RECONFIGURABLE-COMPUTING TECHNOLOGY

A. V. Palagin[†] **and V. N. Opanasenko**[‡]

UDC 004.274

Evolution of computers with flexible architecture is considered. Distinctive features of the architectural and structural organization of reconfigurable computer systems (RCSs) are described. Based on a modified logical-informational method and a configuration file library, a general algorithm is proposed to design RCSs. A new class of universal information processing means is proposed, namely, adaptive logical networks.

Keywords: *adaptive logical network, architecture flexibility, electrically programmable logic devices (EPLD), reconfigurable computing.*

EVOLUTION OF A COMPUTER WITH FLEXIBLE ARCHITECTURE

Development of the technology of erasable electrically programmable logic devices (EPLD) and architectures of modern computers and information and computing systems of all classes became a basis for intensive improvement of reconfigurable computing (RC) technology and expansion of its application. This was due to the simplicity of RC-based designing of computer aids, efficient problem orientation, real-time adaptation to changes of objective functions and environment, modernization and extension of functionalities, and modification of the operation algorithm. Moreover, architectural flexibility is a principal indicator of the internal intelligence and efficiency of a computer system; this is true for both hardware and software. That is why RC allows solving a lot of methodological, technological, and applied problems of constructing and designing knowledge-oriented information systems, especially as applied to sophisticated research that efficiently supports the formation, development, and use of knowledge bases.

There are *subject areas* where reconfigurable computer systems (RCSs) got to their rightful place and continue developing intensively [1, 2]. These are:

• survivable systems providing control safety for especially critical objects;

• sophisticated physical experiments with real-time simulation and control;

• efficient digital processing of high-frequency signals;

• improvement of computer design aids for new-technology objects;

• ontology-controlled intelligence systems serving the interface of various user's systems with the Internet and interdisciplinary cooperation;

• emulation and design of wireless communication systems, etc.

These application domains are important and promising, which indicates that reconfigurable computing and related problems are topical.

During the evolution, computer systems, and first of all their processor component, were developed mainly to provide flexibility (adaptivity) of the baseline architecture, which, in turn, met the requirements of problem-orientation of computer equipment to increase their efficiency in specific applications. V. M. Glushkov Institute of Cybernetics continued famous traditions of the computer-engineering pioneers and became an active participant of this process both in the development of computer science theory in the large and in the elaboration and production startup of specific classes and models of flexible-architecture computers.

V. M. Glushkov Institute of Cybernetics, National Academy of Sciences of Ukraine, Kyiv, Ukraine, [†]*palagin_a@ukr.net*, [‡]*vlopanas@ukr.net.* Translated from Kibernetika i Sistemnyi Analiz, No. 5, pp. 72–86, September–October 2007. Original article submitted May 22, 2007.

The advent of *microprogram processors*, computers, and systems is a historically important stage on this path. Microprogramming simplified the design process. Its key features are regular structure of the stored control logic and convenient verification and debug of microprograms. The world famous MIR computer became a prominent representative of computers with microprogram interpretation of external programming languages. Noteworthy are *dynamic microprogramming* aids, which permit changing microprograms in the modernization and reorientation of a computer system (after its manufacture).

The second stage of the evolution is creating *emulating* computer systems, which can modify and completely change the internal language. The concept of flexibility of CS architecture has been formulated:

$$\Xi = \{\Psi_a\}; \;\; \Psi_a \| \sigma_k^a \|, \quad a = 1, 2, \ldots, A,$$

where A is the number of possible architectures for a specific hardware, σ_k^a is the complete image of the kth level of the architecture of type a.

The third stage is concerned with *multilevel* microprogram control systems and multioperational single-chip microprocessors. V. M. Glushkov Institute of Cybernetics actively participated in creating this class of microcomputer as well (“Elektronika C5-01” (C5-11, 12)) and contributed significantly to the development of the theory of their architectural and structural organization and design [2, 3]. In particular, a *logical-information method* (LIM) of designing microprocessor systems has been developed and tested in practice. It combines theoretical concepts of the theory of digital automata and information theory and can be illustrated as follows:

$$\begin{tikzcd}
\underset{i}{\forall}\, \exists\Omega:\ \mathrm{A_N} \arrow[r, Rightarrow] & \Lambda_\mathrm{N} \arrow[r, Rightarrow] \arrow[d, dotted] & \mathrm{R_N} & & \\
& \mathrm{A_i} \arrow[r, Rightarrow] & \Lambda_\mathrm{i} \arrow[r, Rightarrow] \arrow[d, dotted] & \mathrm{R_i} & \\
& & \mathrm{A_0} \arrow[r, Rightarrow] & \Lambda_0 \arrow[r, Rightarrow] & \mathrm{R_0}(\Theta = \Theta_{\mathrm{extr}}(\mathrm{Q}, \mathrm{t})),
\end{tikzcd} \tag{1}$$

where A_i, Λ_i, R_i $(i = \overline{N, 0})$ are sets of algorithms, operators, and their information-code representations at the ith programming level, respectively, and Θ is the set of generalized characteristics (hardware (Q), time (t), etc.).

The generalized characteristics of formal model (1) can easily be converted into engineering parameters of EPLDs and information characteristics of automatic representations of each $(i = \overline{N, 0})$ hierarchy level of programmable automata. A technique for entropic estimation of the information complexity of an automaton in terms of probabilities of its transitions and states was proposed in [1]. For example, to generate the matrix of transition probabilities, the probability of each transition from a state a_α to a state a_β $(\alpha, \beta = \overline{1, M})$ is determined:

$$p\,(a_\beta) = \delta(a_\alpha, x) = p_{\alpha\beta}. \tag{2}$$

Given the transition matrix $\| p_{\alpha,\beta} \|$ $(\alpha, \beta = \overline{1, M})$, the matrix of the probability P_β $(\beta = \overline{1, M})$ that the automaton is in each $(\alpha, \beta = \overline{1, M})$ state is determined by solving the system of linear equations

$$P_\beta = \sum_{\alpha=1}^{M} P_\alpha\, p_{\alpha\beta}; \;\; \beta = \overline{1, M}, \;\; \sum_{\alpha=1}^{M} P_\alpha = 1. \tag{3}$$

The total hardware scope (Q) of an automaton composition is estimated to be proportional to the sum of entropic estimates of transition and output functions, each being calculated from the well-known entropy formula for the average length of the αth control word:

$$H_\alpha = -\sum_{\alpha=1}^{M} P_\alpha \log_2 p_\alpha. \tag{4}$$

In (2)–(4), the probability is the frequency of the αth state.

An example of a two-level microprogram control system is the Elektronika C5-21 single-chip microcomputer [2]. The evolution of microsystem architectures was supported by the creation of programmable logic arrays, which made it possible to automatize both the design and production of sophisticated chip-based computer systems.

The design of microprogram systems [3] involved the solution of two optimization problems:

(i) microinstruction coding:

$$W \to A \to C \to n = \sum_k \sum_m \mathrm{Ent}(\log_2 q_{km})x_k \to n_{\min},$$

where W and A are the matrices of microinstructions and incidences of microoperations (MO), respectively, $C = \{C_m\}$, $m = \overline{1, M_c}$, $M_c = \mathrm{Card}\{C\}$, is the set of MO compatibility classes, n is the length of a microinstruction word, x_k is an alternate covering ($x_k \in \{0, 1\}$, $\sum_k x_k = 1$, $k = 1, 2, \ldots,$ is the number of alternate coverings);

(ii) generating the address of microinstructions (determining the set of compatibility classes of input patterns, page size of microprogram memory, etc.).

Modern EPLDs opened up a new stage of evolution — creation of high-performance RCSs.

The term "reconfigurable computing" combines two concepts: (i) reconfigurable computer structure (hardware) and (ii) data handling. Reconfigurability is a natural requirement to complex systems that should be highly reliable and flexible to adapt to the structure of problems being solved and external conditions.

To formalize the model of reconfigurable devices, the LIM method is modified as follows:

$$\begin{tikzcd}
\forall_{i}\exists\Omega:\ A_N \arrow[r, Rightarrow] \arrow[d, dotted] & \Lambda_N \arrow[r, Rightarrow] \arrow[d, dotted] & R_N & & \\
\bullet \arrow[r] \arrow[d, dash] & A_i \arrow[r, Rightarrow] \arrow[d, dotted] & \Lambda_i \arrow[r, Rightarrow] \arrow[d, dotted] & R_i & \\
{} \arrow[rr] & & A_0 \arrow[r, Rightarrow] & \Lambda_0 \arrow[r, Rightarrow] & R_0\ (\Theta = \Theta_{extr}(Q, t)).
\end{tikzcd} \tag{5}$$

According to (1), R_i is the set of hardware realizations of ith level operators, with some levels being excluded in synthesizing the optimal structure. The following programming levels are used in (5) for classical architectures: τ_0 is physical or "zero" level; τ_1 is microprogram level; τ_2 is program level; and τ_3 is algorithmic level. Programming at the "zero" level determines the physical structure of the device, which will finally implement the prescribed algorithm, i.e., program the device structure. In contrast to (1), modification (5) implements not a microprogram but rather a hardware gate-level realization of algorithms. This is how the reconfigurable devices with programmed structure differ from modern computers.

The reconfigurability principle means that the logical structure of a reconfigurable device can vary dynamically both in preparing to problem solution and during computations.

The reconfigurability principle is peculiar to all complex systems. In particular, it contributes to the survivability, expandability, and other properties of such systems. If one of the components fails, the complex system can continue operating after changing logical connections among components. The reconfigurability principle allows one to easily reprogram the structure of the device so that it would efficiently implement a prescribed algorithm, which determines the structural universality of the device.

ARCHITECTURE OF RECONFIGURABLE COMPUTER SYSTEMS

Putting EPLDs and HDL-technologies to use (HDL stands for hardware description language) intensified the development of a wide scope of digital modules, which are ready engineering solutions that substantially reduce the time needed to design and manufacture new products [4]. These solutions called IP units (IP stands for "intellectual property"), cores, and parametric modules can be adjusted or initially prepared to meet specific requirements of a new project. They are classed among soft cores or soft units described in an HDL-language at, say, the register transfer level. Such units can easily be adjusted to a new project and they are generally independent of EPLD manufacturing technologies [5].

The most important property of a ready engineering solution is that its functions are guaranteed to be reproduced in a new project according to the specification given by the developer of this solution and amended by the project developer. Note that HDL-description allows not only making the model readjustable and technology-independent but also using various commercial tools to perform simulation and synthesis. Owing to network technologies, the designer may get, via the Internet, an optimized logical core according to technical requirements and include it in the project. The IP strategy is introduced into many branches of technology such as DSP, communications, networks, and general-purpose computers.

The capabilities of the HDL-technology such as hierarchical design, library portability, and platform independence allow using soft cores for macroelements to develop new engineering solutions.

The modern FPGA chip architecture is optimized to use both hard and soft cores and allows integrating them easily into projects. For example, Virtex chips have internal multipliers and PowerPC processors as hard cores and other functional units.

A standard *reconfigurable computer system* (RCS) usually consists of two parts: constant (or "fixed") H (host computer) and variable R (*reconfigurable subsystem* (RSS)), which can be combined into various configurations. The architecture of reconfigurable systems [6] depends on the potency of the set of algorithms (N):

$$N = N_H + N_R,$$

where N_H and N_R are the potencies of the subsets of algorithms run on the H and R hardware, respectively. The combination of these quantities determines the RCS classification being proposed:

• RCSs with functional orientation to the host computer, which carries major computing power; RSS increases the performance for a narrow (yet important for the user) class of problems ($N_H \to N$, $N_R \to 0, N_H >> N_R$) and is a coprocessor;

• RCSs with functional orientation to RSSs ($N_H \to 0$, $N_R \to N$, $N_R >> N_H$), where the host computer is used for auxiliary functions (service, input-output) and all algorithms are mainly run on the RSS, which may have an independent field of peripherals (via expansion cards) or a common field of peripherals with the host computer which are directly accessed by the RSSs;

• RSS is a stand-alone device in the case of $N_H = 0$, $N_R = N$, the host computer being absent;

• combined RCSs, where the host computer and RSS have approximately similar functions ($N_H \approx N_R$).

In the computer systems under consideration (except for the stand-alone ones), the RSS is connected to the host computer via a standard bus such as PCI and PCI-Express buses, which are most popular today.

GENERALIZED STRUCTURE OF RSS

The market offers now a wide range of modules for RSSs. They differ by a hardware controller (if any) of the bus connected to the host computer and by other features.

RSSs or devices with programmable architecture [7–9] have a common *functional processing field* (FPF). It is specially configured to implement some algorithm or its part in a way optimal according to given criteria.

The algorithm can be split into fragments run sequentially; therefore, the associated structures are also loaded into the chip sequentially (in the order they are run), which saves resources substantially. The complexity of the fragments is limited by the logical capacity of the chip, i.e., by the processing field dimension.

The reconfigurable computing technology is a variation of the central paradigm of modern computer aids design.

FPGA-type EPLDs are mainly used as elements for RSSs [5]. The configuration file can be stored in the ROM and automatically loaded to the EPLD chip (upon powerup). In another mode, the configuration file is loaded to the EPLD chip from the RAM. Configurability allows changing a product under operation or to use it to implement various algorithms.

Depending on the purpose and functions (network data processing, DSP, etc.), RSSs can have appropriate structure, embodiment, and resources (logical capacity of the chip, the number of outputs, memory capacity, data bus width (16, 32, or 64), etc.). RSS nomenclature may be rather extensive; therefore, we will consider the most general features of RSS structure regardless their embodiment and resources required. Figure 1 shows a flow chart of the base component of an RSS connected the bus of the host computer.

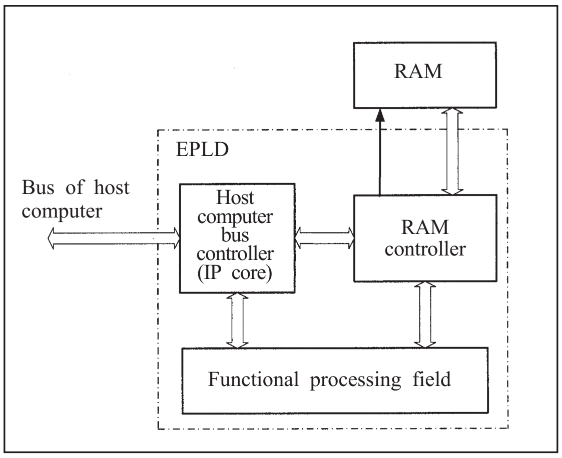
RAM
EPLD
Bus of host computer
Host computer bus controller (IP core)
RAM controller
Functional processing field

Fig. 1. Base component of RSS.

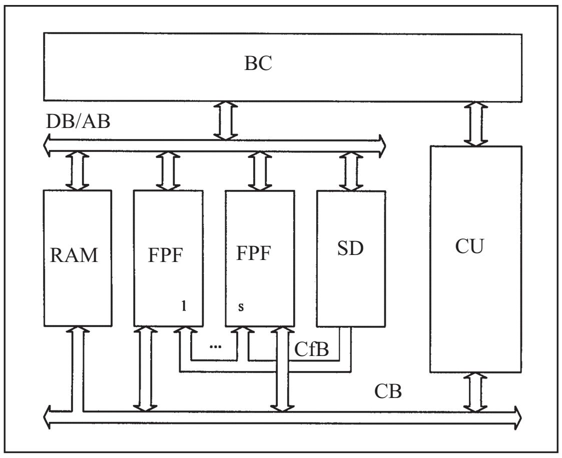

Fig. 2. Generalized structure of RSS.

RSSs have FPF of certain dimension, which can be configured to implement a given algorithm or its part in a way optimal in time and hardware.

Figure 2 represents the generalized structure of an RSS [10] and contains the FPF, the controller of the standard bus of the host computer (BC), storage device (SD) for configuration files, data RAM, control unit (CU), data/address bus (DB/AB), configuration bus (CfB), and control bus (CB). A pipelining mechanism assumes that the configuration file is loaded into the current FPF in parallel with data processing in the current matrix.

The format of the configuration file is standard for FPGA and contains information on the FPF configuration, i.e., corresponds to the basic electric scheme, implements some algorithm. Each $(i=\overline{1,s})$ FPF is a matrix of universal elements, which are assigned a certain function according to the configuration file F_γ, followed by the formation of a connection structure. Configuration files F_γ are written in the corresponding $(i=\overline{1,s})$ FPF from the storage device for configuration files via the CfB bus under the control of the CU.

Data from the RAM or from other FPFs come to the FPF along the DB, and external input data through the BC. The processed data can be transmitted from the FPF to the BC as processed data, to the RAM as intermediate results, or to other FPFs $(i=\overline{1,s})$ for subsequent processing. The set of configuration files $F=\{F_\gamma\}$ is written into the storage device for configuration files via the BC under the control of the CU.

The set s of FPFs allows hardware implementation of parallel data processing while the set of configuration files allows conveyer programming of the structure that implements fragments of the algorithm.

FEATURES OF HIGH-PERFORMANCE RCSs

Creation of supercomputer systems is a promising domain of RCS application. The V. M. Glushkov Institute of Cybernetics makes research efforts to develop the existing supercomputer using the RC technology. In this connection, it is expedient to analyze the world experience to asses the prospect and the capabilities of this line of research.

Let us consider, in particular, the organization of the HERC (High-End Reconfigurable Computing) System developed in Berkeley Wireless Research Center, which created the first BEE (Berkeley Emulation Engine) system prototype with two modules intended to design, produce, and program an HERC system for a number of applications [11]. In developers' opinion, the BEE provides performance higher, by an order of magnitude, than that of a system based on DSP processors with similar power consumption and cost, and by more than two orders of magnitude than that of systems based on standard microprocessors. The principal components of the architecture are the computational node (CN) and global communication networks.

The computational node in the BEE contains five Virtex-type chips directly connected to four memory modules with a maximum capacity of 4 Gb per FPGA chip. The CN uses four FPGAs for computation (processing modules) and one chip for control (controlling modulus). The generalized structure of the CN is shown in Fig. 3.

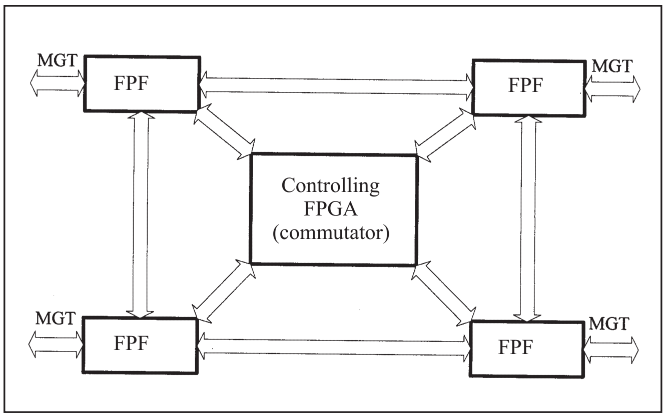

Fig. 3. Computational node.

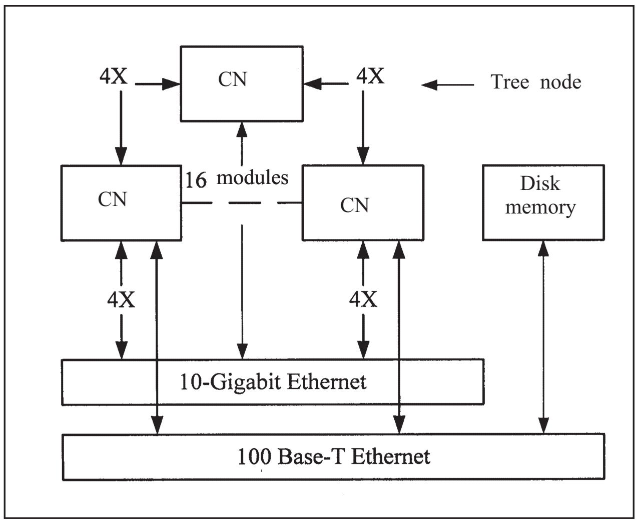

Fig. 4. Global connection tree.

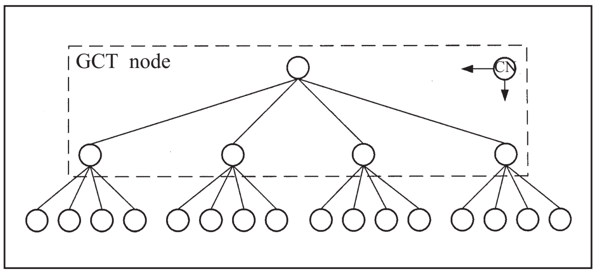

Fig. 5. Realization of a tree-like network.

A local network includes four FPFs connected in a 2D-pattern so that adjacent FPFs are directly coupled to each other, and the FPGA controlling chip acts as a control unit and commutator, forming a common virtual computational environment. Moreover, each FPF can communicate directly with FPF on other modules via the MGT (multigigabit transceiver) serial interface and act as a high-capacity real-time input/output channel (such as gigaherz analog-to-digital or digital-to-analog converters) for peripherals.

All external connections use FPGA-chip MGT. All MGT channels are program-configurable and are united into the InfiniBand 4X (IB4X) physical connector.

Global Connections. The BEE project supports a variety of global connections represented in Fig. 4: the global connection tree with a small idle time, an interconnection network with a high channel capacity (10 Gigabit Ethernet), and 100 Base-T Ethernet.

Each BEE computational node can be a global node of the connection tree, which connects up to 16 other computational modules and up to two independent parent nodes (Fig. 5).

Using physical connections IB4X, computational modules can also form many other interconnection topologies. 100 Base-T Ethernet is accessible only to FPGA controlling chips and provides a communication network for the user interface, control, and data archiving.

ADAPTIVE LOGICAL NETWORKS

Continuing the development of homogeneous computing systems and leaning upon the capabilities of modern EPLDs, we propose a class of universal information processing means as a functional processing field of an RCS. These are adaptive logical networks (ALN) intended to solve a wide range of problems by immediate structural realization of algorithms for processing and mapping the input data set into output data set (with a preliminary functional adjustment) [8]. *An adaptive logical network* is a discrete transducer such as an asynchronous combinational automaton, specified by a directed graph whose nodes are logical functions and ribs are output/input-type links.

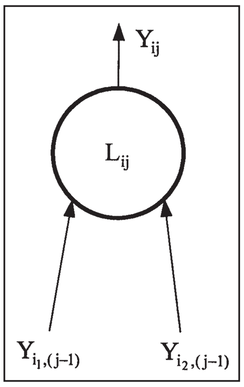

Fig. 6. Structure of the L_{ij} element.

From the topological standpoint, an ALN is a matrix of *universal logical elements* (LE) grouped into functional nodes (FN) and blocks (FB) with fixed location, their operation depending on the class of problems and their purpose.

By a universal LE is meant a combinational automaton $L = \langle n, F \rangle$, where n is the number of binary inputs or the dimension of input variables of the LE; $F = \{f_\rho\}$, $\rho = \overline{[1, 2^{2^n}]}$, is the set of Boolean functions realized by the LE. The LE is universal in the sense that it can be adjusted to realize an arbitrary Boolean function.

Let us introduce some definitions. A vector is an ordered set of binary components (the number of components of the vector is its length (dimension)).

By a mapping of a set of vectors X into a set of vectors Y is meant a rule that associates each vector from the set X with a vector from the set Y. The vectors of the set X are called pre-images of the vectors of the set Y, and the vectors of Y are called images of the vectors of X. A mapping $\Im(X) = Y$ is represented by a superposition of mappings $\Im_m(\Im(\Im_s \ldots \Im_2(\Im_1(X))\ldots)) \Rightarrow Y$, where $\Im_s$ $(s = \overline{1, m})$ determines the structure of connections and types of functions of the sth level.

By the *procedure of separating out* a subset $X_k = \{x_k\}$ in a set of vectors X $(X_k \subset X)$ at a level s is meant the procedure of constructing a mapping $\Im_s$ so that $\Im_s(X_k) \cap \Im_s(X / X_k) = 0$.

By the *procedure of partitioning* a set X into two disjoint equipotent subsets X_1 and X_2 at a level s is meant the procedure of constructing a mapping $\Im_s$ such that

$$X_1^* = \Im_s(X_1),\ X_2^* = \Im_s(X_2);\ \ \mathrm{Card}(X_1^*) = \mathrm{Card}(X_2^*) = 1,\ \ X_1^* \cap X_2^* = 0.$$

The ALN structure can be described by the following system:

$$A = \langle n, h, F, S, L, m, D, X, Y \rangle, \tag{6}$$

where n is the length of input binary vectors (input ALN dimension); h is the output length $(h = \overline{1, n})$ (output ALN dimension); $F = \{f_{ij}\}$ is the set of logical functions of the system; S is the structure of LE connections; $L = \{L_{ij}\}$ is the set of LEs (i is the sequential LE number; j is the number of processing level); m is the number of mapping levels; $D = \{d\}$ is the set n-dimensional binary vectors (learning sample); X is the complete set of input binary vectors $(D \subset X)$; $Y = \{Y_{ij}\}$ is the generalized function of the system, $Y_{ij} = f_{ij}(Y_{i_1,(j-1)}, Y_{i_2,(j-1)})$ is the value of the function f_{ij} realized by the element L_{ij}, $Y \in \{0, 1\}$, whose structure is presented in Fig. 6 (i_1 and i_2 are the values of the index i for inputs of an arbitrary LE).

Each ALN level is a register of ρ-input LEs, each of which can be adjusted to realize any (from the complete set of 2^{2^ρ}) Boolean function of its input variables. At one level, the function type for each LE can be specified separately (elementwise adjustment) or for all LEs (levelwise adjustment).

Functional block is a network of series-connected automata (a hierarchical assembly of functional nodes). We will restrict the consideration to three topologically different ALNs: *rectangular matrix* (RM) $(h = n)$; *triangular matrix* (TM) $(h = 1)$; and *trapezoidal matrix* (TpM) $(h = \overline{2, (n-1)})$, which consists of z composite TMs. These basic blocks can be used to generate a set of alternate functional (combinational) schemes.

FN and FB are adjusted to realize any function: partitioning a priori given sets of vectors into subsets, separating out a class of vectors, generalization with respect to one or several vectors, vector coding and decoding, etc.

Depending on LE connections, there may be ALNs with dichotomic and honeycomb (symmetric and asymmetric) connection structures.

ALN Synthesis Problem. This problem is reduced to determining the types of logical functions for a given (initial) set of vectors. Let us synthesize a TpM as an example.

In the general case, the learning sample $D=\{D^{\gamma}\}$ in system (6) is a set of pre-images ($\bigcap_{\gamma} D^{\gamma}=\varnothing \; \forall \gamma=\overline{1,k}$, $\varnothing$ is an empty set), where each (γth) pre-image $D^{\gamma}=\{d_{\sigma}^{\gamma}\}$ ($\sigma=1,2,\ \ldots$) is an n-dimensional binary vector;

h is the output dimension of the TpM (length of the output binary vectors $y \in Y$) defined by $h=(\mathrm{Ent}\{\log_2 k\}+1)$. The input vectors that do not pertain to the γth image are combined into one code representation.

The set of binary vectors d_{σ}^{γ} of the pre-image D^{γ} is associated with an image g^{γ} ($\bigcap_{\gamma} g^{\gamma}=\varnothing$) that is a binary vector. Each image g^{γ} can be associated with a given output binary vector y^{γ} ($\bigcap_{\gamma} y^{\gamma}=\varnothing$, $y^{\gamma}\in Y$). Thus, the solution algorithm for the classification problem slits into the following steps:

— divide the TpM into h components TM_z ($z=\overline{1,h}$);

— determine a set of learning samples for each TM_z;

— analyze the learning samples and the corresponding components of the output binary vector to determine the set of logical functions for LEs of each TM_z, which is the solution of the synthesis problem.

The output of each TM_z is the component y_z of the vector Y. TM partitions an arbitrary set of binary vectors $X=\{x_z\}$ into two subsets for a given set of pre-images $D_z \subset X_z$ (D_z is a learning sample). Thus, the output y_z is defined as follows:

$$y_z=\begin{cases}1 \text{ if } x_z \in D_z\\ 0 \text{ otherwise.}\end{cases}$$

Let us synthesize a TpM with symmetric honeycomb structure as an example. The learning sample is two subsets of pre-images ($D^1, D^2 \subset D$):

$$D^1=\{1000\},\{1011\},\{1101\},\{1110\};\ D^2=\{0000\},\{0011\},\{0101\},\{0110\};$$

$$D\setminus X=\{0001\},\{0010\},\{0100\},\{0111\},\{1001\},\{1010\},\{1100\},\{1111\}.$$

$$D^1=\begin{Vmatrix}1&0&0&0\\1&0&1&1\\1&1&0&1\\1&1&1&0\end{Vmatrix};\quad D^2=\begin{Vmatrix}0&0&0&0\\0&0&1&1\\0&1&0&1\\0&1&1&0\end{Vmatrix}.$$

Let the images g^{γ} be encoded as follows:

$$g^1=\{1,1\},\ y_1=1,\ y_2=1;\quad g^2=\{1,0\},\ y_1=1,\ y_2=0;\quad \overline{(y_1\,\&\,\bar{y}_2 \cup y_1\bar{y}_2)}\to(D\setminus X). \tag{7}$$

In this case, TM_1 uses the components of the input vector with the index $i=\overline{1,3}$:

$$D_1^1=\begin{Vmatrix}0&0&0\\0&1&1\\1&0&1\\1&1&0\end{Vmatrix},\quad D_1^2=\begin{Vmatrix}0&0&0\\0&1&1\\1&0&1\\1&1&0\end{Vmatrix};$$

TM_2 — $i=\overline{1,4}$:

$$D_2^1=\begin{Vmatrix}1&0&0\\1&0&1\\1&1&0\\1&1&1\end{Vmatrix},\quad D_2^2=\begin{Vmatrix}1&0&0\\1&0&1\\1&1&0\\1&1&1\end{Vmatrix}.$$

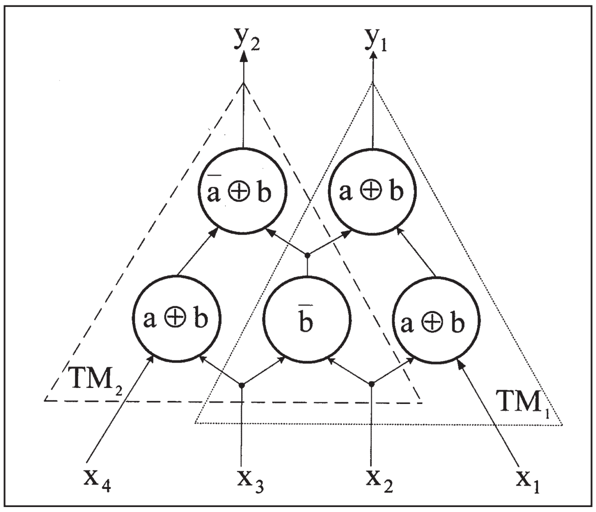

Fig. 7. TpM with symmetric honeycomb structure.

Learning samples for TM_1 and TM_2 are formed based on the matrices D_z^γ.

Considering (7), we unite D_1^1 and D_1^2 for the TM_1 and D_2^1 and D_2^2 for TM_2 and obtain generalized learning samples used to find the functions of logical elements TM_1 and TM_2:

$$D_1 = \begin{Vmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{Vmatrix}, \quad D_2 = \begin{Vmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{Vmatrix}.$$

To determine the unknown set of logical functions $F = \{f_{ij}\}$, the TM structure can be described by polynomials based on Hadamard's matrices [12].

For any Boolean function f of n variables that take values from the set $\{1, -1\}$, there exists an equivalent polynomial $P_{f(n)}$ with coefficients from the set of real numbers: $f(X) = P_{f(n)}(X)$.

A polynomial of n variables has the following form:

$$P_{f(n)} = P_{f(n-1)} + z_n x_n P_{f(n-1)}.$$

A function of two variables can be represented by a polynomial

$$P_{f(2)} = z_0 + z_1 x_1 + z_2 x_2 + z_3 x_1 x_2.$$

The coefficients of the polynomial for $P_{f(2)}$ are defined by

$z_0 = 1/4(y_0 + y_1 + y_2 + y_3)$;

$z_1 = 1/4(y_0 - y_1 + y_2 - y_3)$;

$z_2 = 1/4(y_0 + y_1 - y_2 - y_3)$;

$z_3 = 1/4(y_0 - y_1 - y_2 + y_3)$.

An analysis of asymptotic estimates of the complexity of combinational schemes [8, 13] reveals rather high performance of ALN. The TpM for the above-mentioned example is shown in Fig. 7.

DESIGN OF RECONFIGURABLE COMPUTER SYSTEMS

Figure 8 presents an algorithm for the design of an RSS of arbitrary form and purpose. The design technique developed leans upon this algorithm and the underlying logical and information model of an RSS. It allows accomplishing

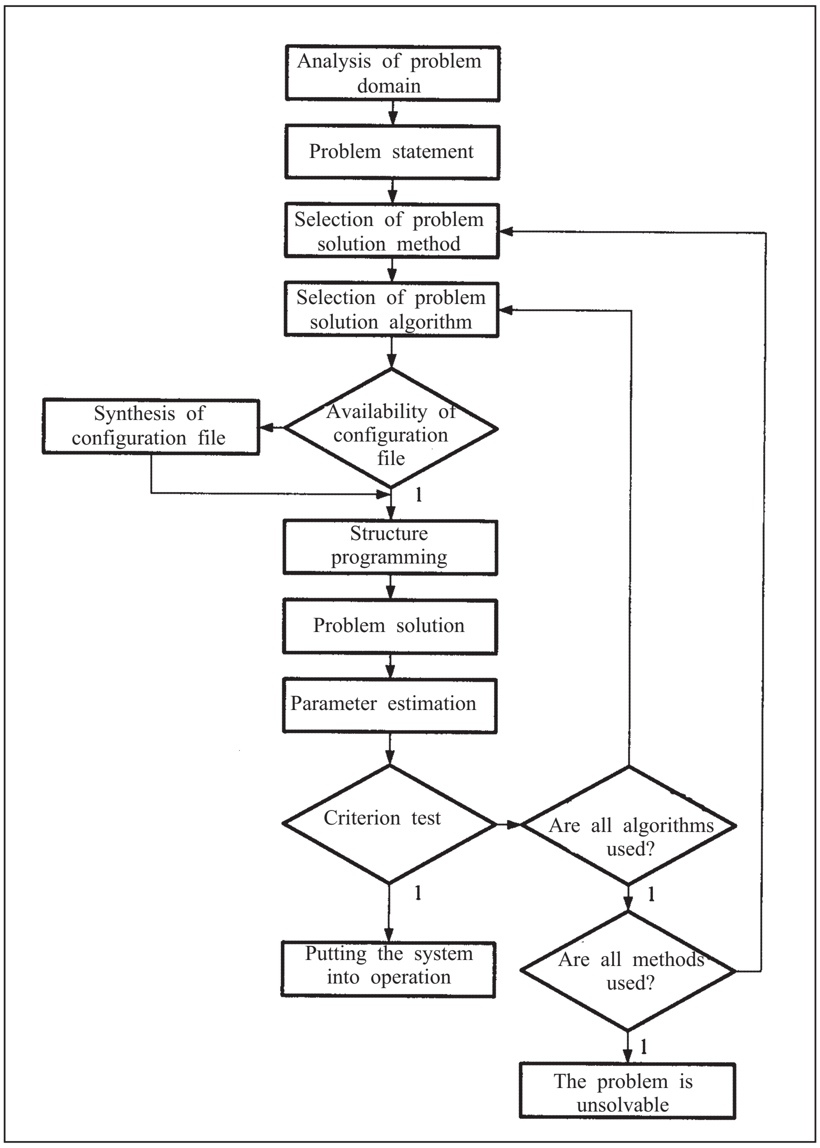
Analysis of problem domain
Problem statement
Selection of problem solution method
Selection of problem solution algorithm
Synthesis of configuration file
Availability of configuration file
1
Structure programming
Problem solution
Parameter estimation
Criterion test
Are all algorithms used?
1
1
Putting the system into operation
Are all methods used?
1
The problem is unsolvable

Fig. 8. RSS design algorithm.

the main task of the design: to formalize searching for an optimal "algorithm–structural realization" pair. The technique is intended to design problem-oriented coprocessors and stand-alone devices that operate with a specified set of algorithms; reconfigurable processors with data pipelining; parametric IP-core to implement given algorithms represented by elements of the library of configuration files; chip-based systems, etc. It can be modified depending on the initial task, class of problems, elements and technologies, etc.

In the known methods of formalized design of computer devices, the process is a sequence of stages, at each of which the system is represented by a set of mathematical models that describe its different parts [14]. There are three main forms of models: functional, dynamic, and structural.

The first decomposition level is related to the analysis of the initial algorithm and its partition into fragments. The fragments are used to obtain the final result — the description (behavioral or structural) of the computer system operation in the CAD source language (for example, VHDL).

The model of the computing device being designed complies completely with the flow chart in Fig. 8 and can be represented by the quadruple

$$S = <M, A, B, D>,$$

where M is the set of mathematical methods applicable to solve the problem posed by the user; A is the set of algorithms implementing the method; $B = \{b\}$ is the alphabet of constructs used to synthesize the structure; and D is the project description procedure (object description). Thus, the design process consists of the synthesis of the structure based on constructs $\{b\}$ of the alphabet B to implement a certain alphabet A implementing the method M, according to specifications. The result of the procedure D is a project description in the CAD source language.

The choice of the method and the algorithm implementing it is the subject of separate study, as regards both the general methodology and practical techniques and guidelines. Note that the solution of the initial problem is iterative, and the efficiency criteria for the desired method (algorithm) are the generalized performance characteristics, hardware scope, solution accuracy, algorithm complexity, reliability of the system being designed, or special criteria such as real-time operation, labor input, etc.

The general approach to the optimal solution of an arbitrary problem in an arbitrary data domain is hardly possible. It seems to be more realistic to create a well-structured library of methods and corresponding architectures of the computing system being designed (stored as program files in the external memory of the base computational subsystem) and to choose an appropriate pair (method–architecture) for a specific problem situation.

Thus, the optimal synthesis problem is reduced to optimal choice of a solution from a pre-generated and constantly expanded set of solutions from the *configuration-file library* (CFL). This process can be formalized rather strictly. The approach proposed not only allows obtaining the optimal solution to the problem but also facilitates its formulation and interaction of the user with the computer system.

Let a sequence of algorithms be given. The CFL uses one library element to implement an algorithm, and a method/problem (M) is represented by a sequence of algorithms ($A_i \ \forall\, i = \overline{1, n}$,): $M = \bigcup\limits_i A_i$.

The base (zero) architecture of reconfigurable devices is implemented on EPLD as a functional processing field of constant dimension, the bus controller of the host computer, memory field, and a well-structured CFL of structural realizations of methods (algorithms) mapping an algorithm into a structural realization ($F : A_i \Rightarrow B_i$). A structural realization (B_i) is a configuration file for EPLD chip. There are several alternate algorithm realizations in the general case (for example, sequential, series-parallel, and parallel):

$$B_i = \bigcup_z B_{iz} \ \ (z = \overline{1, \eta}).$$

Each alternative is characterized by speed parameters (run time t_{iz}) and hardware scope (q_{iz}). If the required realization of the ith algorithm is absent in the library ($B_i = \varnothing$), it is necessary to use CAD EPLD tools to create it and incorporate into the library as a standard element. Thus, the optimization problem is reduced to an ordered assignment of a certain (B_{iz})th element of the library to each ith node of the graph of the algorithm, which results in a structure that realizes this graph.

In the general case, the optimization problem for an integral performance criterion can be presented as follows:

$$\alpha \sum_i \sum_z q_{iz} x_{iz} + \beta \sum_i \sum_z t_{iz} x_{iz} = \min \ \ (\forall i = \overline{1, n}, \ \ \forall z = \overline{1, \eta})$$

under the constraints

$$\sum_i \sum_z q_{iz} x_{iz} \le Q_0, \ \ \sum_i \sum_z t_{iz} x_{iz} \le T_0, \ \ \sum_{z=1}^{n} x_{iz} = 1, \ i = \overline{1, n}, \ \ z = \overline{1, \eta},$$

where α and β are weight coefficients, which can be determined, for example, by the judgment method; T_0 is the admissible run time for all algorithms; and Q_0 is admissible hardware scope. The problem can be solved by an appropriate method of integer mathematical programming [15].

CONCLUSIONS

We have proposed an approach to constructing a comparatively new class of computing systems — RCSs. The RCS architecture differs from the traditional Von Neumann architecture (universal for all algorithms) by the following principles:

— allows the developer (user) to create a custom structure for any arbitrary algorithm (problem being solved);

— the algorithm can be partitioned into sequentially run fragments, which saves hardware; the complexity of fragments is determined by the logical capacity of the FPGA chip.

The configuration files allow designers of algorithm-implementing hardware to make efficient use of functional blocks in developing modern computer aids.

It appears important to create RCSs oriented to the following:

— sophisticated specialized structures, including automata networks;
— ontology-controlled architectures dealing with knowledge systems;
— high-performance computing systems;
— hybrid computer systems combining procedure-algorithmic and network components;
— fail-safe control systems.

We have used a modified logical and information method for designing reconfigurable devices and systems. The method is oriented to the functional capabilities of EPLDs. It allows us to synthesize optimal structures represented by hierarchical systems with an unlimited number of information levels for the class of "speed–realization complexity" criteria.

The structure of a standard reconfigurable computing system with an open file-configuration library for base library blocks has been synthesized. It includes a number of functional blocks developed by the authors and verified on real stands.

REFERENCES

1. A. V. Palagin, "Development trends and issues of theoretical design of microprocessors and microcomputers," USiM, No. 3, 24–29 (1982).
2. A. V. Palagin and A. F. Kurgayev, "Problem orientation in developing computer architectures," Cybern. Syst. Analysis, **39**, No. 4, 615–625 (2003).
3. A. V. Palagin, E. L. Denisenko, R. I. Belitskii, and V. I. Sigalov, Microprocessor Data Handling Systems [in Russian], Naukova Dumka, Kyiv (1993).
4. V. N. Opanasenko and A. N. Lisovyi, "Peculiarities of the VHDL language for programming EPLD chips," Probl. Program., No. 1, 70–78 (2006).
5. A. V. Palagin, V. N. Opanasenko, and V. G. Sakharin, "Experience of designing EPLD-based digital devices using the HDL-technology," USiM, No. 6, 11–20 (2004).
6. A. V. Palagin, V. N. Opanasenko, and V. G. Sakharin, "Computing systems with reconfigurable (programmable) architecture," Probl. Informatizatsii ta Upravlinnya, Issue 10, 5–13 (2004).
7. A. V. Palagin, "A computer with virtual architecture," USiM, No. 3, 33–43 (1999).
8. A. V. Palagin and V. N. Opanasenko, Reconfigurable Computing Systems [in Russian], Prosvita, Kyiv (2006).
9. V. M. Opanasenko and I. G. Timoshenko, "Architecture organization of EPLD-based reconfigurable computers," in: Radioelectronics, Information Science, Control [in Ukrainian], No. 2 (12), ZNTU, Zaporizhzhya (2004), pp. 139–144.
10. O. V. Palagin, V. M. Opanasenko, and V. G. Sakharin, "Reconfigurable processor," Patent 15781 UA, MPK G06F15/00, No. 200600583; Applied for Jan. 23, 2006, Published July 17, 2006, Byul. No. 7 (2006).
11. C. Chang, J. Wawrzynek, and R. W. Brodersen, "BEE2: A high-end reconfigurable computing system," IEEE Design and Test of Comput., **22**, No. 2, 114–125 (2005).
12. V. N. Opanasenko, "Synthesis of a parametric module of a multilevel combinative logic circuit," Matem. Mashiny i Sistemy, No. 1, 2, 34–39 (2001).
13. V. M. Glushkov, Synthesis of Digital Automata [in Russian], Fizmatgiz, Moscow (1962).
14. Yu. V. Kapitonova and A. A. Letichevskii, The Mathematical Theory of Computing System Design [in Russian], Nauka, Gl. Red. Fiz.-Mat. Lit., Moscow (1988).
15. I. V. Sergienko and V. P. Shilo, Discrete Optimization Problems. Challenges, Solution Methods, and Research [in Russian], Naukova Dumka, Kyiv (2003).