of many future papers, we will be not very formal and will try rather to appeal to intuition to demonstrate the deep generality of all our concepts. We are sure that they can be all better formalized, and better algorithms will be with time found for them. However, now we want only to show the very general relations among various electronic design subeareas, that amazingly, have been not discussed in the literature. Although we concentrate on 4-valued logic, most of the results are general.

Chapter 1

The Fundaments of Regularity

1.1 Introduction

1.1.1 Floorplanning and tesselations

In our previous work and in a patent we proposed an architecture for current-mode FPAA. It is suitable for a wide class of circuits, including continuous-time filters, differential and algebraic equation solvers, dynamic systems, and fuzzy and multi-valued logic circuits. Of course, binary is treated below as multi-valued, as its special case. The demonstrated analog examples included an implementation of a ladder continuous-time filter, an analog rank filter, a circuit for tracking a product of two matrices, a linear equations solver, and a circuit for solving a linear programming problem by the method of steepest descent.

Although our FPAA was optimized for analog applications, we clearly see that it can be used for multi-valued, or even binary FPGAs. Respective circuits for the specific application niches and narrower markets can be easily derived in more detail from the general model [42, 40]. In binary case, our ideas can be treated as a generalization and improvement of the fine-grain Concurrent Logic FPGA architecture [5]. Thus **a number of continuous, fuzzy, multi-valued, binary** and **mixed** architectures spread between the FPAA and the Concurrent Logic chip can be created, for which we will present here the general outline and theory only. Reconstruction of each of them should be however possible for a careful reader based on general principles given here.

The power of our model comes not from one or another way of representing signals (binary, multi-valued, fuzzy, continuous, probabilistic), but from **general paradigms of realizing calculations in space in an efficient way.** Thus, the main problem is: **"how to use space efficiently, what are the all possible local connection patterns of cells".** Such questions are **"layout-floorplanning"**, "cellularity", or **"tesselation"** questions rather than particularly analog or digital technological questions. However, we need to refer to analog and multi-valued circuits to prove generality of our answers to this question, and to demonstrate illustrative examples.

1.1.2 Regular structures and algebraic fields

Whether we are considering analog or digital circuits, from the point of view of computing, the most important are the following components of the computing process:

- **operations** that are executed,
- **information** that is stored and processed, and
- the **general computational structures** in space and in time that make the calculations efficient.

In the truly massively parallel and system-oriented ULSI of the future, the structure will be inherent to the calculation pattern, and not to the realization technology. For instance, the butterfly-like structure of a bubble sorter remains the same whether it is used in analog or digital, in combinational or sequential design. Similarly, Fast Fourier or Walsh transform can use digital or analog multipliers and adders, but the structure of blocks' interconnections will remain the same. Small RAMs, registers, or sample-hold circuits can be used for memory, but again, from the point of view of futuristic large systems realized in ULSI, important is only to use some kind of memory element. The most important is how structures from such elements are build so that their connections will be short and/or regular (such as cells' abutting or parallel buses). Thus, if analog calculation is **unidirectional**, there is a one-to-one method of mapping analog concepts to digital concepts, and vice versa, based only on components' functionalities and connection structures. As examples, all our analog circuits from the above mentioned circuits can find their digital counterparts: analog adders are replaced with digital adders, and analog multipliers are replaced with digital multipliers. We use the property that in a **linear system** all calculations are performed in an algebraic structure called a (discrete) **field**. There exist mathematical concepts called **field of real numbers** and **field of complex numbers** for continuous (analog) calculations. Similarly, there exist **Galois Fields** for digital (binary and multi-valued) calculations. Similarly, any digital pipelined, iterative or systolic circuit can be converted to analog circuit. For instance, an analog systolic sorter can be build by analogy to a digital systolic sorter. New structures of digital filters can be designed by mapping from analog filters. The same is true for switched capacitance circuits. (Edmund can find more examples for his Ph.D.). Amazingly, there is not much published on these topics, because the analog and the digital worlds are far away one from another, and there persists also a common belief that there is not much future in analog calculations and computers. We believe, however, that in the far future (say, year 2010) the systems will be so big, that mostly their "mathematical structures" important. Definitely they will be more important than whether they are multi-valued, mixed, analog or binary. The massively parallel systems will use mixed technologies and EDA tools for them will be easily available.

1.1.3 Analog and Mixed ULSI Supercomputers

Because we believe that there is a bright future for **analog self-learning supercomputers**, that will represent instincts and behavioral intelligence for a fraction of cost and with a minimum power consumption, we investigate structures of computations that can be used for fast image processing, voice recognition and other circuits that will find wide market applications. Such "ULSI supercomputer brains" will be used in intelligent toys and household appliances, so their speed, cost and low power consumption will be more important than the accuracy of calculations. Fuzzy logic camera controllers are first examples of such circuits in the market. We believe that a chip with integrated voice recognition and voice synthesis, internal digital camera, image analysis and pattern recognition will be developed before year 2005. Also, micromachining and robotic devices will become common. (Edmund should list here all examples of applications that he discussed in his slides and with Analogix customers and military).

Our device achieves high performance through the reduced use of global signal interconnections, in favor of local ones. Although our FPAA is being implemented in a bipolar transistor array technology, there is no fundamental reason not to implement it in CMOS or any other existing or future technology.

Although analog circuits are a necessary component of many modern signal and information processing systems and they arrive in a number of applications, such as anti-aliasing and smoothing (reconstructing) filters, or pulse-slimming circuits in computer disk memories [59], many researchers and technology gurus have a perception that they are only a niche for interfacing to real world but the digital circuits should be used in computations.

When we, however, take into account modern "artificial life" theories that intelligence is emergent as a result of low-level sensory interactions with an environment rather than based on formal symbol manipulations, the role of sensors and effectors to build modern learning-based robots will be increasing and, as in biology, the future sensors will have much more intelligent processing built into them. It seems impossible to build such circuits entirely in a digital form because of the enormous power and complexity of these calculations, even for a single application in such as computer vision, prosthetic control or "artificial skin". Thus, we believe that for these calculations, especially for intelligent robotics, **high complexity analog/digital chips** will be built that will have also sensors and effectors (mechatronics) integrated inside them. Such devices should include fault-tolerance, masive parallelism, self-repair based on redundancy and reconfigurability, and graceful degradation. These properties, we believe, can be achieved only in **regular structures**, whether digital, binary, or mixed.

Concluding, many analog circuits **cannot** be replaced by digital circuits, either for the reason of speed, or because of their unique ability to work in continuous time mode, or – as in some signal processing applications – they are preferred over digital circuits for their relative simplicity and low power consumption. In this report, we intend to look bravely into the future systems and try to predict what will be the fundamental issues for designing them; our work is thus not for an immediate use.

The plan of this report is the following. In section 2 we will present the main ideas of our Field Programmable Analog Arrays. Section 3 will present fundamental issues related to designing regular arrays, and section 4 will demonstrate few applications of such arrays. These applications will demonstrate the practicality of our regular local/global connection patterns with horizontal, vertical and oblique connections. Section 5 will illustrate the fundament of realizing "logic" of the regular circuits - the regular expansions, both binary and MV. In section 6 we will present more details on how our cell of FPAA realizes the elementary functions necessary to implement various MV logic operations needed in regular expansions and structures. Section 7, the crux of the paper, presents various types of quaternary expansions, some canonical and relatively easily realizable in hardware. Such circuits are of special interest because of easy conversion from/to binary. (Also, they can be realized in regular layout; logic radices 5, 6, and 7 are of not big practical interest, and the radix 8 seems to be too high for many practical regular layouts). Section 8 discusses realization of expansions. Section 9 presents expansions for ternary logic, which are of interest as the simplest generalizations of binary expansions. Section 10 presents fuzzy logic diagrams. Section 11 discusses possible regular structures and extensions to this research. Section 12 presents state machines realized in regular layouts. Section 13 concludes the report.

1.2 The essence of Field Programmable Analog Arrays

1.2.1 Success of digital CPLDs and FPGAs is based on regularity and reprogrammability

The unquestioned success of digital programmable devices of various kinds [61, 4, 8, 9, 10, 11] is perhaps one of the main stimuli for the development of analog circuits with similar degree of programmability. A number of analog programmable circuits for various applications were reported. However, full programmability, i.e. the possibility to program both the parameters **and** the structure, is still an important research issue in analog circuits domain. Whereas several interesting approaches were published, leading to various degrees of programmability, many problems still remain unsolved [8, 17, 19, 25, 26, 30, 31, 32, 33, 34, 46, 48, 50, 51, 62].

Programmable designs for analog circuits presented in the literature to this day seem to favor flexibility (universality) of the topology (pattern of connections in the programmable device) rather than high performance [27, 28, 29, 52]. This is perhaps due to the wide spectrum of topologies of analog circuits designed currently, which do not seem to form any "typical" pattern of interconnection schemes, or at least a pattern suitable for analog circuits. There is nothing like an analog FPLD yet. As a result, architectures with many long (global) signal interconnections are created. Whereas they achieve greater flexibility of interconnection patterns, sometimes allowing almost every cell in the programmable device to be connected with any other cell, this approach seriously jeopardizes high frequency performance of the programmable device. Effects caused by parasitics associated with long signal lines and crosstalk between such lines, crosstalk between long analog lines and digital lines on the same chip, etc. are the source of serious noise and stability problems.

1.2.2 Regular structures in Nature

We believe that the solution should be found by analyzing biological systems, because Nature created the most perfect systems known to humans, in the long process of evolution. When we observe biological (and also human-made structures), we can distinguish certain patterns, such as: **trees, cristals, leafs, nets, spider-nets, honeycombs, walls from bricks, tiles in a bathroom, lattices, fractals,** etc. Because they occur in so many different areas of alive and non-alive matter, there must be something very fundamental in them, perhaps Nature and designers found how to organize informational structures in space in such a way that the communication or the information flow are optimized. Some of these issues are investigated in the area of mathematics called **tesselation structures and design**, but we have not found there many results useful for our research yet.

For the last few hundred years humans are reproducing these structures in the developed by them mechanisms, organizations and ways of thinking (hierarchy, cellularity, bus structures, railway networks, etc). Recently, these "cellular" ideas are being reproduced in neural nets, cellular automata, systolic processors, genetic algorithm structures, and many other.

What are these structures? What is general in them? How to create and utilize them?

Communication in biological systems is slow, their power comes from parallelism. We want to reproduce these parallel structures, but realize them with the fastest possible frequencies allowed by any of the modern technologies (electrical circuits, optical, chemical). (In this report, the words: architecture, topology, and structure will be used interchangeably where it does not lead to confusion.)

Most challenging analog circuits are built for the highest frequencies; it is where they have the most interesting and desired applications. Although some of the abovementioned design problems can be partially counteracted by appropriate circuit and layout design techniques, it seems that for a successful development of analog programmable circuits the **basic architectural questions** need to be addressed and that these issues are very similar to those in future submicron digital technologies.

In this report we focus on the design of **topologies** suitable for **high-speed, high-performance fully programmable analog circuits.** The highest performance can be achieved by reducing the length of signal interconnection lines, if possible – by using locally-only interconnected structures.

Again, this property exist in nature and we can learn much by studying natural cells and tissues.

Such an approach – of course – limits the class of circuits realizable in a given structure. However, interesting circuits, such as ladder continuous-time filters, equation solvers and arbitrary combinational logic, can be realized in **locally-only interconnected topologies.**

Many animal tissues involved in motion control and early vision, especially of lower animals, display amazing regularity of cells, with mostly local connections, but also with some **global connections**. Certain **irregularity** of interfacing regular planes can be also observed in nature (leafs, crystals, bone tissues), and we think that there are some deep information-theory reasons for this. Again, we will reproduce these ideas in our circuits, not to blindly model the Nature, but because we found experimentally that such structures are more efficient than the "pure" regular structures such as a PLA.

In the patent, and partially in our previous papers, we presented architectures of a fully field programmable analog arrays. In a series of examples we demonstrated how by extending an initially locally-only interconnected topology one can accommodate more and more complex analog designs, never compromising the high performance for the sake of functionality. Here we will add more such examples from various areas, also digital.

1.2.3 Local or global interconnections ?

Many digital and analog circuits are characterized by predominantly local signal interconnections. In digital area we can point to adders, multipliers, memories, registers, systolic processors, as examples of such circuits. In analog area, for instance, in multistage amplifiers, there is usually a local coupling between the stages, and local feedback loops around individual stages. Rarely does a feedback loop go around more than two – three stages, and almost never around the whole amplifier. This is dictated by stability considerations. Also, rarely is a feedback loop around many stages necessary [16, 18]. Another example of topologies containing only local interconnections are some of the multiple-feedback-loop topologies for continuous-time filters, particularly one of the most useful, namely the ladder [40, 43, 49].

Other classes of circuits, such as cellular neural networks (CNNs) [3, 4, 45, 46, 58], cellular automata [57, 56], and certain considerations in artificial neural networks (ANNs) suggesting the advantages of locally interconnected architectures over fully interconnected ones, provide strong incentive to address the question: how useful are the local interconnection structures for the design of programmable analog devices ? Most interconnections in biological systems, artificial and social systems are local in time and space. Is this analogy useful?

Observe, that so far, the relations of digital and analog structures were observed mostly by "system researchers", working in filter theory, learning networks, neural networks and cellular logic, perhaps because they concentrate on large structures, starting from high-level functional models and using certain initial high-level functional description languages, such as equations or data flows, which hide the implementation details. Perhaps, this trend will continue, because the future intelligent computational systems will be to a higher extent influenced by biologists and system-theorists than circuit design engineers.

1.3 Fundamental Concepts related to Regular Arrays

We propose to build programmable systems (analog, digital, probabilistic, fuzzy, continous or discrete domain, on level of bits or words), **based on local signal interconnections.** To make the array more general, a carefully designed global interconnection structure is superimposed on the local interconnections.

The array provides flexibility for implementing circuits with local connectivity, in a way maximizing the circuit's performance. Global interconnections can be used when absolutely necessary, but should be well-structured (buses), or better, totally avoided.

The device of the FPAA cell.

Any cell-based programmable device design must address at least two issues:

(i) what is the functionality of an individual cell, and

(ii) how are the cells interconnected to allow implementation of the desired class of systems (processors, computer networks, circuits).

The first question can be addressed more or less systematically, by analysing the functions performed by the circuits under consideration.

In analog designs, a cell will contain usually a gain element of some kind, and perhaps a time-domain (dynamic) operation such as an integration. Some nonlinear functions may be added to enhance functionality of combinational transformations. The design of our family of analog cells and of the respective interconnection structures is oriented on dynamic-system type of applications, fuzzy, and multi-valued logic. The same can be done in digital domain, the only difference is that in binary we have boolean operations and flip-flops (on the bit level) or arithmetic operations and registers (on the word level). However, in all cases, if the system is linear, the same mathematics can be used, linear algebra of **fields** and **groups**. Our examples demonstrate that what is realized in a real field in continuous circuit, can be easily mapped to Galois Field (2) for binary logic and to arbitrary Galois Field(k^n) for k^n radix multi-valued logic. Standard fuzzy logic is a trouble, because it does not satisfy axioms of a Field or even of the Boolean Ring, but some systems similar to fuzzy logic and based on fields can be built and will be briefly discussed in this report. Thus, our interest here will be on all kinds of fields, but systems for some other algebras can be designed in a similar way (we show very few examples of them as well). (Edmund can next add more in his Phd).

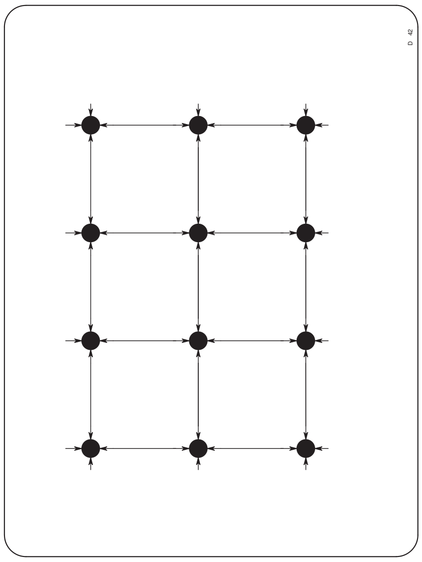


Figure 1.1: *Local 4x4 connection pattern for the proposed FPAA.*

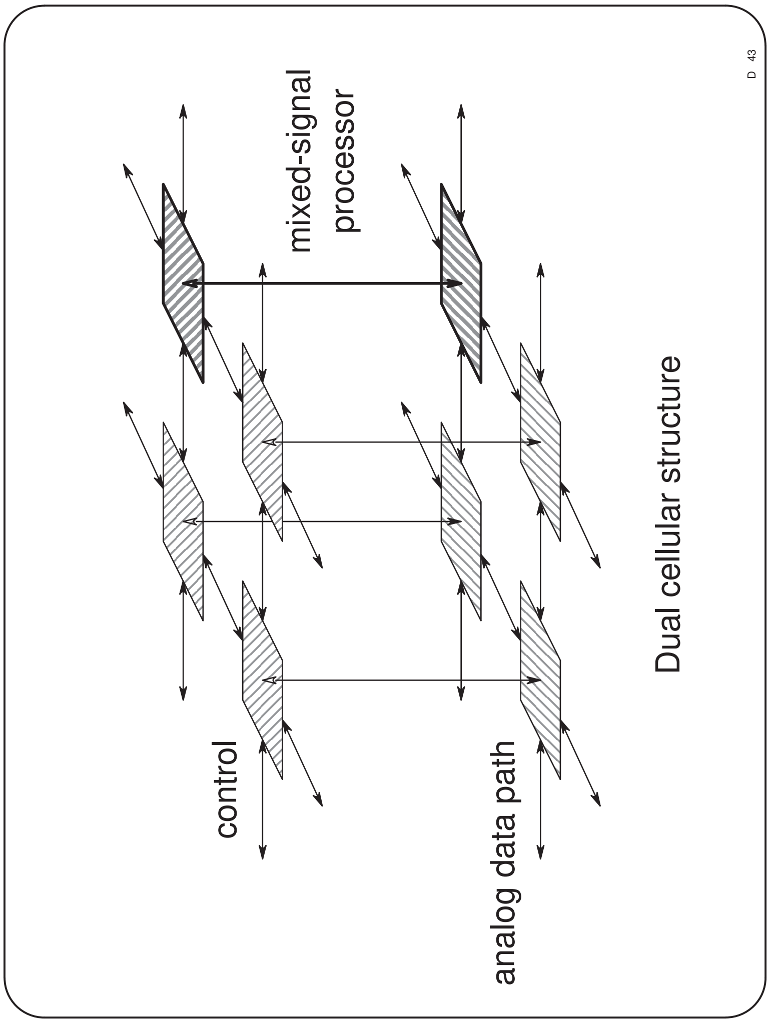


Figure 1.2: *Dual control structure of our FPAA.*

Our Generalized FPGA is based on a regular, square array of processing cells, interconnected on two levels: local, and global one. Figure 1.1 shows the local interconnection patterns. It is the so-called **4x4 connection pattern,** since every cell has 4 inputs and 4 outputs. Figure 1.2 presents the dual control structure of our FPAA based on this pattern. Observe that this pattern is very general and includes patterns of CNNs [3, 4, 19, 46], PLAs, Generalized PLAs [39, 47, 38], multi-level PLAs [63], and other arrays for Boolean logic.

Another advantage of our design is the separation of control and data path (Fig. 1.2). A cell from the control sphere and its counterpart cell from the data path sphere create a kind of mixed-signal reconfigurable processor. The Data Path, in which speed is important, is separated from the control part that reconfigures it. The reconfiguration process occurs only from time to time, so it should not be as fast as the data path operations.

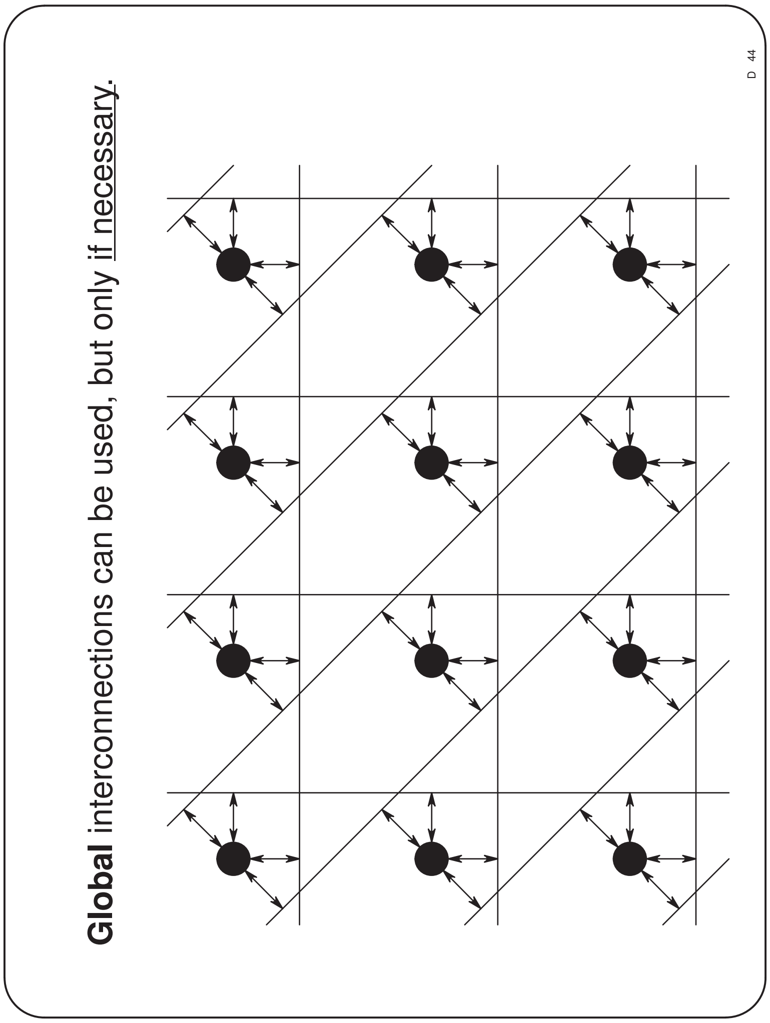


Figure 1.3: *Global connections for the proposed 4x4 FPAA.*

Figure 1.3 presents the global interconnection pattern. Observe, that although many parallel computers have been proposed or build that have vertical and horizontal buses, and although ATMEL or MOTOROLA Fine Grain FPGA architectures use horizontal and vertical buses connected to each cell, there are no industrial or research architectures and arrays with oblique buses. We show here examples with **one oblique** or with **two oblique** buses per cell, one vertical and one horizontal, but these proportions can be, of course, different. We claim, however, that at least one connection (local or global) of each type, horizontal, vertical and oblique is necessary in the **most powerful regular arrays** shown below. It is important that we strongly advocate the use of **oblique, 45% connections, and next also 135% connections**. Although we were told by the industry experts that such connections are not practical for current FPGA technologies, we see no reason why they should be avoided in future technologies. The necessity of using such connections results from the general principle of making the best use of local space available to any cell, thus, making its best possible communication patterns with **local neighbors** and **distant neighbors** (through buses). (By a distant neighbor we will call any sub-circuit on the same bus). **Our goal will be to design circuits where all cells are either neighbors or distant neighbors.**

In the analog variant, every cell is connected to its four nearest neighbors by a two-way current-mode signal interconnection. Thus it is able to receive four different signals produced by those neighbors, whether all of them, or just the selected ones. The cell's own output signal is programmably distributed to the same four neighbors (Fig. 1.1). The global interconnection pattern is superimposed on the local one, but it is shown separately to avoid clutter (Fig. 1.3). Each cell can broadcast its output signal (there is just one output signal produced by the cell, the same one which is sent to its local neighbors) to any of the four global lines to which the cell is connected (possibly to more than one line at a time). Since the signals are in current mode, they are summed on the global lines. Observe that this is a generalization of the WIRED-OR and WIRED-AND principles known from binary logic circuits. If more than one cell receives the signal from a given global line, the signal will be divided evenly by the receiving cells. Each cell can therefore send and receive signals to and from any of it's four nearest neighbors and any of the four global lines to which it is connected. Each cell thus has eight input ports and one output port (or, in fact, eight output ports with copies of the same output signal). Observe, that these principles were created by mathematicians in the 40's [56], used in theoretical designs in the 60's [1], used in binary chips only recently [5], and never proposed for analog or MV arrays before [41].

Thus, in analog and some digital technologies the global summing operations can be done in buses (WIRED-OR, WIRED-AND). In other technologies the summing operation is composed from gates that are distributed near cells. But the general need for summing (collection, OR-ing, ARITHMETIC-summing), remains irrelevant of technology. Also, in unidirectional technologies the organization of adding is different, but the principles remain the same. These are the most general principles of structures and related operations in this report that are our base of thinking about future structures.

1.3.1 Distribution and collection

Based on previous sections, we we can make the following observations:

- There exists only a limited number of tesselation types; i.e. the ways of connecting cells in a two-dimensional space such that the pattern remains the same and the cells are connected only to neighbors (local and distant).
- There are only two main operations related to data flow in structures themselves rather than to the processing of operations: one of these operations is the **distribution** of data flow. The other operation is the **collection** of data flow. In software programming, the distribution operators are the **fork operator** and the **if-then-else operator**, and collection operators are the **JOIN** operator or **DAND** and **DEXOR** operators known from "parallel flow-graphs" of Karp and other authors [64]. In hardware the distribution operator is the multiplexer, demultiplexer and selector (the reader can appreciate their similiarities with if-then-else), and the collection operator is the gate such as OR, AND, EXOR, adder, or multiplier. It can be a wired-OR circuit.

Observe, that there is a duality between the distribution and collection operators, similar to SOP and POS forms, or multiplexer and demultiplexer. In terms of logic functions realized in regular structures, the distribution operator "cuts" to smaller subfunctions, such as a cofactor or an AND gate cut to smaller functions. The collector collects some sub-functions created by distributors. Thus, in AND/OR SOP, there is an AND plane of distributors that cut to smaller and smaller cubes, (the prime implictants), and the OR plane that collects them to Sum-of-Product form. The reader has to think about AND and OR planes as composed of two-input AND and OR gates, to better see this analogy; each AND gate being a distributor and each OR a collector. When we realize a Binary Decision Tree or a BDD, each node is a combination of a distributor that cuts the function to smaller and smaller cofactors, and a collector of cofactors. Such node can be this called distribution/collector and is thus created in a process that requires both operations. Concluding, in SOP the functions of distribution and collection are separated to planes. In BDDs they are combined in signle gates, which has many advantages but leads to loss of regularity. There are no collectors, only the terminal constants 0 and 1.

1.3.2 Symmetric functions have regular layout

In Binary Decision Diagrams, each node is both a distributor and a collector. If a function is symmetric, the structure of BDD becomes completely regular, and **each node plays regularly the role of a distributor and a collector.** Such structure is thus perfect in a sense and beautiful because of its regularity and simplicity. Thus, it is an ideal candidate for layout. All cells are simple, all are the same; all connections are short or in buses. Buses are parallel so they can be designed to avoid **cross-talk** and **noise**.

Unfortunately, only a small class of Boolean functions is symmetrical. However, there are many remedies to this problem:

- non-symmetric functions can be **symmetricized** by **repeating** variables.
- even many non-symmetric functions can have regular layout of their BDDs.
- extending from binary to multi-valued logic (which is equivalent to grouping k binary variables to a 2^k-valued bus) extends the concept of "symmetric layout".
- using Davio expansions (in addition to Shannon expansion used in BDDs), also extends the concept and various uses of symmetries).
- using orthogonal generalizations of Davio expansions [39, 38] for binary and for multi-valued logics are another possible approaches.
- bi-directional decision diagrams can be introduced that correspond to pass-transistor circuits (this can be also expanded to multi-valued logic circuits and devices).
- some small and local sacrification of regularity in the generally "cellular" structure to reduce the number of cells is still possible for any of the above listed approaches.

1.3.3 General approaches to build regular layouts

Now, the following questions occur.

Can we create (semi)regular structures only for analog or binary logic, or there are the counterparts of these structures in multi-valued and continuous logic?
If yes, how to build structures.?
How to find general distributor and collector circuits?
To answer these questions we will try to find analogies between binary logic and other logics. Such analogies can be found based on the following methods:

- **formal analogies of algebras and expansion formulas**, for instance, in [42] we found a continuous-logic generalization of Shannon expansion to be used in regular structures. Observe that the standard binary Shannon expansion separates to two cofactors, one for value of the "control variable" equal zero and the other for value of this variable equal 1. When we generalize it to multi-valued logic, we just separate to as many cofactors as there are the values of the control variable, but if we want to have only two partitions, we have next to combine some of them again together, so that the distribution of data flow will be only to two branches (another approach would be to have distributors and collectors with more than two branches, see below). Thus, one branch can be for values 0,1,3 and the other branch for values 2,4,6, etc. As we see, there are many such expansions. In continuous logic the situation is different, because there is no discrete and fixed values. We have thus to separate to **intervals** of control variable values. The simplest method is to use a single comparator (which is a very simple circuit in analog design, and is a base of many circuits, especially in MV logic). Comparator allows to cut to regions $A > const$ and $A \leq const$. Next, we can use several comparators in a node, each comparing to one constant, make distribution to intervals $[a_{i1}, a_{i2}]$ and sum these intervals. Thus, the expansion becomes a very powerful function and any function of interval logic can be realized in this kind of "symmetric layout" pattern (see next section for more details).

- **The use of EXOR logic.** We have to answer the question - what would be an analog or MV logic most general counterpart of an EXOR gate?

- If more generalized distributors were invented, then more general collectors must also exist. Although in the above example the node is both a distributor and a collector, in general, more complex relations between distributors and collectors are possible. In any case, first we have to concentrate on circuits that are both collectors and distributors, such as binary, multi-valued and continuous multiplexers and demultiplexers, and their generalizations along the lines outlined above.

- For PLA-like structures, which are the simplest regular structures, the distributing (cutting) operators, are easy. They are always generalizations of AND gate, such as MINIMUM, MODULO-MULTIPLICATION, or GALOIS-MULTIPLICATION. The collection operators are MAXIMUM, MODULO-SUM, or GALOIS sum. However, more general PLAs are also possible, in which the AND plane is generalized to a plane from arbitrary gates [47]. Here we will generalize these results to MV and continuous logic.

Thus, we outlined the principles based on which the local-connection symmetric and regular pattern from Figure 1.1 can be used for: continuous filter design, binary symmetric functions, arbitrary (not-symmetric) binary functions, arbitrary MV functions, and arbitrary piece-wise continuous functions.

Another method to create analogies between binary, MV, arithmetic and analog circuits will be to look to well-known structures used already in practice and try to generalize them based on common patterns that exist in them.

1.3.4 The problem of generality of our FPAA cell and structure

As it can be found in [43], the analog cell of our FPAA performs summing of selected input signals, multiplication of two signals (squaring of one signal), or multiplication of two independently derived sums of input signals. Further processing includes lossless or lossy integration, and clipping. Equivalents of all these operations, and more, can be realized in binary or in multi-valued logic.

We believe that these basic operations are sufficient to build a wide class of distributors and collectors from them. The question of course arises, **"are these operations, selected arbitrarily by us, sufficient to build every imaginable circuit?"** To answer this concern, it would be sufficient to show that:

1. P1. We can build many circuits that are **practical** and important.

2. P2. We can build **every** logic function, whether it is binary, MV or continuous.

3. Because every circuit is "logic" plus "dynamic behavior" if P2 is demonstrated **and** we will be able to build any dynamic behavior, then we could build every circuit known. What is a "dynamic behavior"? As an example, observe that an analog filter has a regular structure of cells, each of which having logic and memory functions. "Logic-type" are the operations such as arithmetic additions and multiplications by constants in the Laplace transform. "Memory functions" are

related to 1/S transform blocks. If we would modify now the structure of this filter to a DSP circuit, the "logic" will become adders and constant multipliers, 1/S blocks will become registers corresponding to D or Z operators. Similarly, any state machine can be build from logic and memory elements: binary machine from binary logic and binary flip-flops, MV machine from MV logic and MV flip-flops, fuzzy machine from fuzzy logic and fuzzy flip-flops.

Simple Laplace transforms become counterparts of D and T flip-flops. Thus many flip-flops can be created for MV logic, but one is enough for completeness. Obviously, out of all dynamic behaviors, only very few are entirely sufficient to build arbitrary systems. Linear and non-linear system theories can be used to build control systems with given elementary dynamic behaviors and arithmetic operations (prof. E.B. Lee, private communication). **Again, only the structure of connections is really important.** We can show that every machine can be realized in a regular structure of memory elements and regular-layout "logic", whether this logic is binary, mv, digital integers, or continuous.

The above mentioned functions are important for the implementation of continuous-time dynamic-systems [21], multi-valued and continuous (Lukasiewicz, fuzzy) logic circuits, which are the expected applications of our FPAA. Even if the given by us operations are not sufficient, we believe, based on our few years of patent-writing experience, that ultimately we can add few more basic blocks to the node to cover all necessary behaviors for any practical class of problems.

So again, the main issue are the fundamental structures and not what is in their nodes. If we invent a sufficiently powerful structure, we can keep making it more powerful by adding functionality (flexibility, operability) to its nodes. Observe that already in digital design, the same **structures**, such as one in Figure 1.1 are used in fine grain FPGAs (on the level of single logic gates), and in parallel supercomputing (on the level of entire processors).

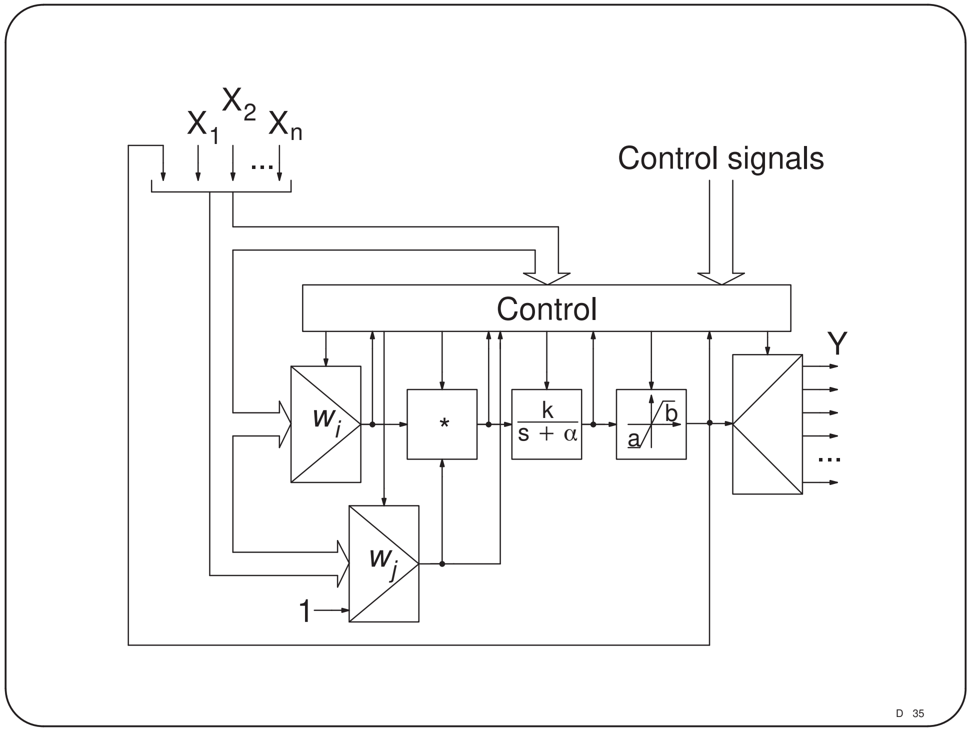


Figure 1.4: *Structure of our analog programmable cell.*

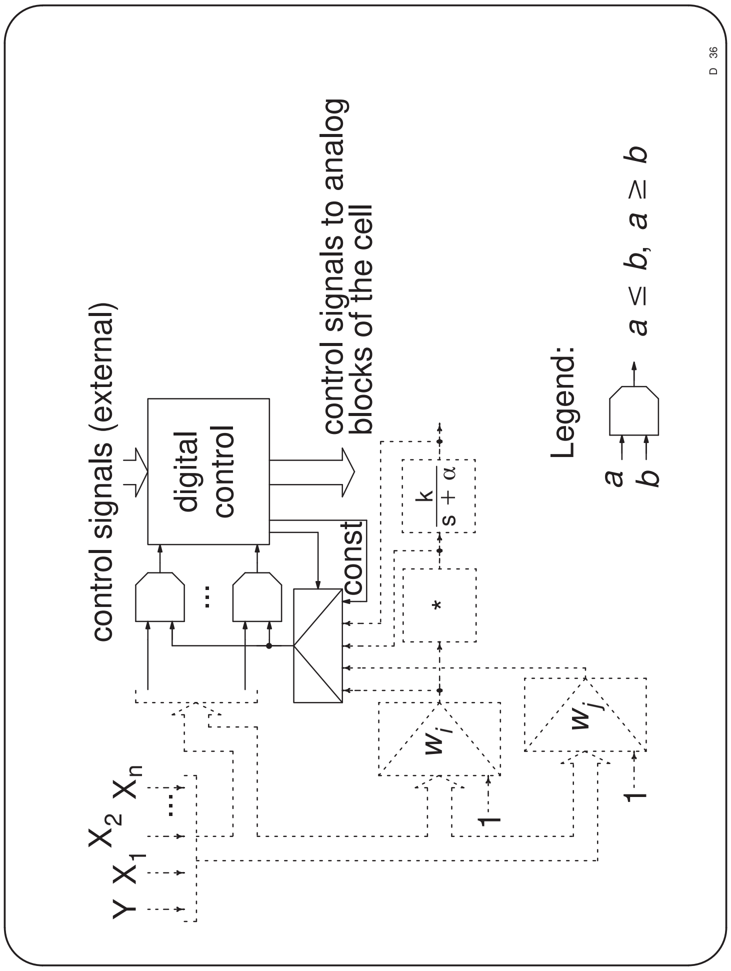


Figure 1.5: *Control of the programmable analog cell.*

The structure of our cell is shown in Figure 1.4 and its control in Figure 1.5.

1.3.5 Examples of regular arrays with local connections only

Continuous-time ladder filter

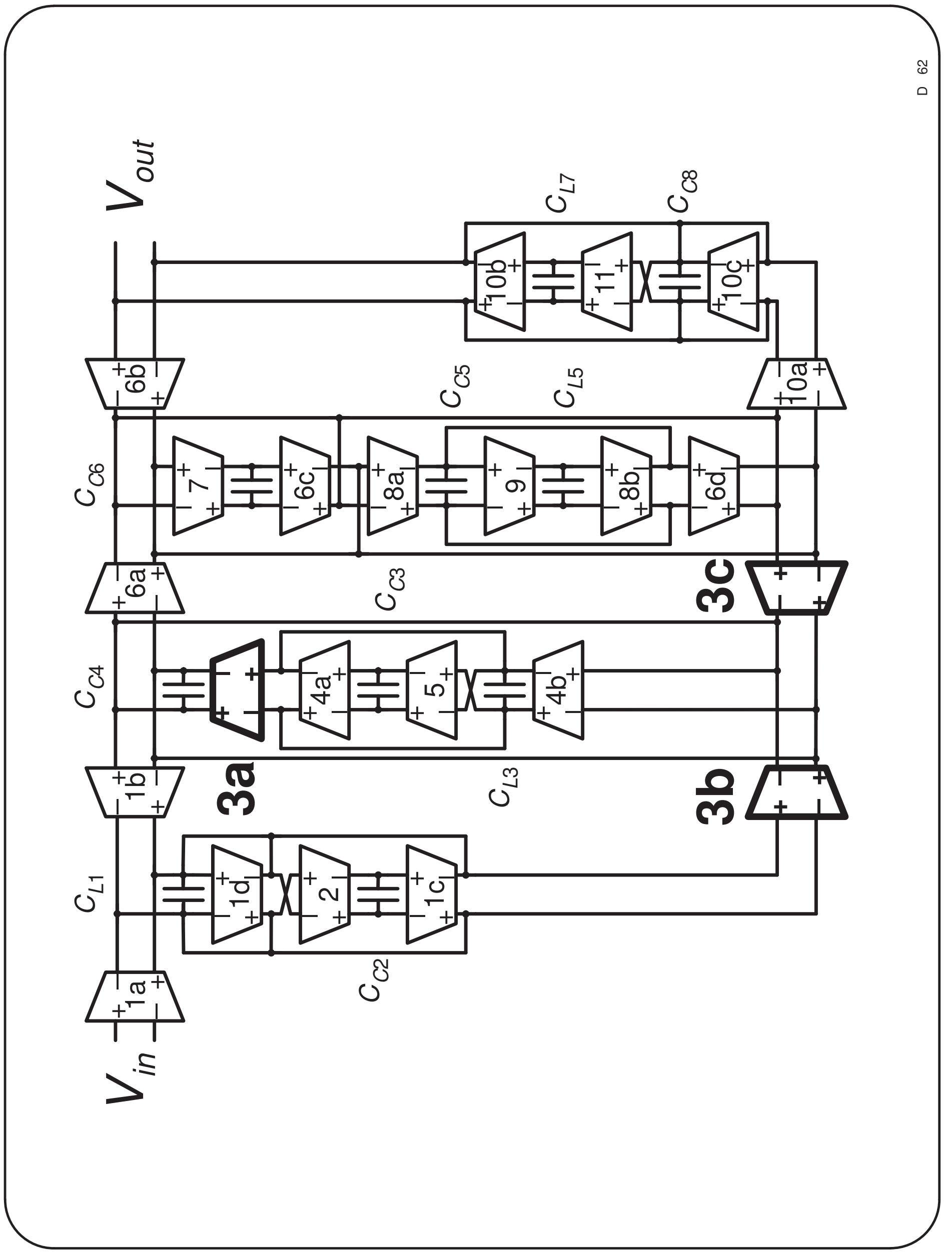


Figure 1.6: *Electric schematic of the ladder filter. It has locally connected topology that is not easily visible.*

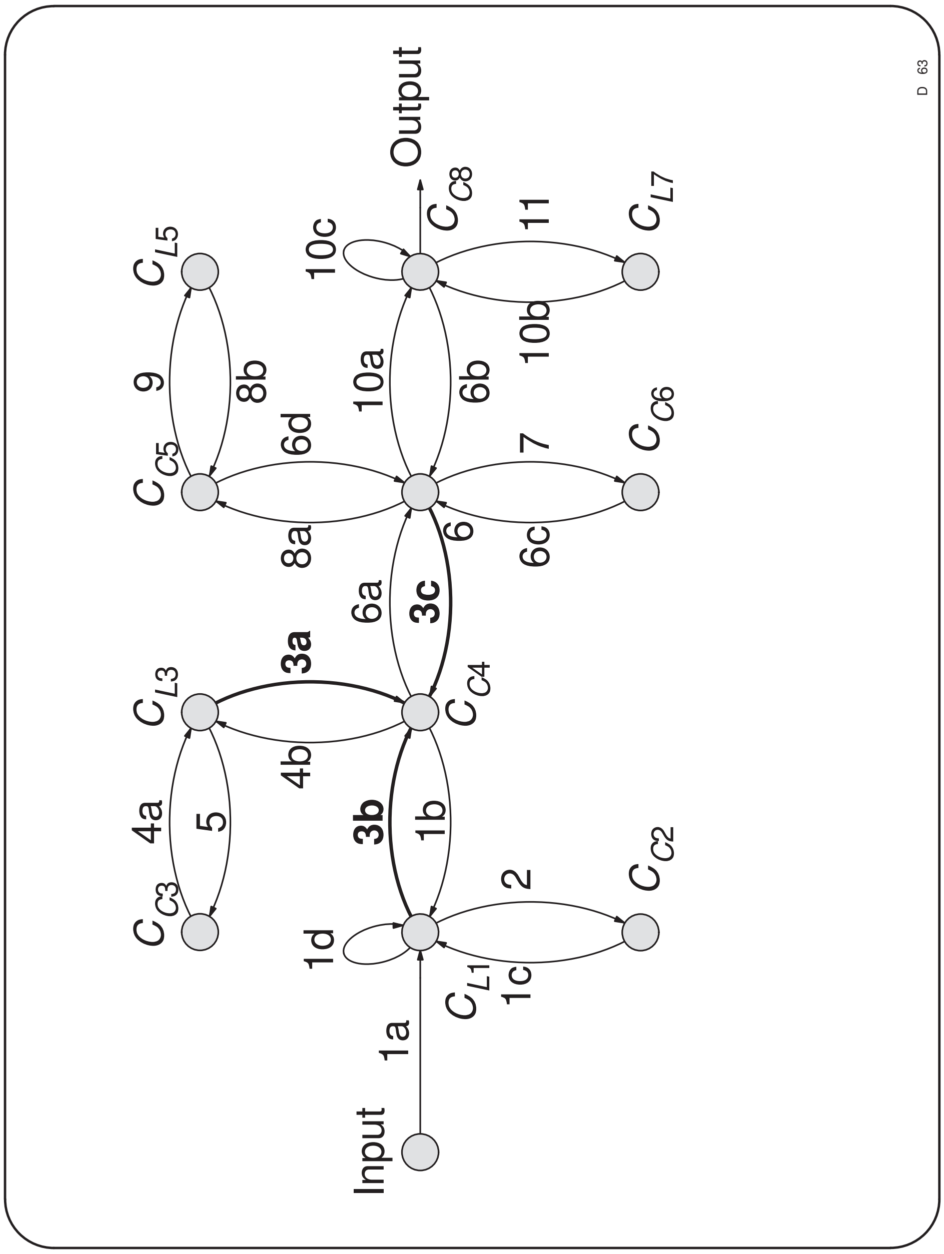


Figure 1.7: *Simplified graph for the ladder filter.*

Fig. 1.6 shows an electrical schematic of an eight-order, elliptic band-pass filter realized as an OTA–C 3 ladder (this schematic results from the so-called OTA–C simulation [49] of an RLC prototype based on the design presented. In a physical realization of the cell the output signal is always limited, i.e. clipping is always present.

At first, the network of the filter does not exhibit much regularity, nor locality of connections. The easiest way to see both is by drawing the graph of connections of the circuit. Since each pair of wires carries one differential signal, it can be represented as a single node of the graph, and each OTA can be represented as a directed edge of the graph (Fig. 1.7). The graph reveals regularity which can lead to a number of different realizations based on regular, locally-only interconnected structures. One particular way of deriving a regular structure for the circuit is by grouping all edges coming into a given node as a single unit (this is a general technique that can be used for arbitrary data-flow-graph, also binary).

All the signal interconnections are local, within the structure provided by the device. Since the input and output terminals are on the sides of the rectangular collection of cells in Fig. 1.7, no extra global connections are necessary for this circuit, which can be simply placed in the corner of the array.

Most of the ladder filters of practical importance can be mapped into the structure of the presented device in a similar way. Second-order (biquad) filters can be easily mapped, too. Since every transfer function can be realized as a cascade of biquads, which can be then put one next to each other in the array, the device provides the way of realizing continuous-time filters by means of local signal interconnections only.

As far as the realization of the whole filter circuit, nodes 1 and 10 would additionally require a feedback connection in one of the OTAs to realize lossy integration. All four OTAs are required only in cell 6.

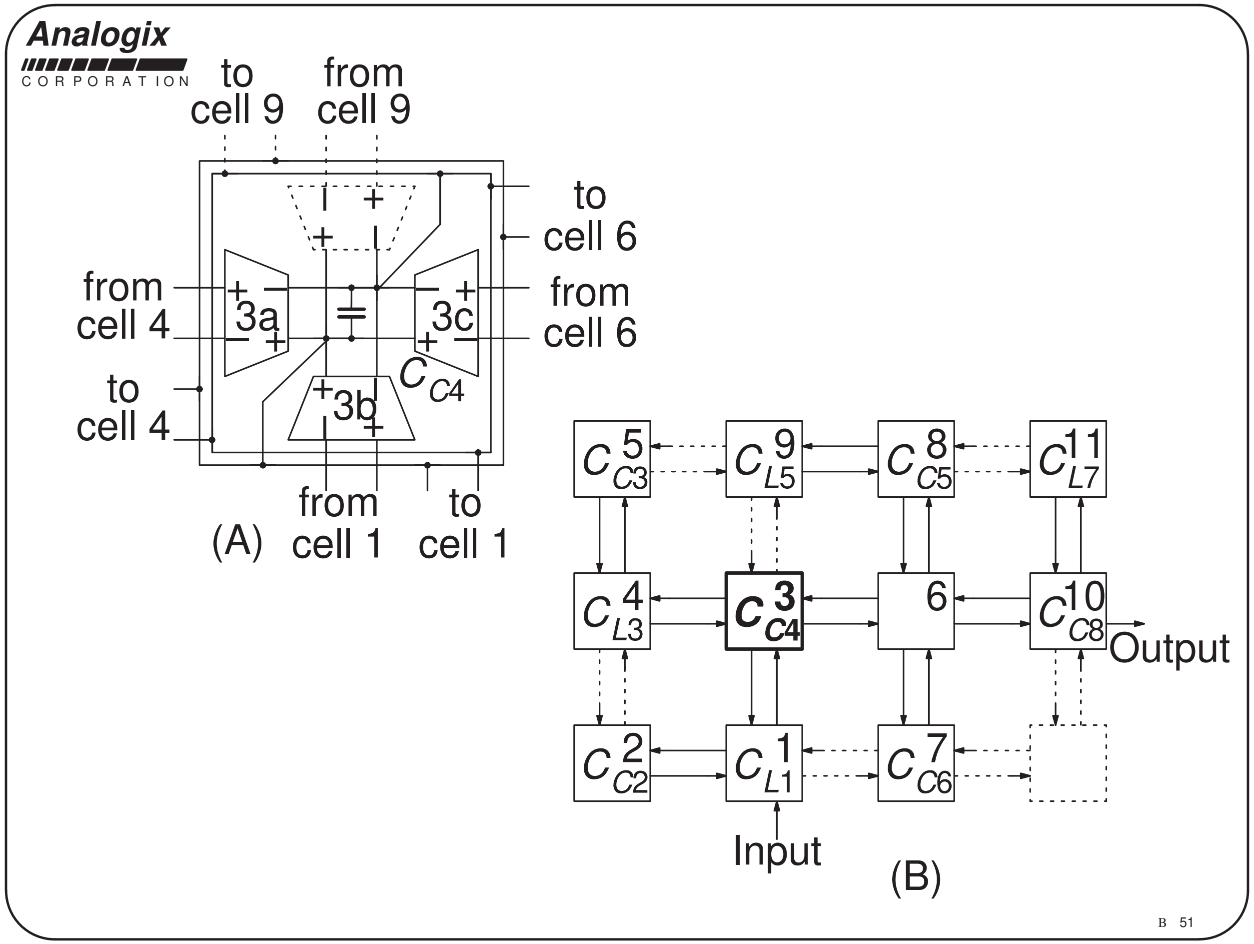


Figure 1.8: *Ladder filter realized in FPAA, observe locality and regularity of connections.*

1.3.6 Rank Filter of Nossek.

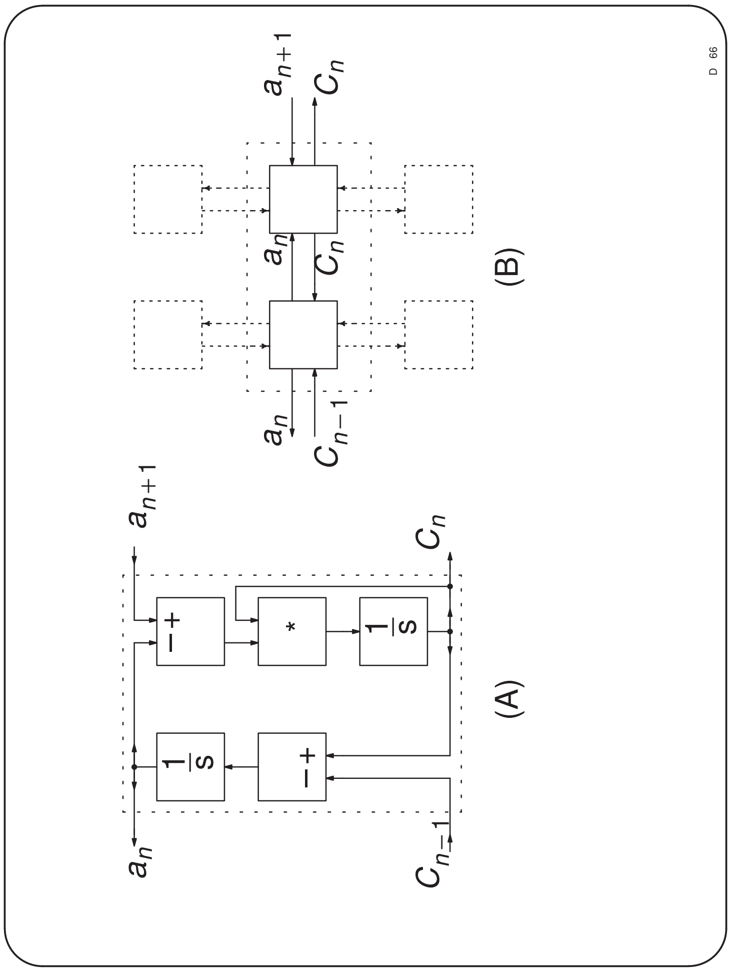


Figure 1.9: *Rank filter.*

Rank Filter is shown in Figure 1.9. Applications are in Digital Signal Processing (DSP) and Image Processing.

1.3.7 Circuits with global connections

Tracking a matrix product

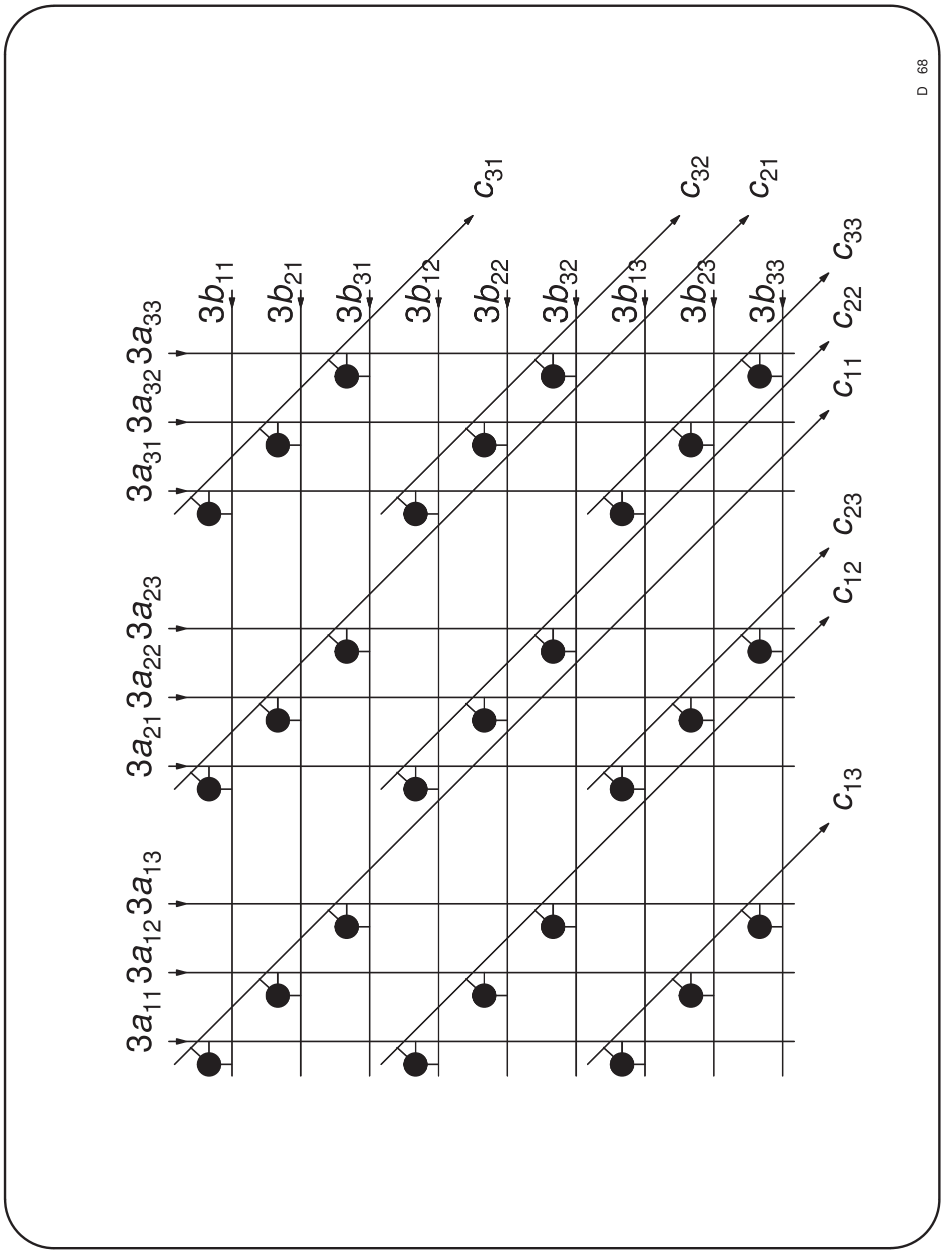


Figure 1.10: *Continuous-time matrix product tracking circuit.*

Fig. 1.10 shows the structure of a matrix product tracking circuit. It takes two time-varying matrices $A(t) = [a_{ij}]$ and $B(t) = [b_{ij}]$,

both 3 * 3, and creates their product $C(t) = A(t) * B(t)$ (a factor of 3 is required to account for the distribution of each input signal to 3 cells; alternatively the gain k of each cell could be increased by the same factor). The circuit can be easily generalized for any rectangular conformable matrices. Each element $c_{ij}(t)$ of the product matrix is produced by a "local" group of cells along a diagonal global signal line. However, to distribute the input signals and to collect the result signals, global connections are necessary. Each diagonal output line is used to sum elementary products $a_{ij} * b_{jk}$, $j = 1, ..., n$, comprising the product element c_{ik}.

It is instructive to note that the "globality" of connections results primarily from the need to distribute input signals, and collect the output signals. Creation of each matrix product is done "locally" (although using global signal lines). What is important also, is that global lines are used here only at the "terminals" of the circuit, i.e. for the input and output signals. Global lines are not involved in transmitting internal signals of the circuit. Global interconnections of this kind will be called here the **Class 1** global connections. In general, Class 1 contains global connections not involved in internal feedback signals in a circuit.

Now, imagine a massively parallel chip of the future that will use this approach to track several matrix products in parallel. It will find applications to control, for instance, a robot with many-jointed arms, one matrix for each joint or degree of freedom. This would be a good application of FPAA.

Tracking the solution of a system of linear equations

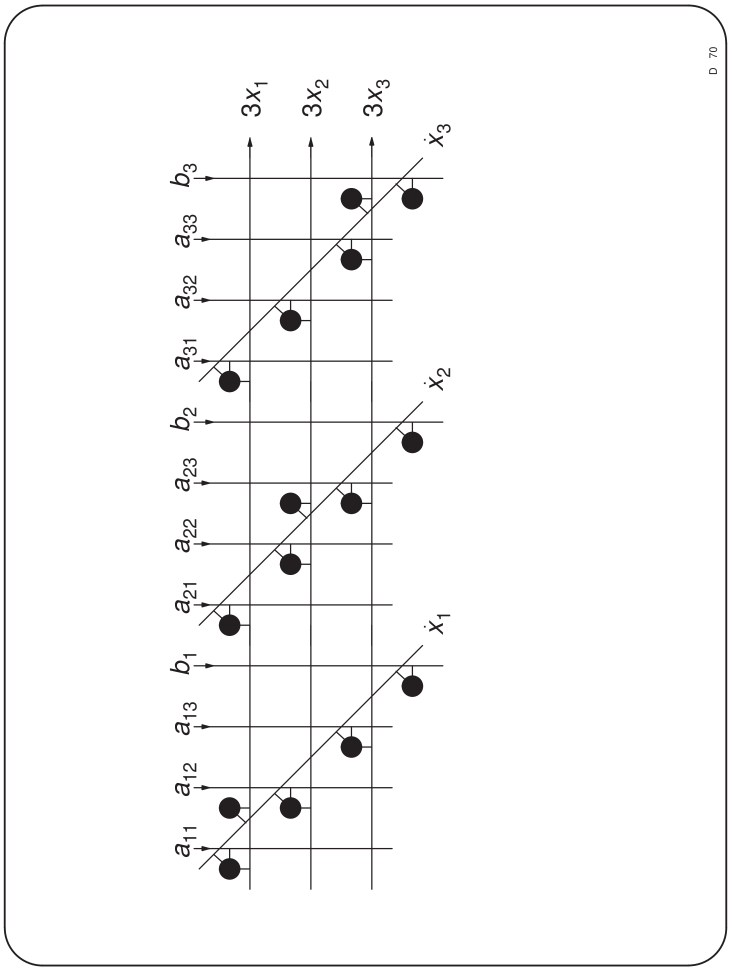


Figure 1.11: *Continuous-time circuit for solving a system of linear algebraic equations.*

Modifying slightly the matrix product tracking circuit, we can build a circuit tracking the solution of a system of linear equations. The solution $x(t)$ of the system

$A(t) \ * \ x(t) \ = \ b(t)$

can be found by solving a system

$\dot{x}(t) = A(t) * x(t) \ - \ b(t) = 0$

of differential equations provided that matrix $A(t)$ is always positive stable [21]. In many practical cases matrix A will be time-invariant, but it is instructive to see the solution of a more general problem, i.e. with a time-varying matrix $A(t)$. Fig. 1.11 shows a circuit solving a system of 3 equations with 3 unknowns $x_1(t)$, $x_2(t)$, $x_3(t)$. The global connections in this circuit carry internal feedback signals, although the distance travelled by these signals is small. Such connections will be called Class 2 global inteconnections.

This circuit will find applications in image processing and solving complex motion or space conflict problems, as well as optimization problems.

Tracking the solution of a linear programming problem

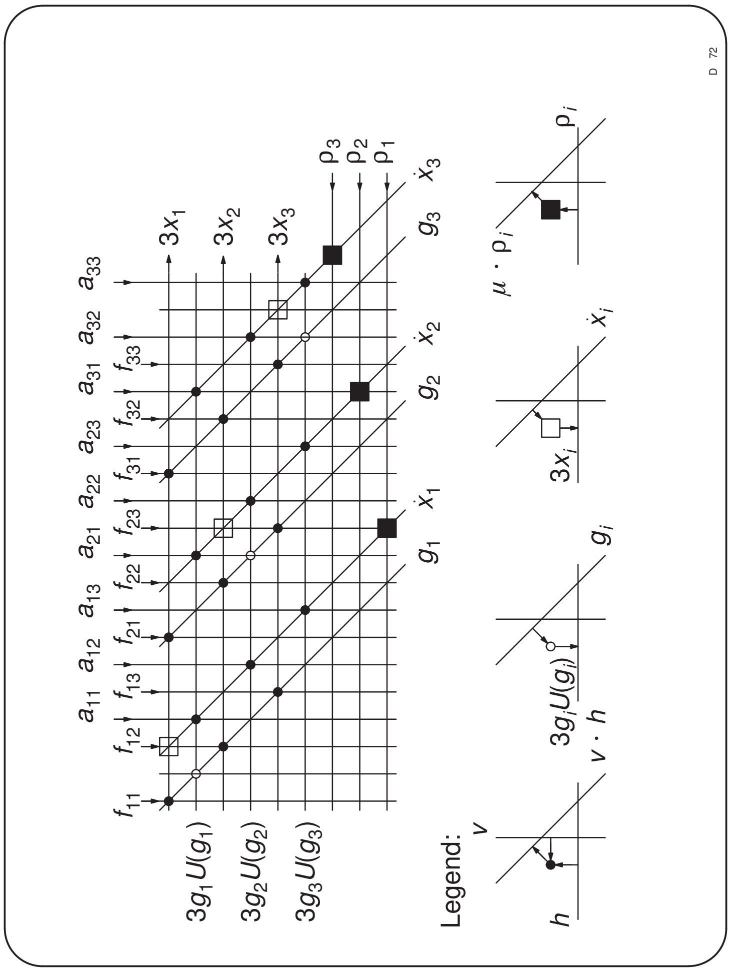


Figure 1.12: *Tracking the solution of a linear programming problem.*

A linear programming problem can be stated as follows. Given a set of constraints

$g(t) \;=\; F(t) * x(t) \;=\; [g_1(t), ..., \; g_m(t)]' \;\leq\; 0$

(the inequality is supposed to hold for every element of the vector; F is a rectangular matrix of constraints coefficients, g is a vector representing individual constraints), minimize the objective function

$\varrho(x_1, \; ... \; , x_n) \;=\; \varrho \;\cdot\; x \;=\; \varrho_1 \; x_1 \;+\; ....\varrho_n \; x_n \;,$

where $\varrho \;=\; [\varrho_1, ..., \; \varrho_n]$. Application of the method of steepest descent leads to the system of equations

$\dot{x} \;=\; -\,\mu 2a \; A \; diag(g) * U(g)$, where

$U(g)$ denotes the step function, $diag(g)$ denotes a diagonal matrix with elements of vector g on the main diagonal, and μ and a are constants ($\mu \Longrightarrow 0,\ a \Longrightarrow \infty$) [21]. (Edmund add more explanation and variants). This system can be solved by the circuit shown in Fig. 1.12. In the case of linear constraints matrix A will be identical to matrix F, nevertheless a more general circuit not assuming this equality is shown as an illustration of the versatility of the device. A simplified circuit, with only matrix F input, can be easily derived.

Among other applications of such circuit would be the telephone traffic control.

1.3.8 Fuzzy and Continuous Functions.

Fuzzy logic and continuous logics (such as Lukasiewicz logic) [35, 36] can be realized as well. As an example, let us consider an implementation of a fuzzy logic controller with correlation-product inference [22]. A structure very similar to that of PLAs and generalized PLAs, as shown in Figure 1.13a, is used to implement a controller with m input variables and n fuzzy inference rules. Figure 1.13b shows details of each rule implementation. Fuzzy membership function is implemented as a trapezoidal transfer function. Activation values w_i are multiplied by centroid values of the fuzzy rules consequents c_i, and their areas I_i, yielding two sums computed on two horizontal global lines. The final expression for the defuzzified output variable v_k is produced by a two-quadrant divider [21], shown in Figure 1.13c. Observe that the connection pattern is a subset of that from Figure 1.1.

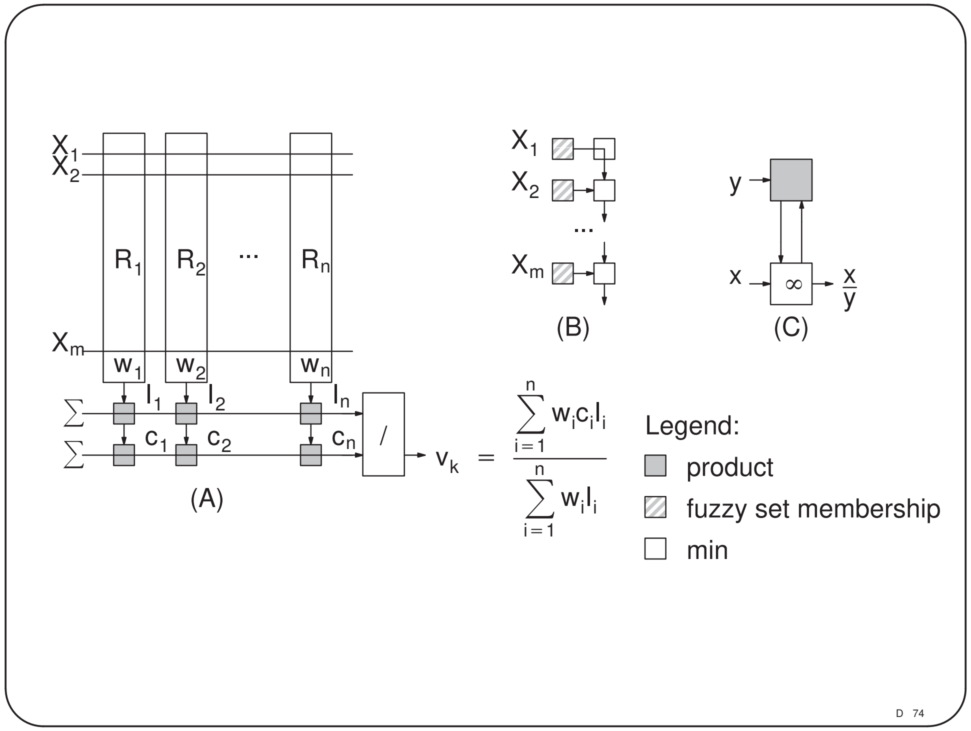


Figure 1.13: *Circuit for Fuzzy Logic*

1.3.9 Realization of typical analog sub-systems

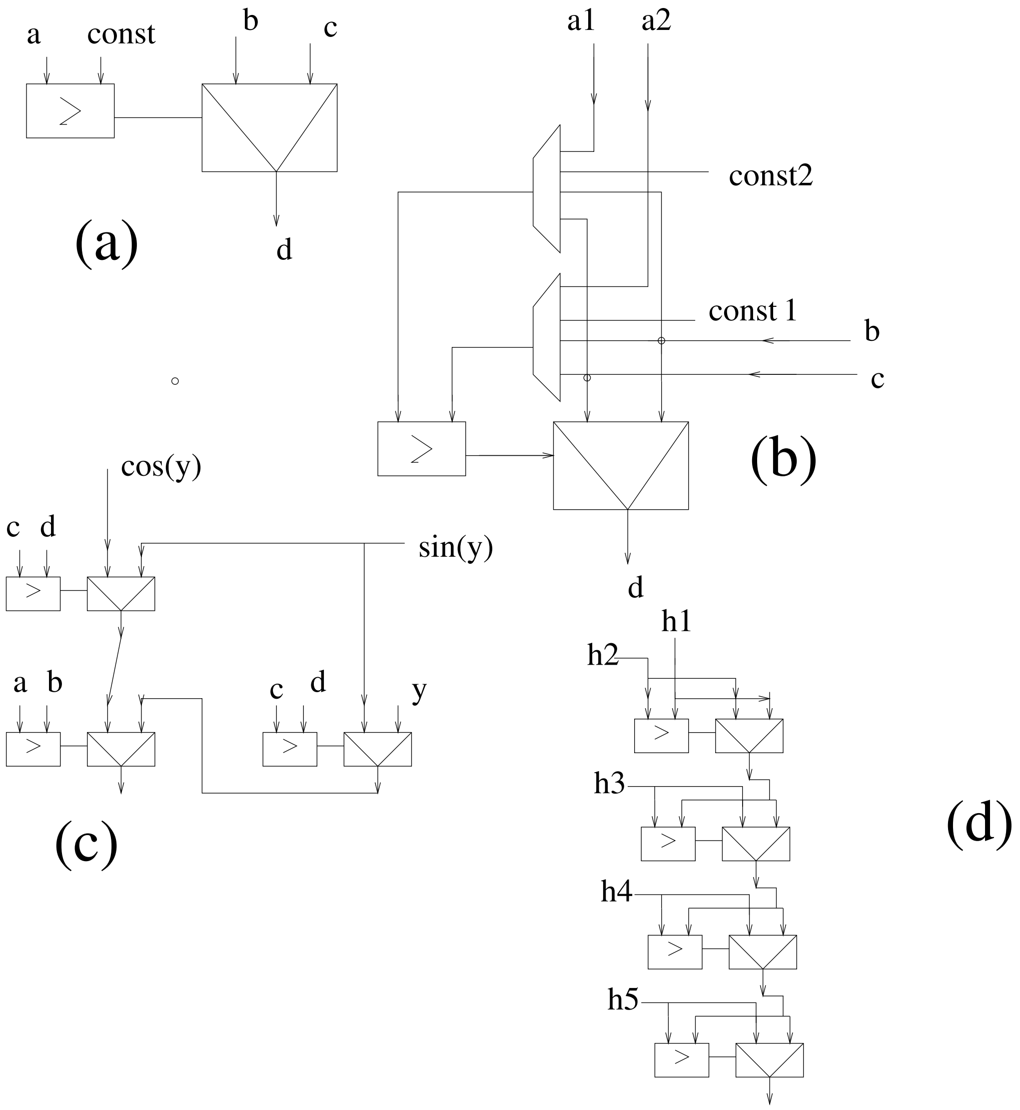


Figure 1.14: *Realization of analog functions.*

Hierarchical design of **iterative** one- and two-dimensional structures is possible, which are cellular connections of "analog logic" blocks, each block realized as a multi-output rectangular diagram. This can be done also for discrete circuits with memory. Analog counterparts use sample-hold analog memories, which play the same role as (binary and MV) flip-flops in discrete technologies. Regular Diagrams allow thus the realization of cellular memory-less functions, finite state machines, and infinite state machines; realized in analog, binary, or multivalued logic. For instance, the digital and analog: filters, pipelined image processors, or systolic processors. An elliptic ladder filter was mapped to this structure, shown above, as just one typical example of a general mapping possibility for analog dynamic circuits. Fig. 1.14a shows a basic circuit with analog comparator and analog multiplexer, and Fig. 1.14b a full cell with SRAM-controlled muxes to switch the inputs, as again, only one example of mapping static analog circuits.

Two simple rectangular diagrams for analog functions are shown in Fig. 1.14c,d. Fig. 1.14c presents a diagram realization of the piecewise continuous function **if** $((c > d)$ **and** $(a > b))$ **then y else if** $((a \leq b$ **and** $(c \leq d))$ **then cos(y) else sin(y)**. Fig. 1.14d shows a diagram realization of analog function MAX$(h_1, h_2, h_3, h_4, h_5)$. Similar realizations can be created for rank and median filters, cellular neural nets, equation solvers, and (analog and digital) image processing circuits. Below we will derive how this kind of circuits can be calculated in general. Under certain restrictions, like variable ordering or expansion selection, the canonicity of some of structures, such as the one from Fig. 1.14c, can be proved.

Now, that we have found basic analog structures being structural counterparts of digital binary circuits: Generalized PLAs, 1-dimensional iterative circuits, matrices, and rectangular diagrams, let us consider how these ideas can be applied to "logic" circuits on various levels of their description.

Chapter 2

Expansions for Multiple-Valued Logic

2.1 Expansions as a base of function calculation in regular layouts.

2.1.1 Expansions that generalize Shannon to Multiple-Valued Logic

The fundament of our approach to **layout-geometry-driven structure design of combinational and sequential circuits** are **expansions** of functions, i.e. operators that transform a function to few simpler functions. We restrict our attention first to combinational circuits (no memory, analog or digital).

For instance, in a well-known canonical Shannon expansion function f is expanded with respect to input variable a as follows:

$$f = af_a + \overline{a}f_{\overline{a}}$$

where $f_a = f\mid_{a=1}$, and $f_{\overline{a}} = f\mid_{a=0}$ are **the positive and negative cofactors** of function f with respect to variable a, respectively.

In the DAG-based expansion, the cofactor functions that are tautological (complete or not complete tautology can be considered) are combined to single nodes. Nodes for functions f_a and $f_{\overline{a}}$ and bus for variable a are mapped to layout, and the procedure is repeated for the next input variable.

As shown in Kohavi's book [24] and Aker's paper [1], this way, any single-output symmetrical binary function can be directly mapped to regular layout with 1,2,3,4,... nodes in successive levels corresponding to input variables.

Here we will show how to extend this approach to **arbitrary** binary functions, not necessarily symmetrical.

As shown below, **other expansions** of functions can also be used, and the expansion nodes are mapped to the neighborhood structures which are more powerful than those investigated theoretically in the past [1], but similar to those from commercial Fine Grain FPGAs.

Our concept of a **"regular diagram"** is based on three ideas: **forward expansion** (distributor), **reverse-expansion** (collector), and **neighborhood geometry** (structure to which expansions are mapped). All our design methods for circuits without memory, discussed above or below, can be derived by combining these ideas.

1. **Forward Expansion** of a node function creates several successor nodes of this node. Function f corresponds to the initial node in the diagram, initially a tree. Thus a function is "decomposed" or "distributed" to smaller functions that create a **structure** of a tree (binary or K-ary).

2. **Reverse Expansion** operation merges several nodes at the bottom of the regular diagram (before the reverse expansion, this level looks like a tree). This is a reverse operation to standard expansion. Similarly as there are several standard expansions, there are also many reverse expansions, some we will discuss below.

3. **Regular neighborhood geometry**, to which the nodes are mapped, guides which nodes of the level are to be combined to keep the connections local. It is a constraint for mapping to the 2D space (and in future, to the 3D space).

Every signal in the regular layout can be treated as multi-valued (particularly, binary). A multivalued (MV) connection for logic with 2^k values can be realized by k binary wires which comprise a bus (making the diagram "fat") and encode the multi-valued signal to binary. In general, a connection with r^k values can be realized by k $r-valued$ wires which comprise a bus and encode the multi-valued signal to several multi-valued signals, if necessary.

Characterization of regular structures.

The well-known structures are:

1. **Fat Trees**, see Figure 2.1.
2. **Generalized PLAs, PALs, GALs, PLAs, ROMs**, see Figure 2.2.
3. **(generalized) Maitra cascades, Fat Maitra cascades, etc.**, see Figure 2.2.
4. **Akers Arrays [1]**. see Figure 2.2.

We can show that they are all just special cases of our powerful concept of regular layout.

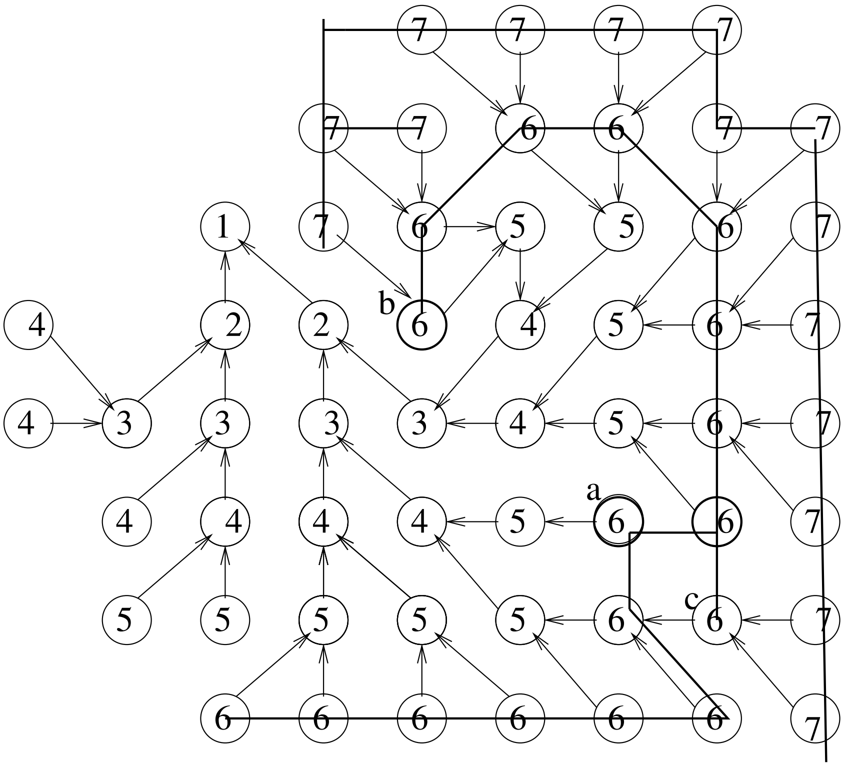


Figure 2.1: *An (incomplete) tree drawn on a 8x8 grid. Observe nodes a, b and c that have no space for both the predecessor nodes. The connections buses for levels 6 and 7 are also drawn.*

Figure 2.1 presents example of a tree. Because tree grows exponentially, in general it cannot be drawn using 8x8 grid. But because real-life trees are incomplete, much of the connections can be completed. The reader has to observe the repeated variable in level 6 and nodes that have no space for predecessor nodes. If the lines were interpreted as not single wires but several wires, Figure 2.1 would represent a Fat Tree.

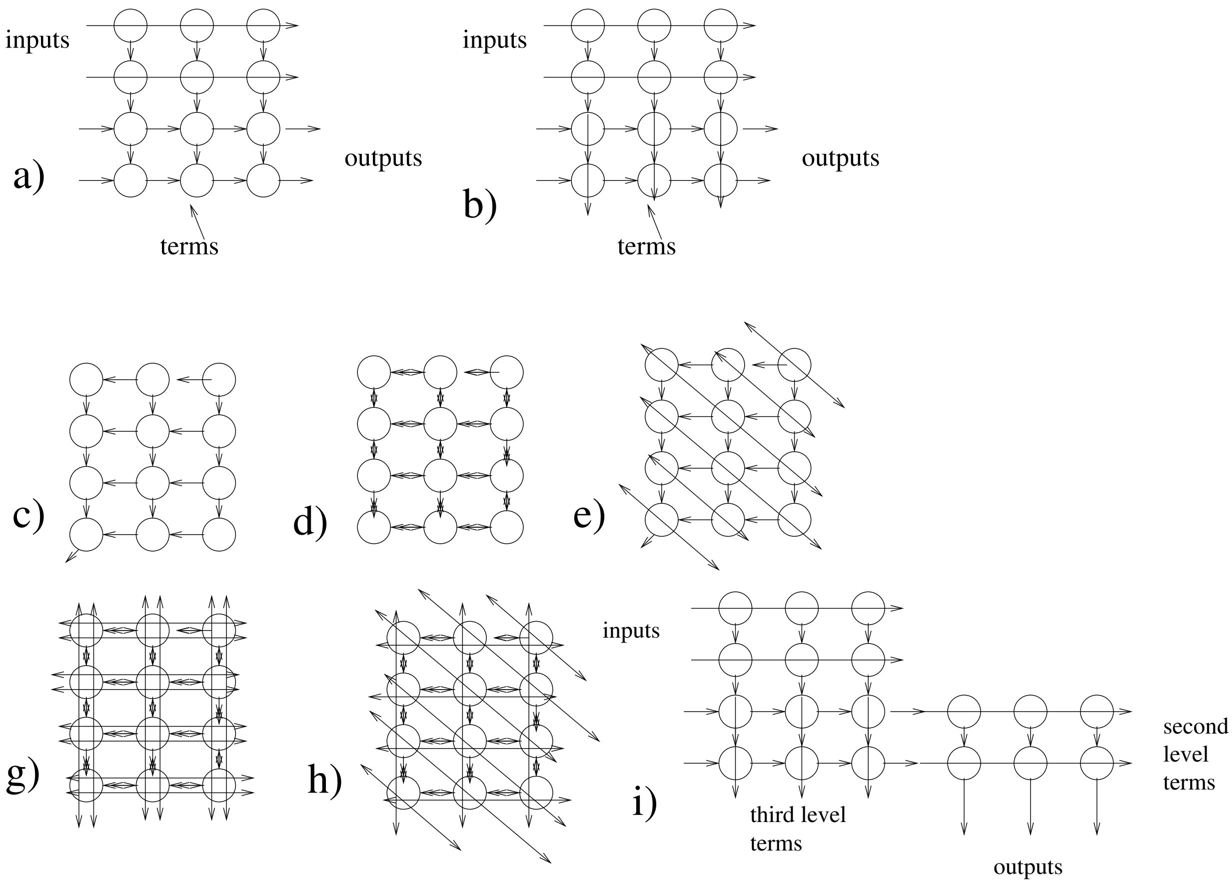


Figure 2.2: *Various regular structures: (a) Generalized PLA of the first type, (b) Generalized PLA of the second type, (c) 2x2 pattern without input buses, (d) 4x4 pattern of FPAA and a computer network of type "mesh", (e) 2x2 pattern of a regular diagram for binary logic with oblique buses, (f) 4x4 pattern of regular diagram with horizontal and vertical buses (Concurrent Logic Fine Grain FPGA architecture), (g) 4x4 pattern of regular diagram with horizontal, vertical and oblique buses (our FPAA) (h) three level PLA for TANT network [63].*

Figure 2.2 presents various regular structures that are either well-known or have been presented above. Figure 2.2a) presentes a Generalized PLA of the first type. There is no bus through the collecting plane, only cell-by-cell abuting in it. This Generalized PLA can be a standard PLA (SOP or POS), EXOR PLA, or an EXOR of Maitra Cascades. Figure 2.2b) represents a Generalized PLA of the second type, with buses from the cutting plane going through the collecting plane. Figure 2.2c) is a 2x2 pattern (it means, two inputs and two outputs) without input buses. Such pattern can be used in systolic processor or with logic gates inside circles. Figure 2.2d) shows a 4x4 local pattern of FPAA and a computer network of type "mesh", this structure has no buses. It can be used for filters and other analog circuits and to design autonomous and regular finite state machines. (Such machines are not general-purpose state machines). Figure 2.2e) is a 2x2 pattern of regular diagram for binary logic with oblique buses. This will be our main pattern to explain expansions. Analogous pattern can be drawn for 4x4 unidirectional regular diagrams for quaternary logic but the explanations become more complicated. Figure 2.2f) shows a 4x4 pattern of a regular diagram with horizontal and vertical buses.

This corresponds directly to the recent (Digital) Fine Grain Field Programmable Gate Array from Concurrent Logic Inc. Figure 2.2g) illustrates a 4x4 pattern of regular diagram with horizontal, vertical and oblique buses. This is a complete pattern of our FPAA. Thus, pattern-wise, with a addition of one vertical and one horizontal bus our FPAA would include all previous connection structures shown above, and also many other similar structures, some of which perhaps have been already used in some designs but we are not familiar with them. Finally, Figure 2.2i) shows a three-layer PLA to realize TANT networks [63], which is similar to a two-layer standard PLA, but has three layers of NAND gates. Similarly, multi-level generalized PLAs can be realized for arbitrary number and type of planes.

Maitra cascades are linear sequences of gates AND, OR and EXOR in any order. Fat (generalized) Maitra cascades are linear sequences of cells (iterative circuits) and have more than one carry signal between the subsequent cells. Akers arrays look as the array from Figure 3.6d), but has inputs repeated many times in oblique buses, always for the worst case.

In addition, observe please, that our regular structure concept extends also some structures that are described in the Concurrent Logic Inc. patent application and also some new non-published patents that we are aware of. It has to be seen if our claims for the **most general structures**, regardless of analog, digital or another realization are patentable.

It should be stressed, however, that we are patenting not only a structure but we will show below constructive and efficient methods of designing discrete and continuous functions in these structures [40].

In past, we showed on many examples [40, 42, 47] that this geometry is very powerful and more universal than the previously investigated general cellular structures. Here we will further extend and unify these notions to expansions **with more than 2 successor functions**, and **geometries with more than 4 neighbors.** In theory the diagram can be extended to any number of neighbors of a cell. But, based on our experiences so far, we believe that 8 is practically enough. Observe, that our structures are not always planar, some of them include non-planar circuits (like a 4 node clique), to allow realization of an arbitrary graph. But, they are **always** regular.

The main principle of forward and inverse expansions.

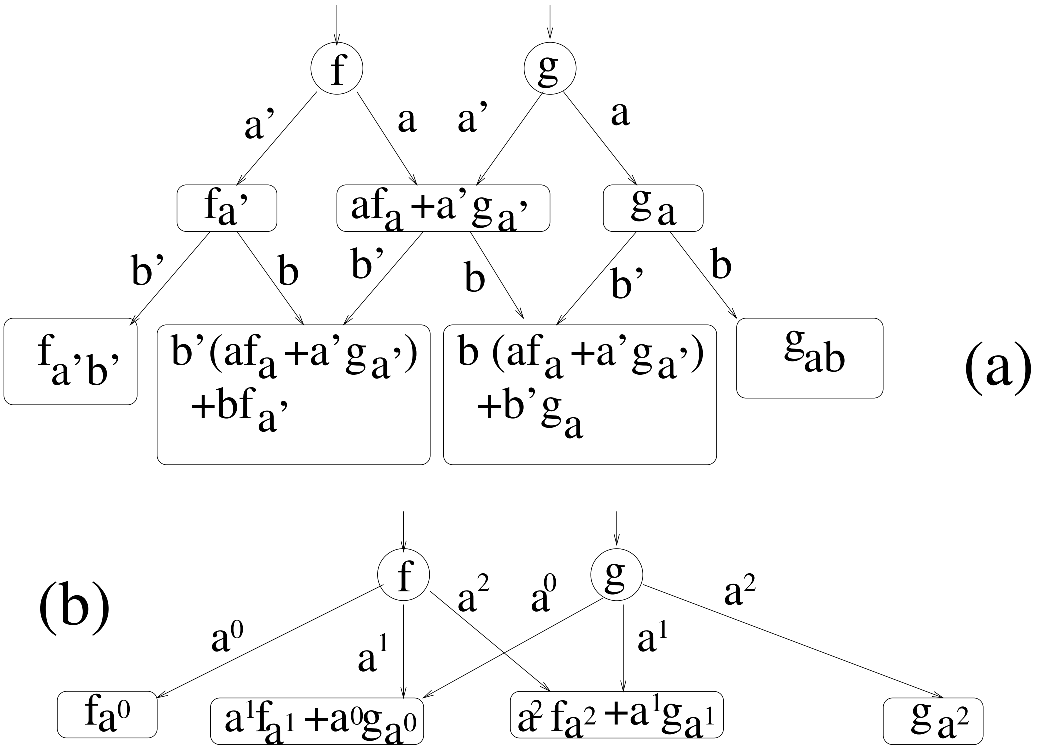


Figure 2.3: *Forward and Reverse Expansions: (a) Shannon in binary logic, (b) Ternary Post, i.e. Post-like expansion in 3-valued logic; a generalization of Shannon of MAX type.*

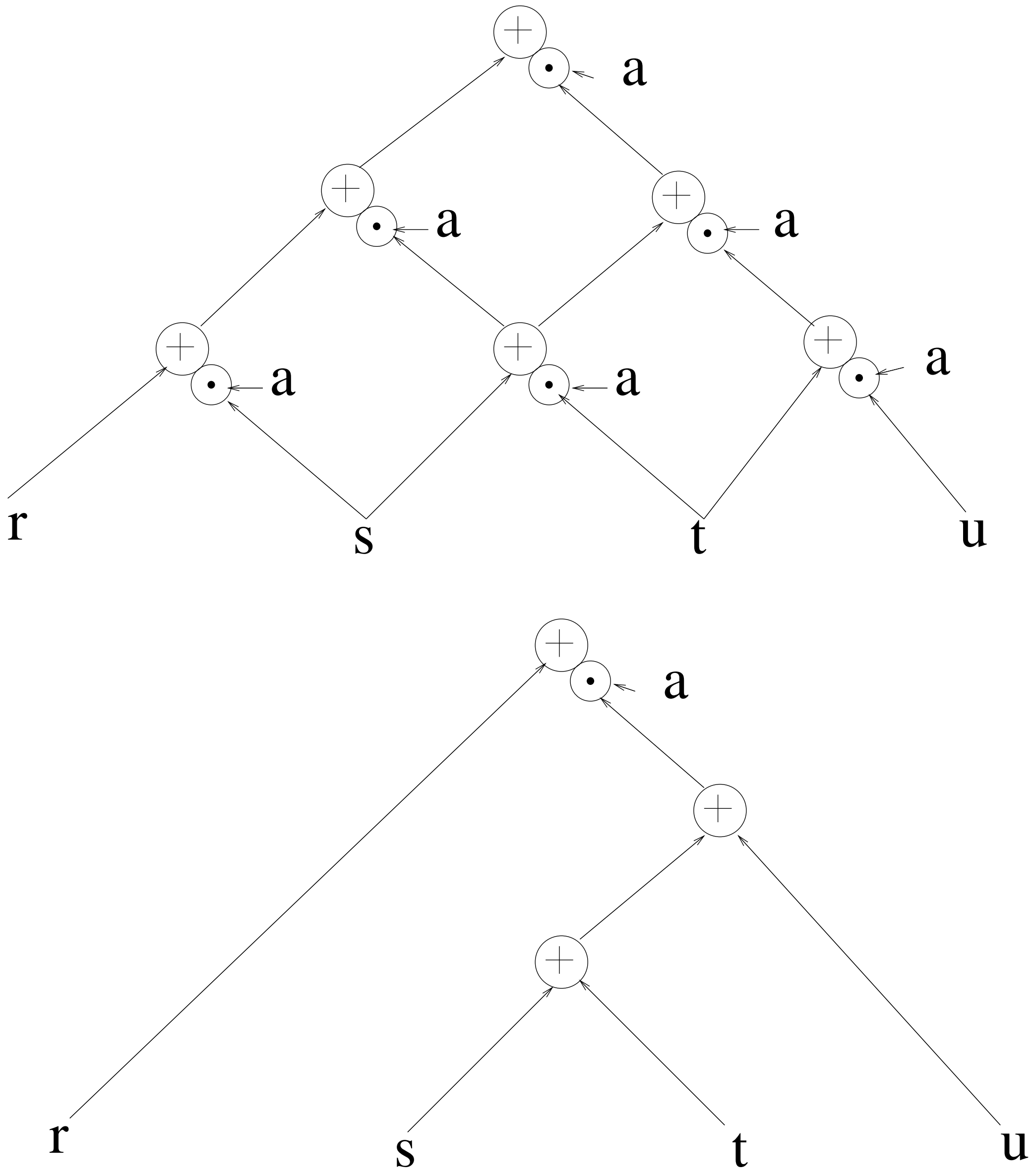


Figure 2.4: *Explanation of trees inside regular layouts.*

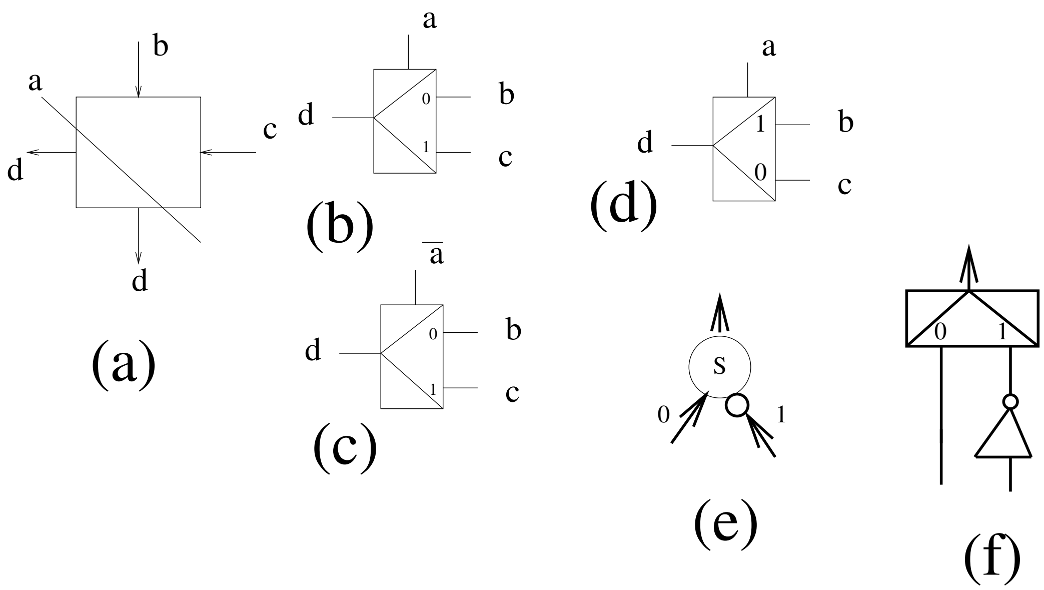


Figure 2.5: *Expansions.*

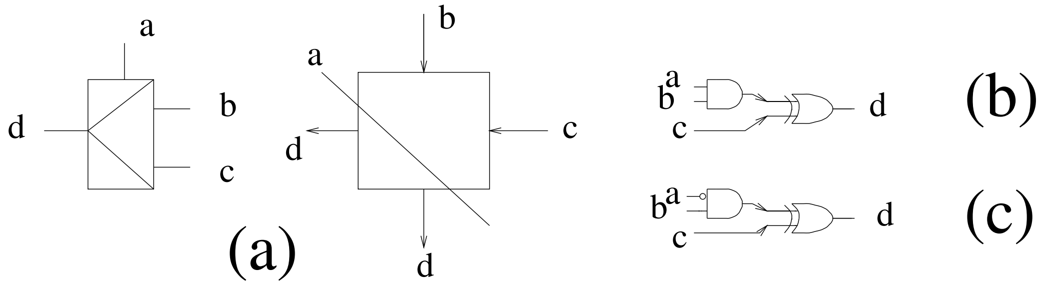


Figure 2.6: *Expansions.*

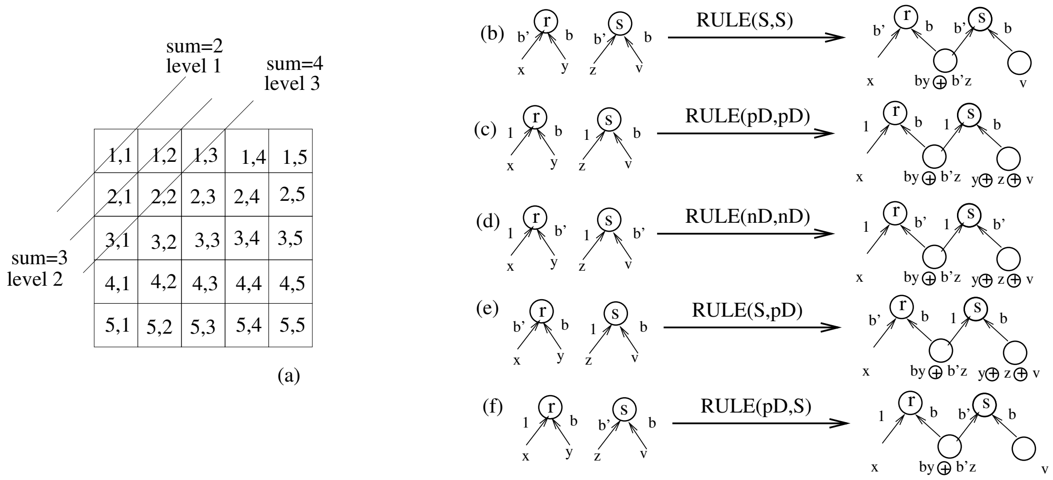


Figure 2.7: *Explanation of data structure and reverse expansions.*

Figure 2.3 presents two expansions, together with their inverse expansions. One is the familiar Shannon expansion with added reverse expansions. The next is its equivalent for Post Logic. As we see, the same principles can be used for any number of values, if the signals in literals are disjoint. We will call this MAX-type expansions. Observe that "+ operation" in Figure 2.3a is OR-ing or EXOR-ing. EXOR-ing is a field operation of addition in GF(2). Observe, the "+ operation" in Figure 2.3b is maximum (MAX) in Post algebra with 3 values, thus not a field. But the same principle of collecting is used in both inverse expansions.

The method to create Shannon Regular Diagrams can be easily expanded to MV Shannon expansions and associated Post Regular Diagrams for multi-output incomplete functions (see Fig. 3.5b). In 3-valued logic, each single-variable expansion cuts a function's map to three v-cofactors, and any two of them can be next recombined by a reverse expansion operation MAX - Fig. 3.5b. We call this Forward and Reverse Post Expansion and Post Regular Diagram.

MAX is the maximum operation denoted by $+$. Let us observe that disjointness of literals a^0, a^1, a^2 is the fundamental condition that must be satisfied to create maximum-type regular diagrams. It is a special case of Orthogonality of functions used in orthogonal expansions (which will be presented in the next section). Observe that symbol $+$ in Fig. 2.3 can denote various operations but the meaning and the method of Forward and Inverse Expansions and the principle of their duality remain the same.

What are all these operations?

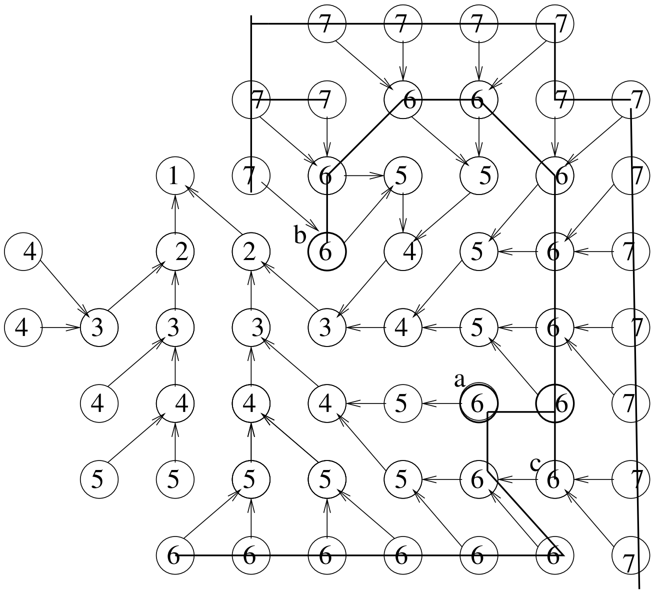


Figure 2.8: *Fat Trees and others regular structures.I.*

2.1.2 Binary Orthogonal Regular Diagrams

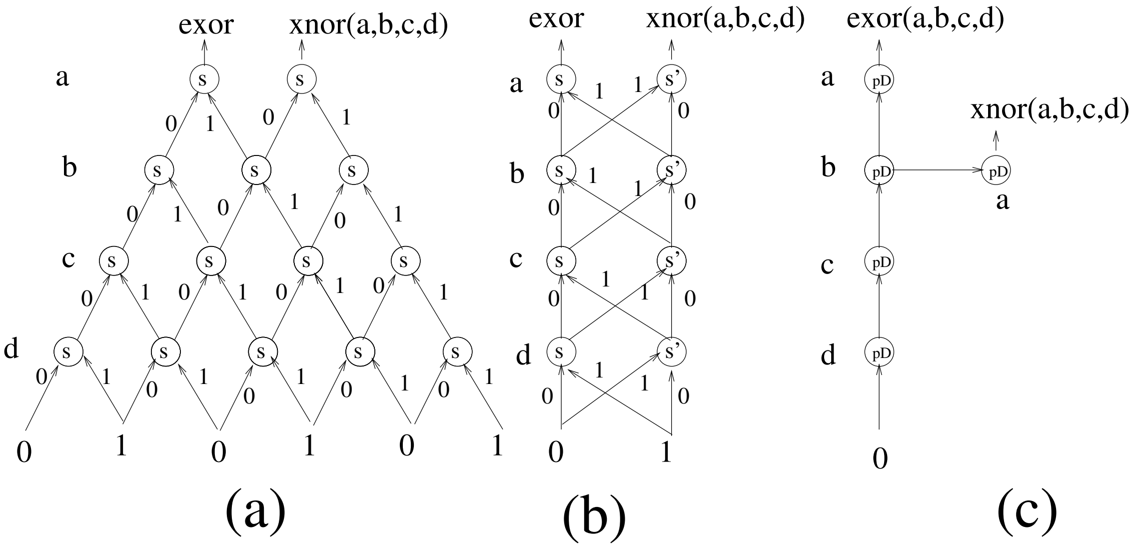


Figure 2.9: *Comparison of three types of Regular Diagrams for two-output EXOR/XNOR function.*

Similarly, as we invented the Functional Kronecker diagrams [67] by the generalization of the idea of BDDs, here we generalize the MAX-type regular diagrams from Figure 2.3 to "orthogonal" or "Kronecker" regular diagrams. They have in general much smaller areas and less gates. For instance,

Fig. 2.9 presents a comparison of sizes of a standard binary Shannon regular diagram and two new types of regular diagrams for the EXOR/XNOR two-output function.

Fig. 2.9a presents a solution that would be obtained using the standard Shannon regular diagram or the (very inefficient in this case) basic Akers Array (that in general has repeated variables [1]). The order of control variables is a, b, c, d. Because the function is symmetric, variables are not repeated. Observe that arrows with 0 (for negated control variable) are always in left. The shape is a trapezoid and the size is 14 nodes. Connectivity pattern is 2x2. The Akers Array [1] would have (5 * 5) * 2 nodes (it realizes each of two functions separately, and uses a 5 * 5 fixed square for a 4 variable function).

Fig. 2.9b presents our solution with 3x3 connectivity pattern array of multiplexers. It is linear in shape and has 2 * 4 = 8 nodes. In addition to Shannon (S), the Shannon expansions with negated control variables (S') are now used. Observe that arrows from the left have both 0 and 1 values.

Fig. 2.9c presents Positive Polarity Reed-Muller Regular Diagram 2x2 connectivity pattern array of positive Davio (pD) nodes. It is nearly linear in shape and has 5 nodes. This figure clearly demonstrates an advantage of having higher connection patterns and more general expansion types. Predictability and equality of delays should be appreciated in all regular diagrams.

The functions in Fig. 2.9 where symmetric, but **what about regular diagram realization of non-symmetric functions?**

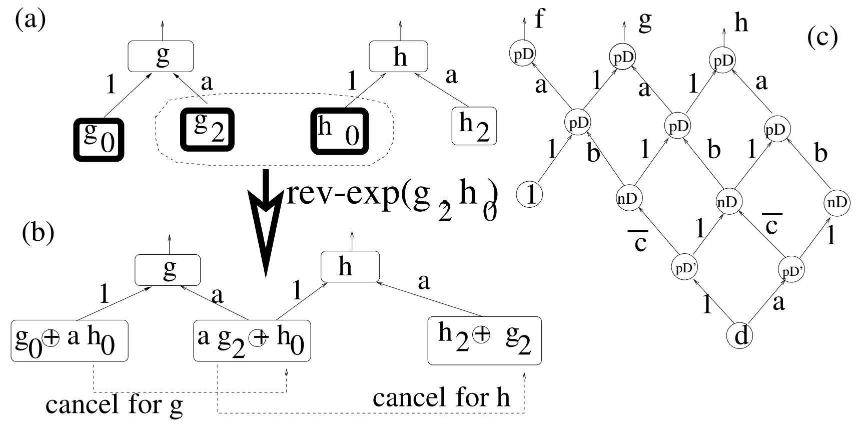


Figure 2.10: *Creation of a Positive Davio level in a Regular Diagram: (a) two expanded nodes before reverse expansion, (b) layer of regular diagram after reverse expansion of nodes g_2 and h_0, (c) Fixed-Polarity RM Regular Diagram for functions f, g, h.*

- We can observe that more general symmetries can be consider when all types of expansions are used, not only Shannon, which is always considered in symmetries. These new generalized symmetries are much more general than the known symmetries of functions, so using them, more functions can be put to regular diagrams without repeating variables.
- Functions that do not have these more general symmetries can be still realized in regular diagrams with repeated variables.

This requires, however, to use reverse expansions for nodes. Figs. 2.10a,b present the principle of reverse expansion operation for EXOR-based, and in particular orthogonal logic. Although it is shown here only for pD nodes and an ordered regular diagram, the same principle is used for more complex expansions and regular diagrams of the orthogonal type. In Fig. 2.10a illustrates the local situation in a level of regular diagram after using pD expansion with respect to variable a to nodes g and h.

In this figure, $g_0 = g(a=0)$, $g_1 = g(a=1), g_2 = g_0 \oplus g_1$ are the negative and positive cofactors and Boolean difference, respectively. Fig. 2.10b presents the result of reverse expansion on the successor nodes g_2 and h_0. The reverse expanding rule is: $g_2 \; REVERSE - EXPANSION \; h_0 = ag_2 \oplus h_0$, which means that nodes representing functions $g_2 = g_0 \oplus g_1$ and h_0 are combined to a new node representing function $ag_2 \oplus h_0$. The **correction terms** ah_0 and ag_2 are propagated to left and right, respectively. It can be easily checked, that because of the term cancelling (based on principle $x \oplus x = 0$), in the diagram level of variable a from Fig. 2.10b the pD expansions of g and h are still satisfied: $g = g_0 \oplus ag_2$, and $h = h_0 \oplus ah_2$.

Fig. 2.10c presents Fixed-Polarity Reed-Muller Regular Diagram (expansions pD and nD) for functions:
$f = a \oplus ab\bar{c}d$,
$g = 1 \oplus b\bar{c}d \oplus a\bar{c}d \oplus abd \oplus ab\bar{c}d$,
$h = \bar{c}d \oplus bd \oplus ab\bar{c}d \oplus a\bar{c}d \oplus abd \oplus ad$.
Variable a is used first in pD level on top of the diagram from Fig. 2.10c. As shown in Fig. 2.10c, expansions for variables a and b in first two levels are pD, and the expansion for variable c in the third level is c. Variable a is repeated once more in the bottom level of the diagram. The expansion in this level is pD', which means, a reversed pD, that is a pD expansion with reversed role of data inputs. Observe, that although the function is not symmetric in a standard way, it is diagram-realizable without variable repetitions because it has polarized Pseudo-Kronecker symmetries. In some types of expansions the propagation of correction terms is only to right, or only to left. In some other expansions, especially the non-canonical ones, more powerful corrections types are created, and the algorithm selects the correction rule evaluated as the one leading to the simplest next level of the diagram. Selecting the order of (repeated) variables and the expansion type in each node are the most important and difficult problems to be solved.

Concluding what was demonstrated until now, we can state that many analog circuits can be realized with regular structures, and that these structures are also good for binary OR-based and EXOR-based circuits and some of their MV-logic generalizations. But how broad are these classes of MV functions? Can they be realized using our FPAA cells? This will be a subject of next few sections.

2.1.3 MVL Orthogonal Expansions

Orthogonal expansions for binary case use EXOR gate, and are generalizations of Davio expansions. For finite multiple-valued logic they are based on Galois Field Addition gate or Modulo Addition, and in general, for arbitrary algebras, they should have at least one linear (group) operation as addition. In continuous logic, they need continuous group (field) addition. Such operations can be created by logarithmic (tangensoid) mapping from standard real field.

But in general they should be based on the algebraic structure of an arbitrary **field**, [7]. In particular, they include S (OR can be replaced with EXOR in Shannon expansion), Positive and Negative Davio (pD and nD, respectively), general orthogonal (binary and MV) [39].

Reverse expansion operations for these expansions are more complicated and will be presented below.

The diagram is created as follows. One level of function f is expanded to an assumed type of the Regular Diagram for a selected variable (or a group of variables in case of orthogonal expansion. Then, the level of the tree is mapped to the assumed type of Regular Diagram. This means combining together some nodes of the tree-like lower part of the diagram. The procedure requires repeating some variables in the diagram, the key point was thus to find good methods of variable and expansion types selections. One approach to the variable order and expansion types selection can be based on generalized partial symmetries for cofactors.

We will illustrate below on examples that the overhead of variable repeating in planary diagrams is not excessive, but further experiments are necessary. We believe that good results will also be obtained for MV logic and non-planar diagrams.

2.2 Types of Expansion Nodes

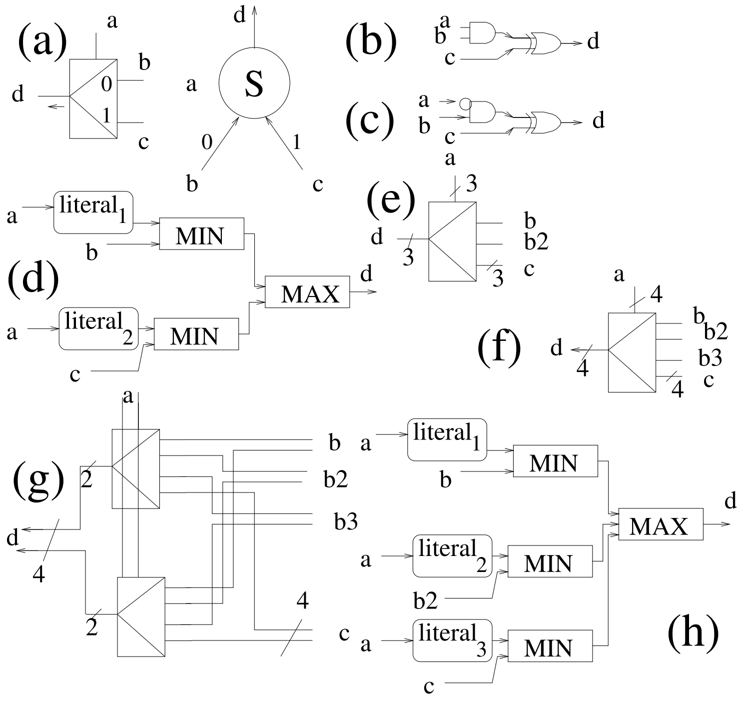


Figure 2.11: *Comparison of Expansion nodes for regular diagrams.*

The procedure of applying forward and reverse expansions in levels is iterated for (repeated) variables until all node functions become variables or constants. Cell with n inputs and m outputs is said to have n x m connectivity pattern. Below, we will present some 3-valued regular diagrams (with 3 inputs and 3 outputs from a node) and 4-valued diagrams (with 4x4 connectivity pattern) for Shannon, Davio, orthogonal, fuzzy and analog expansions.

Fig. 2.11 presents different expansion nodes for various kinds of expansions for binary, multi-valued, and fuzzy functions.

Fig. 2.11a shows two views of a cell for Shannon (S) expansion: a multiplexer, and a general notation of a 2x2 cell in a Diagram that may be realized by this mux (the notation of inputs and outputs is preserved in next examples). When input a is inverted, the so-called Reversed Shannon (S') expansion is executed, which means that the role of inputs b and c is reversed.

Fig. 2.11b shows the positive Davio expansion node (pD), and Fig. 2.11c the negative Davio node (nD). Such nodes are used in Positive-Polarity, Fixed-Polarity, Kronecker and Pseudo-Kronecker Regular Diagrams and their generalizations [38, 39].

Fig. 2.11e presents Shannon node for 3-valued logic,

Fig. 2.11f Shannon node for 4-valued logic, and Fig. 2.11g realization of the 4-valued Shannon node from (f) in binary logic. Two binary signals routed together simulate a 4-valued signal.

It can be observed, that a fundamental condition for existence of reverse expansions and thus, ability of creating regular diagrams is that in the underlying algebraic structure any two literals are disjoint (in binary, this property reduces to $a \cdot \overline{a} = 0$). This leads to binary and multiple-valued (MV) Max-type regular diagrams (we denote MAX-type operations by + and min-type operations by $\cdot$). The principle of operation of binary MAX-type regular diagrams is that any path in a diagram that includes x and $\overline{x}$ cancells. EXOR function is:

$$a \oplus b = a \cdot \overline{b} + \overline{a} \cdot b.$$

Thus,

$$a \oplus a = a\overline{a} + \overline{a}a = 0.$$

This leads to orthogonal type diagrams.

The principle of operation of orthogonal-type diagrams is that any two identical paths to the root in the diagram cancel one another ($x \oplus x = 0$).

2.2.1 Multi-Valued Logic base functions in FPAA

Digital programmable devices (FPGA, EPLD) for classical, two-valued logic have grown in importance over recent years. Although there are many published circuits for multiple-valued logic (e.g. [22, 55, 60]), to the best of the Authors' knowledge, no programmable devices for multi-valued logic have been proposed.

In this report we show how a general-purpose programmable analog array (FPAA) presented in [42], can be used for the implementation of a wide class of MV logic circuits. Minor modifications extending the set of nonlinear operations realized by individual cells of the device can be applied in order to reduce the number of cells necessary to realize basic logic operations. Both the structure of the device and the functionality of individual cells were chosen primarily for dynamic system type of applications, but they prove also to be well suited for the realization of various mentioned logics. The presented examples are linked to the theory of realizing MVL functions in finite fields, introduced in [38]. Some new theoretical results will be also introduced in this report.

Recall that two signals of equal value on the output of any comparator in the control [40, 42] indicate equal input signals. This way the control block can produce control signals being a function of certain conditions of instantaneous input signal values, such as equality of two or more signals, equality of a number of signals to zero or another constant, etc. This feature is also used to realize minimum and maximum follower (MIN and MAX), absolute value (ABS), and other operations. To realize MIN and MAX operations the control block simply detects the smallest (largest) signal and selects this signal on the input of the cell. This is accomplished by comparing the output signal of the selected multiplexer-summer with the input signals. If one (or more) of the input signals are smaller (larger) then the multiplexer-summer output, the control circuit sends appropriate signals to the multiplexer-summer to adjust its weights until the smallest (largest) signal is selected. When realizing the ABS function, the control block changes the sign of input weights if the weighted sum is negative.

Some operations important for MVL applications, performed by the cell, are summarized below. X_i denote input signals, Y denotes the output signal. No distinction is made between local and global signals, since the cell processes them in the same manner.

1. $Y = k(\sum_{w_i \in W_1} w_i X_i)(\sum_{w_j \in W_2} w_j X_j)$
   W_1 and W_2 are independent sets of input weights; k is tuned in the range 0 – 80dB. Complements of the signals (to the maximum possible signal value, MAX) can be calculated.
2. $Y = k(\sum_{i \in W} w_i X_i)$
3. $Y = kX_iX_j$
4. $Y = k{X_i}^2$
5. $Y = kMIN(X_1, ..., X_n)$
   The control block "watches" input signals and selects the smallest one.
6. $Y = kMAX(X_1, ..., X_n)$
7. $Y = kY_{1-6}\frac{1}{s+\alpha}$
   Y_{1-6} is any of the functions presented in rows $1-6$ above; $\alpha \geq 0$.
8. $Y = asign(Y_{1-6})$ $a = b, k = \infty$.
9. $Y = bU(Y_{1-6})$ where U denotes the step function. $a = 0$, $b = MAX$, $k = \infty$.
10. $Y = k \mid Y_{1-6} \mid$
11. Identity.

As seen in points 1-11 above, the cell performs summing of input signals selected by the control circuitry, multiplication of two signals (squaring of one signal), or multiplication of two independently derived weighted sums of input signals. Further processing includes lossless or lossy integration, and clipping.

The functions shown above can be all used in the implementation of continuous-time dynamic systems [21], and multi-valued, fuzzy, and continuous (such as Lukasiewicz) logic circuits. The architecture of the device is motivated by the desire to enable circuit realizations with minimal signal delays. A number of examples, including an elliptic eighth-order ladder filter [54], a rank filter cell [37], circuits for tracking the product of two matrices, a solution of a system of linear equations [21], and a solution of a linear programming problem by the method of steepest descent [21], were designed and analyzed (some with Spice).

The cell is being implemented in a bipolar transistor array process. An implementation in BiCMOS and analog CMOS (to increase density) is considered.

Chapter 3

New Canonical Forms for Quaternary logic

3.1 Realization of Galois field(2^2) operations in 4-valued logic

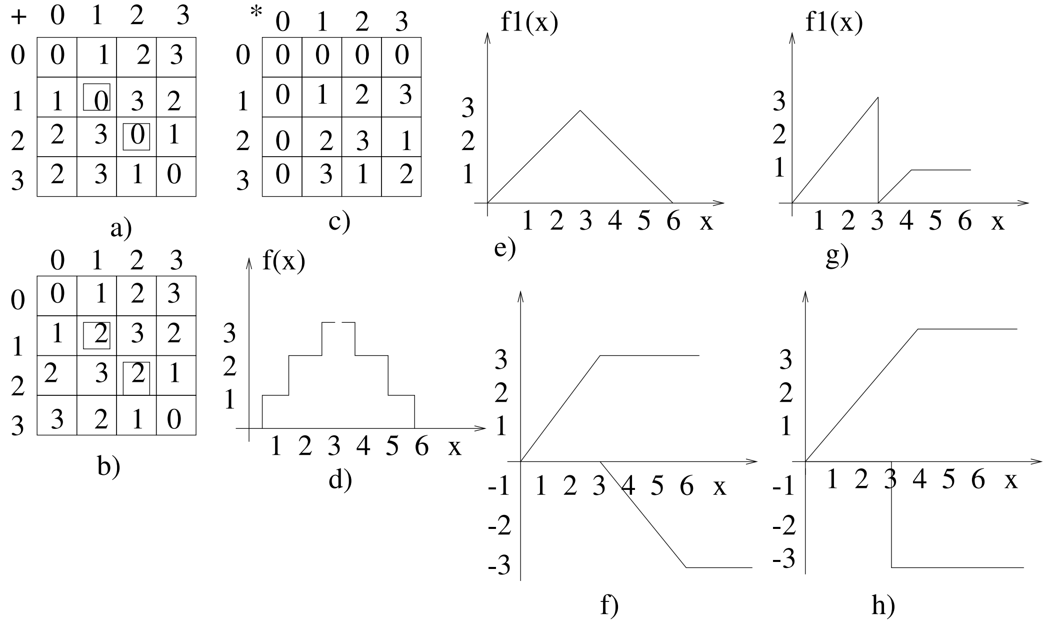


Figure 3.1: *$GF(2^2)$ operations.*

Figures 3.1a and 3.1b show the tables for addition and multiplication in Galois field of four elements. Each of these operations can be realized by a single cell of FPAA, assuming that only two of the cell's inputs are used at a time.

Addition can be realized as

$$a \oplus b \; = \; f(a+b)$$

for $a \neq b$ (Figures 3.1c and 3.1d), and $a \oplus b = 0$ otherwise. The condition $a = b$ can be detected by the control block. This requires programming the weights of one of the input multiplexers-summers to calculate the difference $a - b$ of the input signals, selecting constant 0 for comparison in the control block, and controlling the weights of the other input multiplexer to set them to zero if $a = b$ was detected. Instead of function $f(x)$ (Figure 3.1d) a smooth function $f_1(x)$ (Figure 3.1e) can be used. This function can be realized by adding two characteristics of the saturation blocks shown in Figure 3.1f. If the function of the form shown in Figure 3.1d is required, it can be realized by providing more nonlinear blocks in the cell.

Multiplication $a \odot b$ in the field (Figure 3.1b) can be realized as $a \odot b = ((a + b - 2)_{mod3}) + 1$ for $a \neq 0$ and $b \neq 0$, and $a \odot b = 0$ otherwise. The two conditions for a and b can be tested independently by the comparators in the control block, and upon at least one of them being true the input weights of the multiplexer-summer would be turned down to 0. Mod 3 operation can be realized as shown in Figures 3.1g and 3.1h. The control block performs the necessary logic operations.

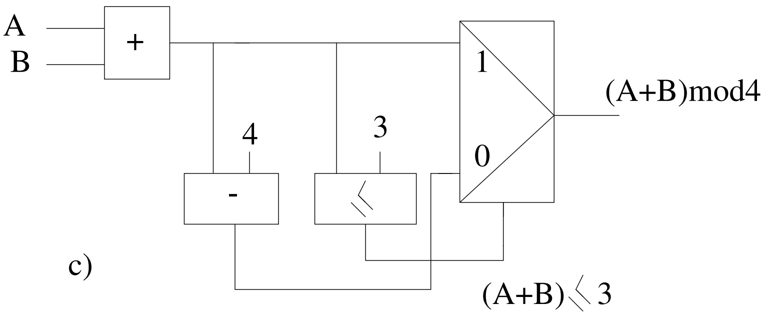
The realizations of GF(2^2) operations proposed above are similar to the realizations presented in [60].

+	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

a)

$$(A + B) \bmod 4 = \begin{cases} A + B \text{ if } A+B \leqslant 3 \\ A+B-4 \text{ otherwise} \end{cases}$$

b)



c)

Figure 3.2: *MODULO-SUM operation.*

Schematic realization of Modulo-Sum operation is shown in Figure 3.2. Observe that basic blocks such as the "comparator of order" or "analog multiplexer" are used. Thus, analog logic blocks are very similar to binary logic blocks.

3.2 Orthogonal Expansion Structures for Quaternary Logic.

Having defined the addition and multiplication in $GF(2^2)$, we can apply the combinational functions synthesis method based on orthogonal functions presented in [38, 39].

Figure 3.3a shows a block diagram of a structure realizing a function of input variables X_1, X_2, ..., X_m. Each column realizes one orthogonal function over $GF(2^2)$. Multiplied by a constant from $GF(2^2)$, this function is added to the other orthogonal functions. All operations are in $GF(2^2)$. Figure 3.3b shows an example of realization of one of the functions f_i. Since each cell can realize the identity operation (as shown before), it is possible to omit certain input variables X_i, X_3 in this example. More than one column of cells can be used for the realization of each f_i if necessary. Also, it may be convenient to make certain input variables available on more than one horizontal line.

An alternative approach, based on providing literals on horizontal lines, or some functions of single variables which are convenient for the creation of literals, is also possible. In one such approach, the powers (i.e. multiple products in $GF(2^2)$) would be used to create polynomial expansions of MVL functions.

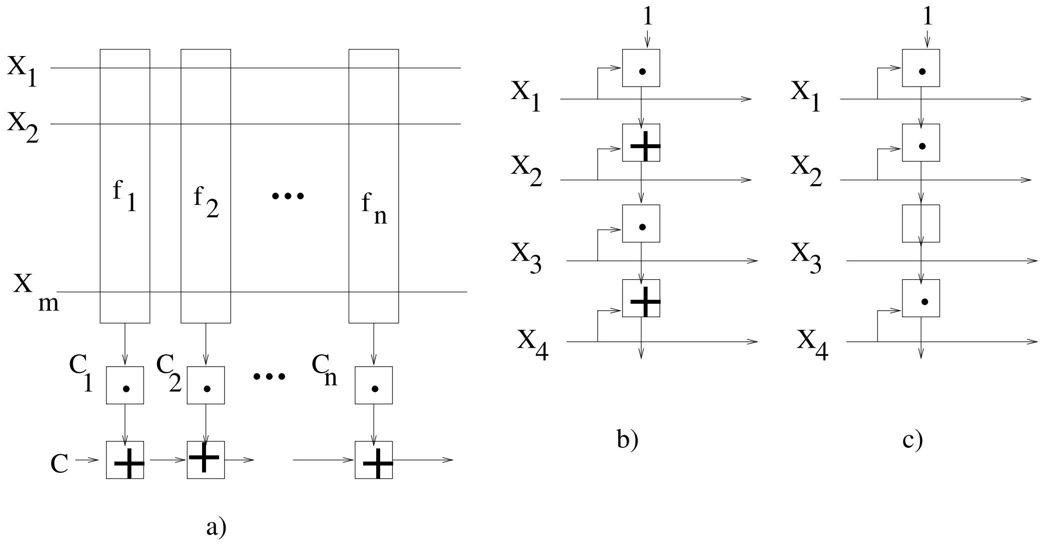


Figure 3.3: *Example of Generalized PLAs: Orthogonal Expansions in regular PLA-like structures: (a) General structure, (b) orthogonal column with addition and multiplication operators, (c) orthogonal column with multiplication (MIN, Galois multiplication, AND, etc) operators*

3.3 Post logic

The same structures shown in Figure 3.3 can be used for the implementation of Post logic. Each cell can realize MIN and MAX operations, and literals of the form shown in Figure 3.3c. Each function f_i is realized as in Figure 3.3b, except that the cells realize MIN or IDENTITY operation. Instead of summing over $GF(2^2)$, MAX operation is used.

3.4 Other applications of Generalized PLA-type structures.

The structure of Figure 3.3a can be used for realization of combinational functions with other methods. Such realizations, unlike the ones based on the orthogonal expansions, may not be unique in the presented structure, however due to the availability of addition, multiplication (in the conventional sense), and nonlinear operations on signals, some combinational functions may have very efficient implementations. Also, the topology of MVL circuits mapped into the FPAA does not have to be constrained to the form shown in Figure 3.3. Global vertical and diagonal signal lines can be used, if necessary, to achieve greater flexibility of the circuits' topologies.

As we will demonstrate in next sections, the most general are the orthogonal expansions. They all start from the well-known Shannon (Boole) expansion, from which Davio and next all orthogonal expansions can be derived.

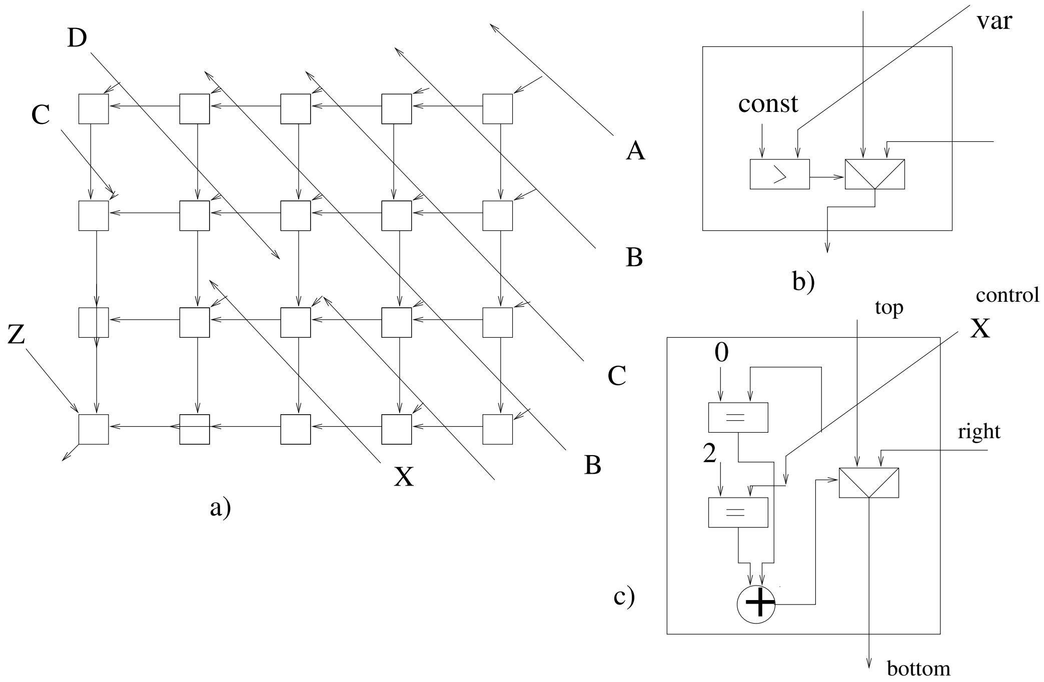


Figure 3.4: *Regular array to realize symmetric and symmetrizable analog, binary and multivalued circuits: every control variables cuts the set of values (finite or not) into two subspaces that generalize the concepts of cofactor in binary logic. (a) The general view of the array with repeated and folded variables, some cells do not obtain variable, they are fed with constants to select the direction of transfer, (b) the example of a cell for continuous logic, (c) the example of a cell that executes the operation of quaternary logic:* **if** $X \in \{0,2\}$ **then** *top* **else** *right*

For instance, Figure 3.4 shows a structure [42] for implementations based on generalized Shannon expansion of MVL and piece-wise continuous functions that generalizes the binary scheme known from Kohavi's book. Remember that the structure with **const** set to 0 and binary values is the same as in Kohavi and can be treated as a decision diagram for arbitrary Boolean function, not only symmetric. This requires repeated variables as in Akers Array, but if good choice is done, and especially for incomplete functions, there is no need to repeat exponentially as in Akers. Some input variables need to be connected to more than one diagonal line to allow realization of arbitrary functions. More general forms of the same kind are possible, based on other operators than the operator "$>$" used for separation, for instance even vs. odd parity, based on matrix orthogonality, which is a generalization of the approach presented in [38, 39] for two-valued functions. The expansion matrix should be orthogonal (non-singular) with respect to the addition operation, for instance, in case of EXOR logic, this operation is EXOR. In case of arbitrary Galois Logic, this operation is Galois addition. In case of arbitrary field, this operation is field addition. Some of these generalizations will be presented below.

Finally, the integrator block can be used as a memory element, enabling realization of sequential circuits. Since each cell can realize identity function, and global connections are available, larger irregular structures, composed of combinational and sequential parts can be built in the presented structure.

In the future add here discussion of four-valued expansions with {{0,3},{1,2}}, {{0,2},{1,3}}, etc.

3.5 Forward and Reverse Expansions of MAX type

The investigated by us types of expansions can be characterized as of **maximum-type**, or of **Orthogonal type.** Maximum-type expansions use the MAX gate (in binary logic - OR), and **disjoint** literals or subfunctions for cofactors. They include binary Shannon (S) and Sum-of-Products (SOP) expansions and their multiple-valued logic generalizations, such as in Post logic.

In this section, we will present the MAX-type expansions: **Shannon, SOP**, and **Post (MV Shannon)**.

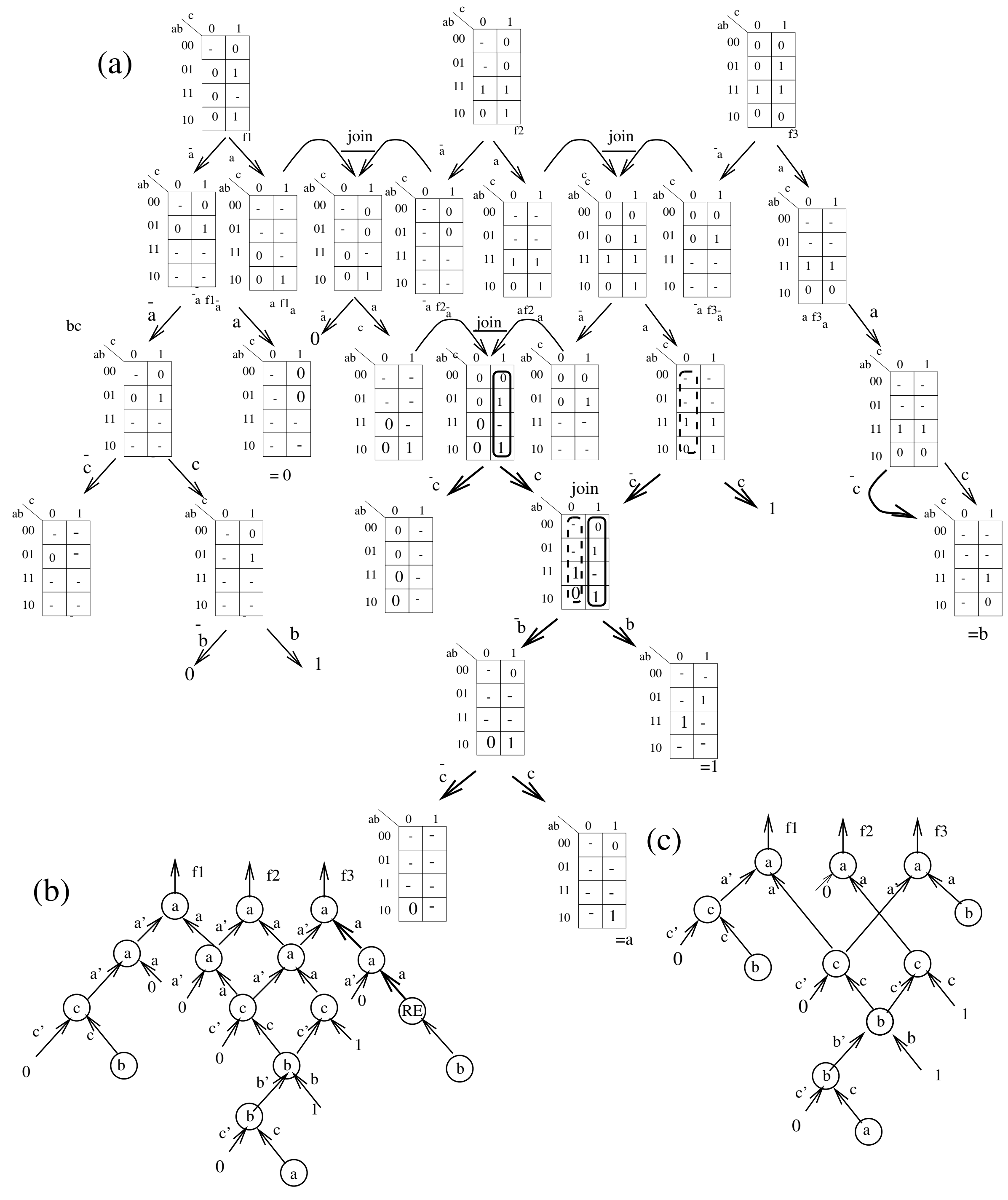


Figure 3.5: *Shannon Expansions and Inverse Expansions for Incomplete multi-output binary function*

We assume that each binary function f is represented by a pair [ON(f),OFF(f)]. Thus all cofactors f_a for the product of literals a, are pairs: f_a = [ON(f_a),OFF(f_a)]. Fig. 3.5a explains the principle of creating a Shannon Reular Diagram based on ordered Shannon expansions for a multi-output function (functions f,g). Direction of arrows shows the expansion flow.

Observe that every cofactor f_a of the product a of an (in)complete function f can be interpreted as intersecting f with a and replacing all K-map cells outside product a with don't cares. A standard cofactor f_x where x is a variable does not depend on this variable. In our interpretation, though, f_x is still a function of all variables including x, but as a result of cofactoring the variable x becomes vacuous. We will call this a **vacuous cofactor**, and denote by **v-cofactor**.

Thus, for any two disjoint products a_1 and a_2, the v-cofactors f_{a_1} and g_{a_2} are disjoint (observe that standard cofactors are in general not disjoint), Therefore functions f_{a_1} and g_{a_2} are in an incomplete tautology relation, and functions f and g are not changed when f_{a_1} and g_{a_2} are combined (OR-ed) to create a new function: $a_1 f_{a_1} + \overline{a_2} g_{a_2}$, as in Fig. 3.5a (where: $a_1 = a_2 = a$, and $\bar{a}$ is denoted as a'). This is what we call the Reverse Shannon Expansion. (Observe that for every expansion one can create a reverse expansion).

This way, the entire layout is created level-by-level, only three levels shown in Fig. 3.5a. Observe that functions in the regular layout nodes become more and more unspecified when variables in levels are repeated, and ultimately nodes become constants, which terminates the layout generation process. This way, because every variable cuts a Kmap into two disjoint parts, arbitrary two functions f and g can always be expanded together to a Shannon diagram, with OR-ing as a reverse expansion operation, provided that the same variable x_i is used in the level, and all expansions use negated literal $\overline{x_i}$ in the left, and positive literal x_i of the variable in the right.

Observe, that for quaternary Shannon expansions and Inverse Expansions the method is exactly the same, the only difference is that every variable has 4 cofactors, so the map is split into 4 and not 2 areas. If we want to use 2x2 array for quaternary logic, then we have to combine any subset of cofactors to one branch of the distributor/collector, and the remaining cofactors to the second branch. For instance, the groupings can be: $X^{\{0,1\}}$ to the left and $X^{\{2,3\}}$ to the right, or $X^{\{0,2\}}$ to the left and $X^{\{1,3\}}$ to the right, or $X^{\{0,2,3\}}$ to the left and $X^{\{1\}}$ to the right, etc.

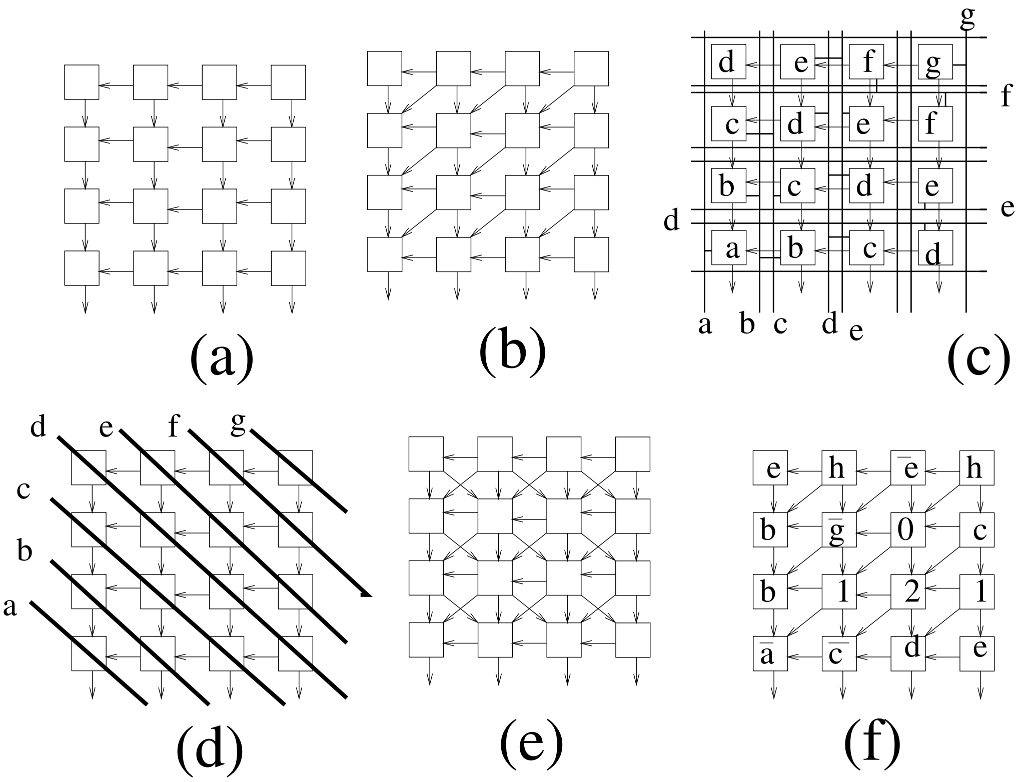


Figure 3.6: *Examples of regular diagrams: (a) standard 2x2 array with only local connections, (b) standard 3x3 array with only local connections, (c) 2x2 array with two vertical and two horizontal buses per cell and the programming of its buses,*

As seen in Fig. 3.5, arbitrary functions of the same arguments are cut in half by expansion of each variable and new functions in levels are created by rearranging the cofactors in reverse expansions. This process can lead to a slight increase of the number of nodes in comparison with a shared OBDD of these functions. But a regular structure is created, thus simplifying layout and making delays predictable. In case when the products a_1 and a_2 are **not** disjoint, the v-cofactors f_{a_1} and g_{a_2} can, in some cases, still form an incomplete tautology of functions. When these two cofactors satisfy a tautology relation, then functions f_{a_1} and g_{a_2} can be combined (OR-ed) without changing functions f and g. Obviously, the same method works for arbitrary number of output functions.

Observe however, that our general scheme allows to create more general expansions/inverse expansions and structures, all derived starting from the basic pattern of Figure 3.6a. By adding oblique local connection, the 2x2 array is converted to a 3x3 array from Figure 3.6b. As we will see, there are many 3x3 patterns, and for many of them we found interesting applications. Figure 3.6c) presents an example that for small arrays we do not need oblique buses, because cells can obtain control variables from vertical and horizontal buses. Here, two vertical and two horizontal buses are assumed, the same as in Concurrent Logic architecture. Figure 3.6d) presents an example of our basic structure with oblique buses, which in this case is programmed to a (generalized) symmetric function, because there is no repetition of variables. This array with repeated variables; a, b, b, c, b, b, a would depict the Universal Akers Array. Figure 3.6e) presents an example of basic structure with two oblique local connections, thus creating a pattern 4x4. Such cell can either execute Quaternary Shannon Expansions, or it executes standard Shannon expansion to two branches, and next selects any two of its successors to map the branches to them and execute corresponding inverse expansions. Finally, Figure 3.6f) presents an example of a 3x3 structure in which each cell is programmed to a different control variable or a constant. Such array executes ternary expansions. For larger array it may be a problem with routing all control input signals.

Concluding this subsection, it should be obvious by analogy to standard diagrams, that with given order of (possibly repeated) variables these expansions are canonical if the same variable and expansion type is in every level. (Analogy to Functional Kronecker diagrams [67]). By assuming special additional rules, also more general diagrams with mixed (Pseudo-Kronecker type) expansions could be made canonical, the same as the Pseudo-Kronecker decision diagrams, as presented by paper with Li Fei Wu. [66].

3.6 SOP Expansions

In binary SOP expansions a branching from node f is for any subset of literals l_j that their union covers the node function f. The SOP expansion is:

$$f \;=\; l_j f_{l_j} \;+\; l_r f_{l_r} \ldots. \;+\; l_s f_{l_s}.$$

The method to create ordered Shannon diagrams presented above can be expanded to free (non-ordered) Shannon Diagrams and SOP Diagrams, and their corresponding layouts.

Any two nodes from the expansion that form an incomplete tautology can be combined as shown above. S and SOP expansion types can be mixed in levels, thus creating *"pseudo"* type of diagrams, by analogy to pseudo-Kronecker diagrams.

3.7 Layout Geometries

3.7.1 The Case of 4 neighbors

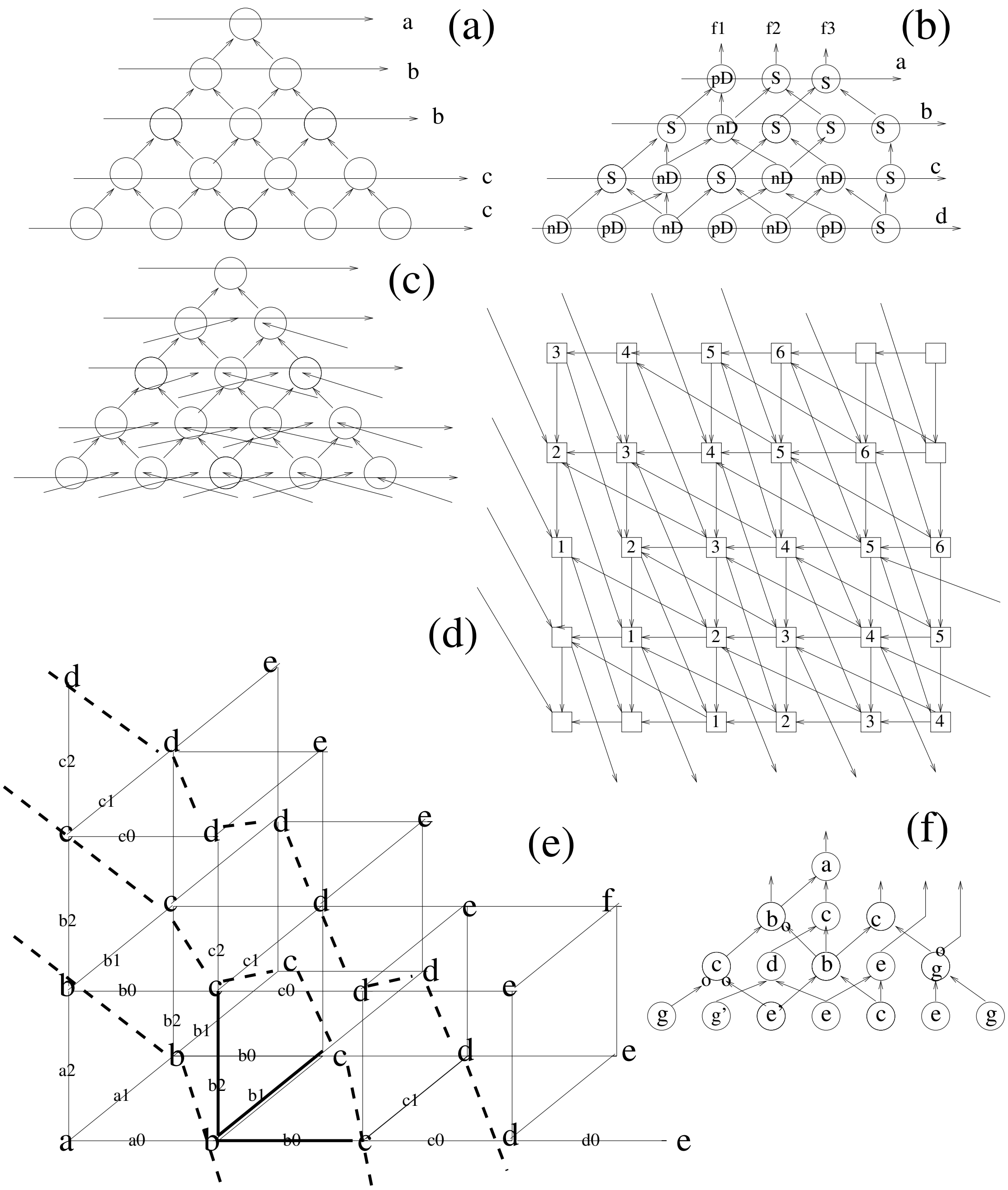


Figure 3.7: *More examples of regular diagrams.*

In case of 4 neighbors, 2x2 cells, Fig. 3.7a,c,d, the diagram is planar and it is based on a rectangular grid. Each cell has two inputs (from N and E) and two outputs (to S and W) [1]. Our structure allows for Positive and Negative Davio expansions, negated variables and constants as control variables of the nodes, nodes controlled not by variables but by functions, and inverted edges between nodes. We created regular diagram counterparts of Kronecker, Pseudo-Kronecker and Free Kronecker Diagrams.

We will show in this report that every function that is not symmetric can be symmetricized by repeating variables in the diagram layers, and the selection of the next variable is done using the repeated variable maps.

With respect to possible technological realization, the condition of only a single control variable in a level is no longer required, and all three types of buses (vertical, horizontal and diagonal) are used to lead any variable to the circuit's levels.

Similarly we do not always require the existence of the diagonal buses, as well as the condition of having only constant values on the envelope of the circuit, or having the outputs only on the envelope. Now, arbitrary variables can occur on the envelope, and outputs can be taken from the middle by use of buses (this is an approach from commercial Fine Grain FPGAs [5]). Moreover, pairs or triplets of binary control variables can be used in nodes for arbitrary orthogonal expansions [39]. Or equivalently, multivalued controls are used. In short, for a 4-neighbor diagram geometry, any canonical form of Reed-Muller logic and its orthogonal generalizations can be realized (recall that the Reed-Muller logic is a special case of Galois Field logic, the GF(2)). Also, any MV logic (for instance in GF(4), can also be realized in the 4-neighbor diagram, but this would often require many repetitions of variables. Continuous and fuzzy functions can also be realized: such as the arbitrary piece-wise continuous function with a generalized Shannon expansion [42].

3.7.2 The Case of 6 neighbors

In **case of 6 neighbors**, 3x3 cells, Fig. 3.7b,f and Fig. 2.11b,f,e, the rectangular grid is enhanced with one diagonal connection, thus every cell has three inputs (from N, NE and E) and three outputs (to W, SW and S). This allows a realization of the generalized 3-valued diagrams (for binary EXOR logic), as well as realizations of arbitrary expansion-based Post logic or GF(3) logic functions [39]. Finally, any binary, or MV logic can be mapped as in the case of the 4-neighbor diagram, but now larger full trees are mappable to subsets of diagrams.

3.7.3 The Case of 8 neighbors

In **case of 8 neighbors**, 4x4 cells, Fig. 3.7e, and Fig. 2.11c,d, the rectangular grid is enhanced with one more diagonal connection, so that every cell has four inputs (from N,NE,NW, and E), and four outputs (to S, SW, SE, and W). This allows a realization of the generalized 4-valued diagrams (for GF(4)), as well as arbitrary expansion-based Post or GF(k), $k < 4$ functions. Again, any binary or MV logic can be mapped, and more efficiently so. In other geometry variant, the neighborhoods are: NWW, NW, NE, NEE for inputs, SWW,SW SE, and SEE for outputs.

Good results were found for new 3-valued diagrams and Galois(4) expansions for multi-output incompletely specified functions. Regular Kronecker Decision Diagrams with negated edges are based on three orthogonal expansions (Shannon, Positive Davio, Negative Davio).

Many other new families of functional decision diagrams were created, including: Pseudo Kronecker Regular Diagrams, and Free Kronecker Regular KDDs. The Boolean 3-valued Regular DDs introduce nodes with three edges and requires AND, OR and EXOR gates for expansion circuit realizations.

Galois GF(3) and GF(4) diagrams require three and four neighbors on inputs, respectively.

The families of diagrams that we introduced are the counterparts and generalizations of several diagrams known from the literature (BDDs, FDDs, Kronecker diagrams). Due to this property, our diagrams can provide a more compact representation of functions than either of the standard decision diagrams, because they do not require any placement or routing. Placement and routing come as a side-effect of logic synthesis. Our methods are very efficient especially for strongly unspecified functions,

the more unspecified the function, the better the results. Still, after our logic/layout design some layout compression (compaction) is necessary, for instance for sparse functions such as
$abcde + \overline{a}\,\overline{b}\,\overline{c}\,\overline{d}\,\overline{e}$.

3.8 Further Generalizations.

It is easy to generalize the binary Shannon expansions used in Fig. 2.9a,b to 3-valued and 4-valued Shannon expansions. The new diagrams would require 3 inputs and 3 outputs from a node, and 4 inputs and 4 outputs from a node, respectively. Next 3- and 4- valued counterparts of S' can be created, and respective 3-valued and 4-valued diagrams can be formed by expanding and reverse expansion formulas. This way, Post-type and Galois-type regular diagrams are created in an uniform way, with only difference of using various expansion and reverse expansion rules. However, the two kinds of principles, of creating the expansion and of the reverse expansion rules, remain the same: disjoint literals for MAX-type Regular Diagrams, and $a + (-a) = 0$ term cancelling for Orthogonal Regular Diagrams (which generalizes the rule $a \oplus a = 0$ of Galois Field (2). The regular diagrams have advantages especially for (nearly) symmetric functions and strongly unspecified functions that can be completed to symmetric functions.

By a **regular layout** we understand a layout of indentical cells that connect by abutting. By a **complete layout structure** we understand connection pattern between cells, that allows to realize every symmetric function without repeating variables. It can be proved that in a 2x2 diagram **every** binary symmetric function can be realized without variable repetitions, and with connections between cells having the same length. Thus, **regular diagram layout for binary logic is regular and complete**. In contrast to binary functions, symmetric 3-valued functions cannot be realized in regular 2-dimensional 3x3 regular diagrams. Although we created 3x3 diagrams that can realize every symmetric 3-valued function without variable repetitions, it is not possible to find regular layouts for realizing them. Thus the cells distances in subsequent levels grow. Hopefully, it is not a practical problem for small functions realized in MV logic, but the beautiful simplicity of binary realizations does not longer exist. Thus, if mapped to a 2-dimensional space, the 3-valued diagrams are either regular and not complete, or complete but not regular. It is still possible to obtain regular and complete 3x3 regular diagrams assuming layout of cells in a three-dimensional space. But it is not possible to create regular layout for 4x4 diagrams, because our Universe is 3-dimensional. Although these considerations are theoretically interesting, the two types of 3-valued and 4-valued diagrams that we developed; the regular and incomplete, and the irregular and complete, are very useful in practice. We do not know of any paper that would create these ideas of two-dimensional and three-dimensional regular structures and that would practically show how many circuits and functions can be mapped to them. It is an amazing result, and filters and Boolean functions are just a few examples given in this report. More work, however, is needed in this area.

As shown in Figure 2.11, pairs of binary variables correspond to 4-valued variables. Although here we discuss orthogonal regular diagrams for only two variables in each **variable block**, all concepts and algorithms can be expanded to variable blocks of arbitrary size. Fig. 4 shows an example of a circuit obtained by substituting nodes of a 4-valued orthogonal regular diagram with their circuits (this particular circuit is even 3x3 realizable, a dummy node is used in level 2).

The Orthogonal Regular Diagrams for pairs of variables are created similarly to regular diagrams for single variables in section ??. Nodes are now for pairs of variables, and orthogonal expansions of Orthogonal logic [38, 39] are used. (Such expansions have invertible (non-singular) matrix). Every node has at most 4 inputs. Instead selecting among only three expansions, S, pD and nD, the choice in every level of nodes is among all 840 orthogonal expansions in exact algorithm (this is the maximum number of orthogonal expansions for a pair of variables), or some subset of them in approximate algorithms.

The same type of expansion is selected in Kronecker type regular diagrams, or various expansions are selected in nodes of Pseudo-Kronecker type diagrams. The reverse expansions are based on the same principles as in sect. 2. The diagrams for all single outputs of a multi-output function are created together, level-by-level from their root nodes (outputs). In every level, the possible expansions are evaluated based on the complexity of the next level (look-ahead strategy). The best expansion found by the Polarity Selecting Algorithm for a level is next applied to all nodes (Kronecker types) from the level of the multi-output diagram. In Pseudo type of diagrams, the expansion decision for each node is done separately.

Now we will show some further generalizations of these ideas, all created as more practical and realistic approaches to geometry-driven logic synthesis by relaxing many constraints such as planarity, canonicity, number of neighbors, and total regularity, that existed in the previously defined regular diagrams. Our ultimate goal is the synthesis in cellular environment, as defined by Fine Grain FPGAs, sea-of-gates, standard FPGAs such as Xilinx.

Based on analysis of realizations of difficult functions and new technologies, we generalize these concepts in the following ways:

1. Instead of assuming only **canonical** expansions or orthogonal expansions, we consider also **non-canonical** expansions such as the Boolean 3-valued expansions.

2. We consider **all** canonical and non-canonical Kronecker, Pseudo-Kronecker, Mixed, 3-Valued, and other Decision Diagram **concepts** that are used in Reed-Muller and standard logic, and we **generalize these concepts.**

3. We allow more **powerful neighborhood geometries.** Instead of having only two inputs and two outputs from every node (we call them 2x2 Regular Diagrams), we consider also regular regular diagrams in which node has 3 inputs and 3 outputs, diagrams with 4 inputs and 4 outputs from a node, and diagrams with 8 inputs and 8 outputs from a node. The 3x3 Regular Diagrams are for 3-valued logic, pairs of binary variables (non-canonic), or for Boolean 3-valued Decision Diagrams. The 4x4 Regular Diagrams are for 4-valued logic, or pairs of binary control variables. They are for instance used for Diagrams realizations of GKTs, PGKTs, etc. The 8x8 Diagrams are for 8-valued logic, or triplets of binary control variables. Similarly we can create higher-neighborhood geometries, but we believe it is more practical to keep the number of neighbors small. In essence, the 2x2, 3x3, 4x4 and 8x8 neighborhoods are possible.

4. We allow to mix control variables in diagonal buses. This permits to realize Free and Mixed diagrams.

5. We allow to apply reverse expansion of **any non-isomorphic nodes**, not only some kind of neighbors (like nodes in 3 *3 or 4 * 4, or other neighborhood). This is a very important and powerful property, because it allows to create arrays of arbitrary shape if no space is available, we can accomodate shape and size by combining more nodes together.

Most of the diagrams are **Regular Diagrams**. They lead to regular layouts. The only exceptions are: the previous to the last row, that corresponds to regular 3x3 regular diagrams created with nodes for pairs of control variables with non-canonical expansions in nodes, and the last row, with generalized, non-regular diagrams. The **reverse expansion operation** can be done on any two nodes, not only on neighbors. It was executed on neighbors in previously discussed diagrams only for the regularity of layout. It can be, however, observed, that for some technologies, such as Xilinx with many connections possible in channels, the total regularity based on neighbor-only connections is not necessary and can be relaxed to decrease the CLB count. This leads to the last category of representations, **Generalized Diagrams**, in which arbitrary nodes can be combined. So, the layout is no longer regular, but if proper nodes are selected for combining, a well-routable placement can be mapped from the Generalized Regular Diagram.

Most of the representation, with the exception of the last three rows, use **canonical expansions. Non-canonical expansions** give a freedom of creating, for a given subfunction in the actually expanded node, such an expansion, that minimizes well this function locally.

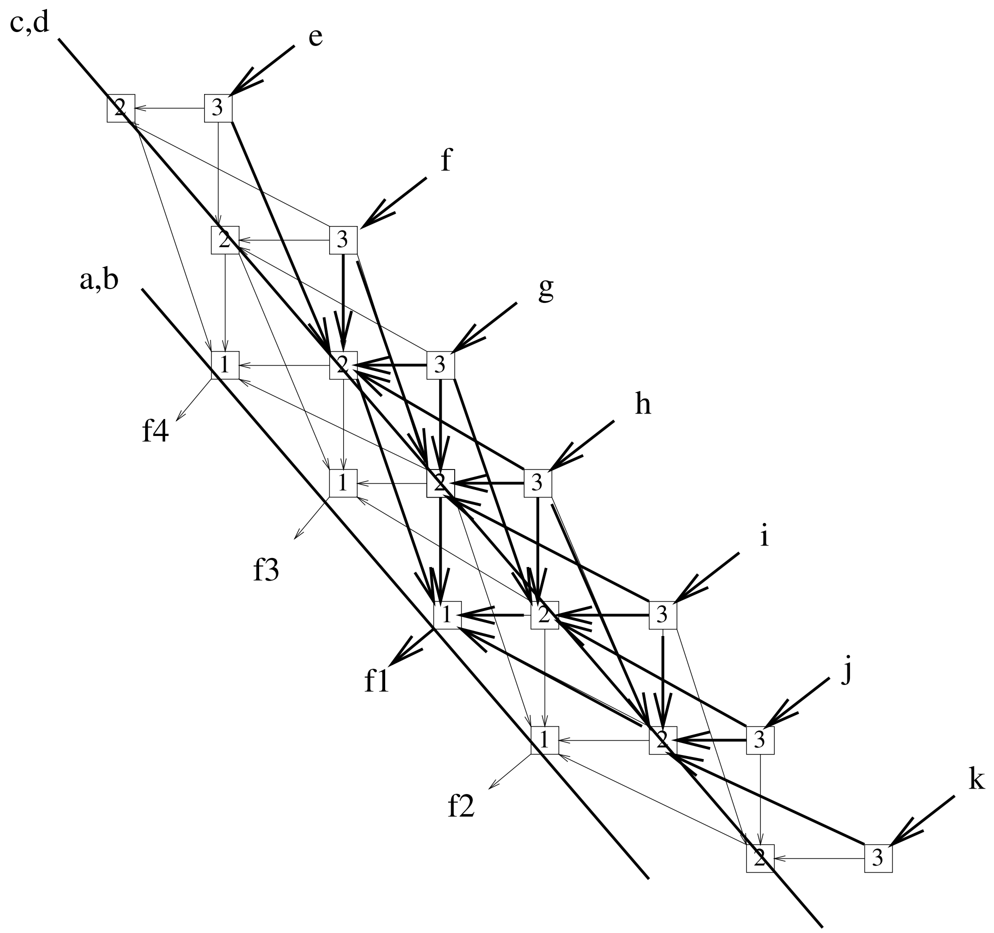


Figure 3.8: *An Example of Regular 4x4 Diagram that realizes a 4-output, 11-input function. Every small rectangle is a Expansion Module with corresponding two control variabls in diagonal buses, and four data inputs. Numbers in squares correspond to levels of the diagram.*

The choice of the appropriate regular diagram and array type for a given Boolean function remains a difficult problem to be solved and at the moment we dispose just examples and heuristics, but we do not know any general solution to it. In theory, one would need just the most powerful layout array type, and assume that the design algorithm will select the best expansions and variable ordering in any case (the same as Kronecker Diagram versus BDD or FDD). But creating a good heuristic algorithm for the most general arrays is more difficult than to create such an algorithm for a restricted type of array. Perhaps, some simpler regular arrays are also better for layout, like assuming only Shannon Expansions (multiplexers) allows to design the function from mostly pass-transistors and very regular small-grid layout.

Calculation of data input functions to diagram nodes for **any** type of expansions and **any** diagram neighborhoods is performed by the same technique of solving logic equations for a given structure, as one used for Orthogonal logic. This technique is very general and can be adopted to many non-binary logics. However, in contrast to the Orthogonal logic, where the equations have always one solution resulting from non-singularity of the matrix M, the structural equations, in general case, can have one, many, or no solutions. (Also, they are not only equations over Galois Field.) When there are many solutions, the one evaluated as best is taken. When there are no solutions, the backtrack to another structure, another expansions, or another blocks of input variables should be executed. Selection of the order of (usually repeated) variables is done using the concept of the best separation of most different-value minterms, using repeated variable maps and more general symmetries.

The problems of variable ordering and variable partitioning, known for long in logic synthesis to be tough ones, here become even more important, and at the same time more difficult, because the variables must be repeated. Hopefully, it was found that in contrast to the worst-case randomly generated functions, for real-life benchmark functions only few repetitions of variables are enough. It is especially easy to symmetricize weakly specified functions.

Figure 3.9 explains the symmetricization process using a Kmap. Based on the function, we create the conflict graph and select the variable that has most conflict with other variables. In the map there is the conflict between cell 011 and and 110, thus a conflict in "a" and in "c". There is also the conflict between cell 011 and and 101, thus a conflict in "a" and in "b". Thus "a" is the conflict variable and is repeated in a new Kmap. Then we repeat it again and again, until the function is symmetric, it means there are no conflicts between its variables, which means, in every set of cells with the same symmetry index (Kohavi) the values are all 0's and dc's or all 1'a and dc's.

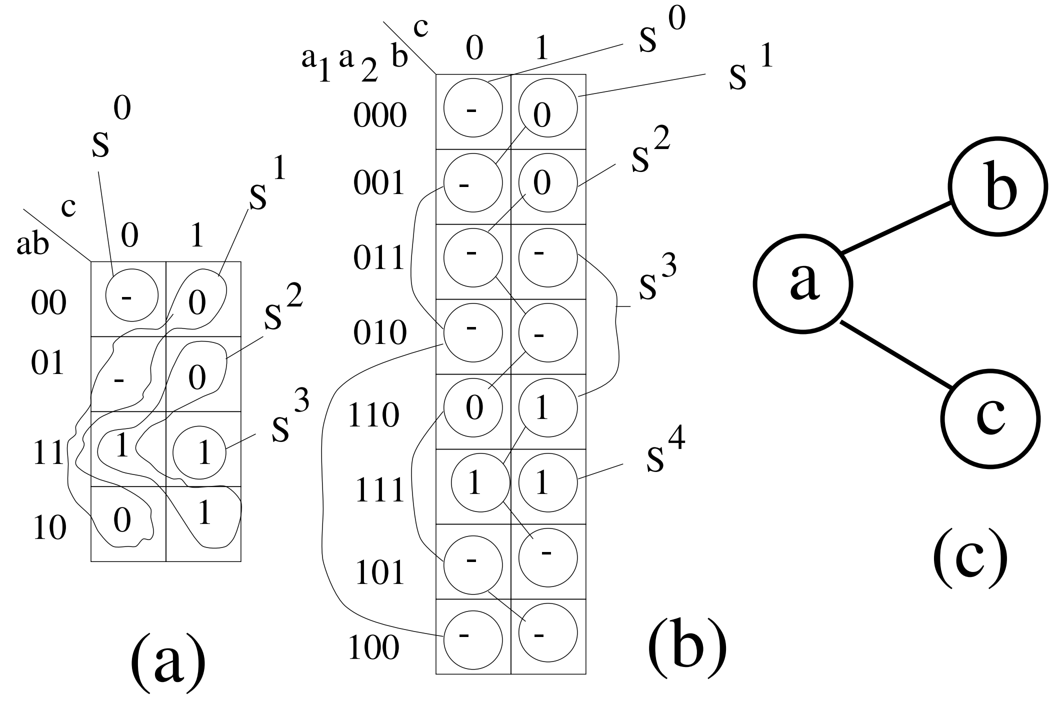


Figure 3.9: *Explanation of Symmetricization algorithm. (a) Kmap of a non-symmetric function, (b) variables are repeated to make function symmetric, (c) the graph whose edges correspond to conflict of variables - the most conflicting variable is selected to be repeated*

3.8.1 Applications of Regular Diagrams to realize Fuzzy Functions

Because in standard fuzzy logic $a \cdot \overline{a} \neq 0$, and $a \oplus a = a\overline{a} + \overline{a}a \neq 0$, both the MAX-type and orthogonal methods would not work. However, let us observe that one can define a **negation-less fuzzy logic**, which we call a **Disjoint Fuzzy Logic (DFL)**, in which all fuzzy logic axioms besides those related to negation are satisfied, and negation is simulated by using special type of literals. In

DFL logic, any two literals $literal_i$, $literal_j$ can have arbitrary shapes, but must be disjoint; for any value of $x \in [0,1]$ $literal_i(x) \cdot literal_j(x) = 0$.

Fig. 2.11d presents the binary expansion node for binary DFL, and Fig. 2.11h the 3-valued expansion node for 3-valued DFL. The literals $literal_1$, $literal_2$, $literal_3$ are all mutually disjoint. DFL expansions are realized in 3-valued fuzzy regular diagrams, similar to MV 3-valued diagrams. Because of disjoint literals, reverse expansion can be always performed. Observe, that k-valued Post logic is a special case of DFL with k literals, (so the 3-valued Shannon expansion is a special case of the 3-valued DFL expansion).

3.9 Conclusions

In this report, the first in series, we have presented a **very general concept** and we showed some of its applications. We believe, however, that it can be much more developed and illustrated in future, based on the broad fundamentals **already given here** and explained in a sufficient detail. The presented model of creating architectures is suitable for the realization of a wide class of calculations; analog, digital (binary and mv), and mixed.

This specific architecture model results from the general premise to **use local signal interconnections whenever possible**, and global signal interconnections only regularly (like in input busses) and only when absolutely necessary. It results also from attempts at finding general tesselation structures for two- and thre-dimensional space in which such structures can be practically realized.

The design of the individual cells, and specific details of the architecture, were determined upon the consideration of the perceived applications of the device, i.e. fast dynamic systems and fuzzy and multi-valued logic circuits. The device is capable of time-continuous and discrete-time operation. Of course, binary signals are a special case of both multi-valued, and fuzzy logic signals, so everything true for a wider case remains still true for binary.

Specifically, the general purpose programmable analog array can be used for the implementation of various logic circuits. Although the original FPAA was designed for dynamic systems and general analog electronics, its structure and functionality is generally well suited for many applications in various areas. For instance, only minor modifications of the FPAA introduced in [43], namely the inclusion of two or more nonlinear blocks, were necessary to enable convenient realization of several multi-valued logic examples.

This report demonstrated that the realizations based on orthogonal expansions as well as more general ones, based on sets of **not necessarily orthogonal functions**, lead to regular circuit structures which can be easily mapped to our arrays.

The proposed particular variant, realized for 4-valued logic, is an excellent tool for fast prototyping of circuits in various logic systems. It will provide the researchers in the field with an opportunity to experiment with hardware realizations of various logic circuits without the necessity to design and fabricate them. Analog, MVL and binary logic will be mixed. Presented examples demonstrate simplicity of realization of a wide class of such circuits, which will also enable the implementation of design automation procedures based on regular data flows and expansions.

The main idea of the presented approach can be summarized as follows: starting from **all possible** neighbor geometries in two and three dimensional spaces, we create **all possible regular structures**. This is more powerful than in the previous structures [1] which considered limited planar geometries. Also, we design the structure step-by-step in the processo of expansions and reverse expansions until functions are trivial, while Akers creates the worst-case structure from the scratch, which is **extremely** wasteful for most functions. On nearly all of investigated by us functions our areas and numbers of cells were much better than those of Akers. Also, Akers did not show methods for Davio, multi-valued and fuzzy, which makes our approach much more general. Similarly, we can show that our approach is more general than PLAs, fat trees or other known structures.

Next we design **arbitrary expansions** for any of the structures. New expansions can be constructed based on the orthogonal logic approaches, or any other canonical or non-canonical function expansions.

We demonstrated the very high number of **various new expansions**, in contrast to only Shannon expansion types used by Akers [1].

This way, the same, in principle, layout-driven synthesis approaches are created for binary, multivalued, orthogonal, Galois and continuous functions [42]. Thus, the presented approach generalizes and unifies many known expansions, decision diagrams, and regular layout geometries. These methods are of special interest to new technologies that use pass-transistors and which require assuming high noise, high resistance/capacitance, crosstalk and long-line effects in connections. We believe that all these effects and phenomena will cause in future, for one reason or another, the increased interest in regular realizations of multi-valued, binary, and continuous logics. Our patent covers not only continuous logic but any logic included in it, thus multi-valued and binary. We presented methods that exceed narrow boundaries of how circuits have been designed until now.

Bibliography

[1] S.B. Akers, "A rectangular logic array," *IEEE TC*, Vol. C-21, pp. 848-857, Aug. 1972.

[2] R.E. Bryant, "Graph-based algorithms for boolean function manipulation, *IEEE TC*, Vol. C-35, No. 8, pp. 667-691, 1986.

[3] L.O. Chua, "Cellular Neural Networks: Theory", IEEE Trans. on Circuits and Systems, Vol. 35, No. 10, pp. 1257–1272, Oct. 1988.

[4] L.O. Chua, "Cellular Neural Networks: Applications", IEEE Trans. on Circuits and Systems, Vol. 35, No. 10, pp. 1273–1290, Oct. 1988.

[5] Concurrent Logic Inc., "CLI 6000 Series Field Programmable Gate Arrays," *Prelimin. Inform.*, Dec. 1, 1991, Rev. 1.3.

[6] K. Current, "Multiple Valued Logic: Current-mode CMOS Circuits", pp. 176–181, ISMVL'93.

[7] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M.A. Perkowski, "Efficient representation and manipulation of switching functions based on Ordered Kronecker Functional Decision Diagrams," submitted to DAC conference.

[8] W.A. Fisher, R. Fujimoto, and R. C. Smithson, "A Programmable Analog Neural Network Processor", IEEE Trans. on Neural Networks, Vol. 2, No. 2, pp. 222–229, Mar. 1991.

[9] F.C. Furtek, "Programmable Logic Cell and Array", U.S. Pat. No 4,918,440, Apr. 17, 1990.

[10] A. El Gamal, "Programmable Interconnect Architecture", U.S. Pat. No 4,873,459, Oct. 10, 1989.

[11] A. El Gamal, J. Greene, J. Reyneri, E. Rogoyski, K.A El–Ayat, and A. Mohsen, "An Architecture for Electrically Configurable Gate Arrays", IEEE J. Solid–State Circ., Vol. 24, No. 2, pp. 394–398, Apr. 1989.

[12] B. Gilbert, "A New Wide-Band Amplifier Technique", IEEE Journal of Solid State Circuits, Vol. SC–3, No. 4, pp. 353–365, Dec. 1968.

[13] B. Gilbert, "A Precise Four-Quadrant Multiplier with Sub-nanosecond Response", IEEE Journal of Solid State Circuits, Vol. SC–3, No. 4, pp. 365–373, Dec 1968.

[14] B. Gilbert, "Current-mode Circuits From a Translinear Viewpoint: A Tutorial", in: Analogue IC Design: the current-mode approach, ed. C. Toumazou, F. J. Lidgey, D. G. Haigh, pp. 11–91, Peter Peregrinus Ltd., 1990.

[15] B. Gilbert, "A New Wide–Band Amplifier Technique", IEEE Journal of Solid State Circuits, Vol. SC–3, No. 4, pp. 353–365, Dec. 1968.

[16] A.B. Grebene, Bipolar and MOS Analog Integrated Circuit Design, J. Wiley, 1984.

[17] H.P. Graf, and D. Henderson, “A Reconfigurable CMOS Neural Network”, ISSCC, IEEE, San Francisco, CA, Feb. 1990.

[18] P.R. Grey, and R. G. Meyer, Analysis and Design of Analog Integrated Circuits, 3rd ed., J. Wiley, 1993.

[19] K. Halonen, V. Porra, T. Roska, and L. Chua, “Programmable Analog VLSI CNN Chip with Local Digital Logic”, ISCAS, IEEE, pp. 1291–1294, Singapore, 1991.

[20] H. Harrer, J.A. Nossek, and R. Stelzl, “An Analog Implementation of Discrete–Time Cellular Neural Networks”, IEEE Trans. on Neural Networks, Vol. 3, No. 3, pp. 466–476, May 1992.

[21] Hausner, A., Analog and Analog/Hybrid Computer Programming, chap. 11, Prentice–Hall, Inc., Englewood Cliffs, N.J., 1971.

[22] B. Kosko, Neural Networks and Fuzzy Systems, A Dynamical Systems Approach to Machine Intelligence, Prentice Hall, Englewood Cliffs, NJ, 1992.

[23] D.A. Johns, W.M. Snelgrove, and A. S. Sedra, “Continuous-Time Analog Adaptive Recursive Filters”, ISCAS, IEEE, Portland, OR, 1989.

[24] Z. Kohavi, ”Switching Circuits and Finite Automata Theory,” Mc Graw Hill, 1972.

[25] F.J. Kub, K.K. Moon, I.A. Mack, and F. M. Long, “Programmable Analog Vector–Matrix Multipliers”, IEEE Journal of Solid-State Circ., Vol. 25, No. 1, pp. 207–214, Feb. 1990.

[26] F.J. Kub, I.A. Mack, and K. K. Moon, “Programmable Analog Voltage Multiplier Circuit Means”, U.S. Pat. No 4,931,674, June 5, 1990.

[27] E.K.F. Lee, and P.G. Gulak, “A CMOS Field–Programmable Analog Array”, 1991 IEEE ISSCC Dig. Technical Papers, Vol. 34, pp.186–187, Feb. 1991.

[28] E.K.F. Lee, and P.G. Gulak, “A CMOS Field–Programmable Analog Array”, IEEE J. Solid–State Circ., Vol. 26, No. 12, pp. 1860–1867, Dec. 1991.

[29] E.K.F. Lee, and P.G. Gulak, “Field Programmable Analogue Array Based on MOSFET Transconductors”, Electronics Letters, Vol. 28, No. 1, pp. 28–29, IEE, Jan. 2 1992.

[30] K.H. Loh, D.L. Hiser, W.J. Adams, and R. L. Geiger, “A Versatile Digitally Controlled Continuous–Time Filter Structure, with Wide–Range and Fine Resolution Capability”, IEEE Trans. on Circuits and Systems, Vol. 39, No. 5, pp. 265–276, May 1992.

[31] S. Manetti, and M. C. Piccirilli, “A Fully Programmable Structure for Continuous–Time MOS Filters”, 6th Mediterranean Electrotechnical Conference, (MELECON ’91), IEEE, pp. 355–358, Ljubljana, Yugoslavia, May 22–24, 1991.

[32] J. Mann, R. Lippmann, B. Berger, and J. Raffel, “A Self-Organizing Neural Net Chip”, CICC, IEEE, pp. 10.3.1–10.3.5, 1988.

[33] M. Marchesi, G. Orlandi, F. Piazza, and A. Uncini, “Neural Networks with Self–Adaptive Topology”, ISCAS, IEEE, pp. 2498–2501, Singapore, 1991.

[34] M.S. Melton, T. Phan, D.S. Reeves, and D.E. Van den Bout, “The TInMANN VLSI Chip”, IEEE Trans. on Neural Networks, Vol. 3, No. 3, pp. 375–384, May 1992.

[35] J. W. Mills, “Area-Efficient Implication Circuits for Very Dense Lukasiewicz Logic Circuits”, ISMVL ’92, pp. 291–298, May 1992.

[36] J. W. Mills, "Lukasiewicz' Insect: The Role of Continuous-Valued Logic in a Mobile Robot's Sensors, Control, and Locomotion", ISMVL'93, pp. 258–263.

[37] S. Paul, K. Hamper, and J. A. Nossek, "A Simple Analog Rank Filter", ISCAS, pp. 121–124, San Diego, CA, 1992.

[38] M. A. Perkowski, "A Fundamental Theorem for EXOR Circuits", Proc. IFIP W.G. 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design, pp. 52–60, Hamburg, Germany, Sep 1993.

[39] M.A. Perkowski, A. Sarabi, and F. R. Beyl, "Universal XOR Canonical Forms of Switching Functions", Proc. IFIP W.G. 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design, pp. 27–32, Hamburg, Germany, September 1993.

[40] E. Pierzchala, and M.A. Perkowski, "High Speed Field Programmable Analog Array Architecture Design," accepted, *Proc. FPGA'94*, Berkeley, California.

[41] E. Pierzchala, and M.A. Perkowski, Patent pending, Analogix Corporation, date withdrawn.

[42] E. Pierzchala, M.A. Perkowski, and S. Grygiel, "A Field Programmable Analog Array for Continuous, Fuzzy, and Multi-Valued Logic Applications," submitted to ISMVL'94.

[43] E. Pierzchala, and M.A. Perkowski, "High Speed Field Programmable Analog Array Architecture Design", submitted to FPGA'94 conference. First version.

[44] E. Pierzchala, "The Design of a Current-Mode Signal Processing Cell for a Field Programmable Analog Array", Ph.D. in preparation.

[45] A. Rodriguez-Vazquez, S. Espejo, R. Dominguez-Castro, J.L. Huertas, and E. Sanchez-Sinencio, "Current-Mode Techniques for the Implementation of Continuous- and Discrete-Time Cellular Neural Networks", IEEE Trans. on Circuits and Syst., II: Analog and Digital Sign. Proc., Vol. 40, No. 3, March 1993.

[46] T. Roska, and L.O. Chua, "The CNN Universal Machine: An Analogic Array Computer", IEEE Trans. on Circuits and Syst., II: Analog and Digital Sign. Proc., Vol. 40, No. 3, March 1993.

[47] A. Sarabi, N. Song, M. Chrzanowska-Jeske, and M. A. Perkowski, "A Comprehensive Approach to Logic Synthesis and Physical Design for Two-Dimensional Logic Arrays," submitted to DAC.

[48] S. Satyanarayana, Y.P. Tsividis, and H. P. Graf, "A Reconfigurable VLSI Neural Network Chip", IEEE J. Solid–State Circ., Vol. 27, No. 1, pp. 67–81, Jan. 1992.

[49] R. Schaumann, M.S. Ghausi, and K.R. Laker, "Design of Analog Filters," Prentice Hall, Englewood Cliffs, NJ, 1990.

[50] D.B. Schwartz, R.E. Howard, and W. E. Hubbard, "A Programmable Analog Neural Network Chip", IEEE J. Solid–State Circ., Vol. 24, No. 2, pp. 313–319, Apr. 1989.

[51] B.J. Sheu, "VLSI Neurocomputing with Analog Programmable Chips and Digital, Systolic Array Chips", ISCAS, IEEE, pp. 1267–1270, Singapore, 1991.

[52] M.A. Sivilotti, "A Dynamically Configurable Architecture for Prototyping Analog Circuits", in Advanced Research in VLSI, Fifth MIT Conf., ed. J. Allen, F. Thomson Leighton, pp. 237–258, The MIT Press, Cambridge, MA, Mar. 1988.

[53] J. Van der Spiegel, P. Mueller, D. Blackman, P. Chance, Ch. Donham, R. Etienne–Cummings, and P. Kinget, "An Analog Neural Computer with Modular Architecture for Real–Time, Dynamic Computations", IEEE J. Solid–State Circ., Vol. No. 1, pp. 82–92, Jan. 1992.

[54] M.A. Tan, Design and Automatic Tuning of Fully Integrated, Transconductance–Grounded Capacitor Filters, Ph.D. Thesis, Univ. of Minnesota, 1988.

[55] K. Taniguchi, M. Sasaki, Y. Ogata, F. Ueno, and T. Inoue, "BiCMOS Current Mode Multiple Valued Logic Circuits with 1.5V Supply Voltage", ISMVL'92, pp. 216–220.

[56] S. Ulam, "Random Processes and Transformations", Int. Congr. Mathem. (held in 1950), Vol. 2, pp. 264–275, 1952.

[57] J. Von Neumann, Theory of Self–Reproducing Automata (edited and completed by Artur Burks), Univ. of Illinois Press, 1966.

[58] J.E. Varrientos, E. Sanchez-Sinencio, and J. Ramirez-Angulo, "A Current-Mode Cellular Neural Network Implementation", IEEE Trans. on Circuits and Syst., II: Analog and Digital Sign. Proc., Vol. 40, No. 3, March 1993.

[59] G.A. De Veirman, and R. G. Yamasaki, "Monolithic 10 – 30 MHz Tunable Bipolar Bessel Lowpass Filter", ISCAS, IEEE, Singapore, pp. 1444–1447, 1991.

[60] Z. Zilic, and Z. Vranesic, "Current-mode CMOS Galois Field Circuits", ISMVL '93, p. 245–250.

[61] The Programmable Gate Array Data Book, Xilinx, San Jose, CA, 1989.

[62] S.R. Zarabadi, F. Larsen, and M. Ismail, "A Reconfigurable Op-Amp/DDA CMOS Amplifier Architecture", IEEE Trans. on Circuits and Syst., II: Analog and Digital Sign. Proc., Vol. 39, No. 6, June 1992.

[63] M. Chrzanowska-Jeske, M. Perkowski, TANT Networks with Multiple-Valued inputs, to be submitted to ISMVL'94.

[64] M.A. Perkowski, "The method of solving combinatorial problems in the automatic design of digital systems". Institute of Automatic Control, Technical University of Warsaw, Ph.D. Thesis, 1980, 434 pages in Polish. Name of Advisor: Professor Wieslaw Traczyk, Director of the Institute of Automatic Control, Department of Electronics, Technical University of Warsaw, Warsaw, Poland, 1980.

[65] M.A. Perkowski, "Application of logical schemata of algorithms to the synthesis of automata," *M.S.Thesis*, Institute of Automatic Control, Technical University of Warsaw, 247 pages in Polish, 1970.

[66] Li-Fei Wu, and Marek A. Perkowski, "Minimization of Permuted Reed-Muller Trees for Cellular Logic Programmable Gate Arrays," *Proc. of the 2nd Intern. Workshop on Field-Programmable Logic and Applications,* FPL'92, Vienna, Austria, pp. 7/4.1-7/4.4, August 31-September 2, 1992.

[67] A. Sarabi, P. F. Ho, K. Iravani, W. R. Daasch, M. A. Perkowski, "Minimal Multi-Level Realization of Switching Functions Based on Kronecker Functional Decision Diagrams," *Proc. of IEEE International Workshop on Logic Synthesis,* IWLS '93, Tahoe City, CA, pp. P3a-1 - P3a-6, May 1993.