A Study and Survey on Reconfigurable Computing Architecture using FPGAs

Singaravelan N[1], Ramachandran R[2]
[1, 2] Department of Electronics and Communication Engineering
[1, 2] RMK College of Engineering and technology, Puduvoyal, Tamilnadu

Abstract- *Reconfigurable computing has been driven largely by the development of commodity field-programmable gate arrays (FPGAs). Standard FPGAs are somewhat of a mixed blessing for this field. In this survey we give a brief overview of programming logics and we present configurable logic block (CLB) and Look Up Table (LUT) as logic elements. Also we presented the definition of fine and coarse-grain architectures and present some commercial examples. This survey is also introduced the reconfigurable computing models like static and dynamic, single and multi-context and partial reconfiguration architectures. Finally run-time reconfigurable computing and the coupling of reconfigurable processing unit (RUP) delineated.*

Keywords- Reconfigurable computing, ASIC, Field Programmable, LUT, CLB, FPGA, reconfigurable systems, Run-time reconfiguration, VHDL, RUT

I. INTRODUCTION

One of the most important research areas in computer hardware is reconfigurable computing and it is becoming gradually more attractive for many useful applications. Reconfiguration is the process of changing the structure of a reconfigurable device at star-uptime respectively at run-time [6] [11]. Reconfigurable computing involves the use of reconfigurable devices, such as Field Programmable Gate Arrays (FPGAs), for computing purposes. FPGAs are programmable microprocessors which end-user can configure them without any fabrication facilities. In most applications, FPGAs have all of Application Specific Integration Circuits (ASICs) benefits without more construction costs [1].

This paper is organized as follows: in Section II, we first give a brief overview of programmable logics, implementation spectrum and the differences. Section III presents programmable logic elements such as logic block based on look up table (LUT) and configurable logic block (CLB). In Section IV we define the fine-grain and coarse-grain architectures. Section V presents reconfigurable computing hardware devices such as SRAM, anti-fuse technology, EPROM, EEPROM and FLASH. In Section VI, FPGA structures and routing resources described and we analyze the famous Programming technology properties in summary. We present the overall definition of Run-time configuration in Section VIII. Finally, we draw the overall conclusions in section IX.

II. IMPLEMENTATION SPECTRUM

In the area of computer architecture, designers are faced with the trade-off between flexibility and performance [2]. The architectural choices span a wide spectrum, with general-purpose processors and application-specific integrated circuits (ASICs) at opposite ends. Fig. 1 presents a schematic overview of the speed/flexibility trade-off. The application specific integration circuit (ASIC) stands at the end of spectrum. It is designed for one specific task therefore there is no need for an instruction set. ASICs are dedicated hardware devices that are tuned to a very small number of applications or even to just one task. For a given task, ASICs achieve a higher performance, require less silicon area, and are less power consuming than instruction-level programmable processors. However, they lack in flexibility [2]. Thus ASICs gives high performance at cost of inflexibility. The general-purpose processor (GPP) can be found at another end of spectrum. A GPP is very flexible, as it can execute any function. Each function can be assembled by various instructions supported by the GPP. However, fetching instructions from memory and decoding them costs (a lot of) time Whereas GPP are very slow [3].

The best solution is to use programmable circuits such as PLD, CPLD and FPGAs. There are some reasons that encourage using programmable circuits. First, the implementations of complex digital functions are simple, second, test and analysis of circuits are very easy and fast. Third, inexpensive fabrications for few productions and the most important reason are the accordance with functional requirements and the ability to change the design over again by user.

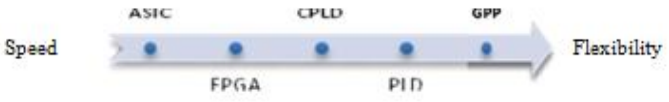

Fig.1. Speed/Flexibility comparison

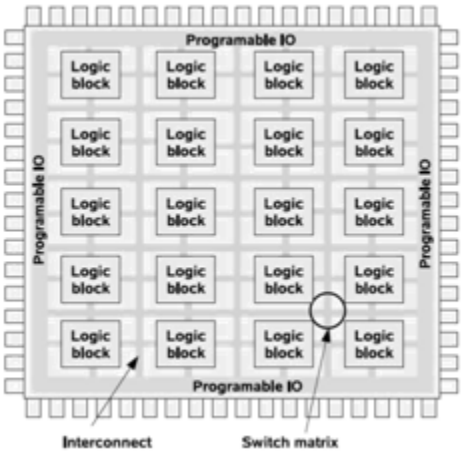

Fig.2. An abstract view of an FPGA

A. PLD, CPLDs and FPGAs Structures

Programmable logic devices (PLD) are integrated circuits with internal logic gates that are connected together through fuses. A process that is called programming defines the functionality of the chip.ROMs (Read Only Memories), PALs (Programmable array logic) and PLAs (Programmable Logic Arrays) are examples of PLDs. The main difference between these devices is the position of the fuses and the fixed connection between gates. Inside each PLD is a set of fully connected macrocells. These macrocells are typically comprised of some amount of combinatorial logic (AND and OR gates, for example) and a flip-flop. A PLA consists of two levels of logic gates: a programmable "wired" AND-plane followed by a programmable "wired" OR-plane. A PLA is structured. So that any of its inputs (or their complements) can be AND'ed together in the AND-plane [4].For large logic circuits, complex programmable logic devices (CPLD) can be used. A CPLD consists of a set interconnection network. The connection between the input/output blocks and the macro cells and those between macro cells and macro cells can be made through the programmable interconnection network [9] [11]. In practical view point, there are two differences between FPGA and CPLD;first, the FPGAs with thousand gate capacity have more facilities instead of CPLDs to design more complex and huge digital systems. Second, FPGAs use more programmable switches for FPGA's interconnection blocks that have more delay. Therefore FPGAs have more delays instead of CPLDs and PALs. Also FPGAs are implementable by using computer aided design (CAD) tools such as Very Log hardware design language (VHDL).

B. Field Programmable Gate Arrays

An FPGA is a programmable device consisting, like the CPLDs, of three main parts. A set of programmable logic cells also called logic blocks or configurable logic blocks, a programmable interconnection network and a set of input and output cells around the device.A function to be implemented in FPGA is partitioned in modules, each of which can be implemented in a logic block. The logic blocks are then connected together using the programmable interconnection.

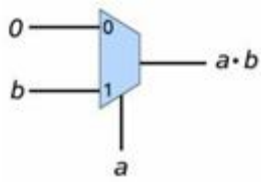

Fig. 3. Implementation of f=ab in a 2-input MUX

The user in the field can program all three basic components of an FPGA. FPGAs can be programmed once or several times depending on the technology used [11].Also FPGA is a general structure that allows very high logic capacity. Whereas CPLDs feature logic resources with a wide number of inputs (AND planes), FPGAs offer more narrow logic resources. FPGAs also offer a higher ratio of flip-flops to logic resources than do CPLDs. FPGAs comprise an array of uncommitted circuit elements, called logic blocks, and interconnect resources, but FPGA configuration is performed through programming by the end user. Fig. 3 illustrates the internal workings of a field-programmable gate array, which is made up of logic blocks embedded in a general routing structure. The logic blocks contain processing elements for performing simple combinational logic, as well as flip-flops for implementing sequential logic. Because the logic units are often just simple memories, any Boolean combinational function of perhaps five or six inputs can be implemented in each logic block. The general routing structure allows arbitrary wiring, so the logical elements can be connected in the desired manner. Because of this generality and flexibility, an FPGA can implement very complex circuits [5].

III. PROGRAMMABLE LOGIC ELEMENTS

Programmable logic cells used to implement Boolean equations with more inputs. The complication of each logiccell is depended on the number of functions, which could be implemented on it. The structure of programmable logic cell is based on multiplexer or look up table (LUT).Logic cells based on multiplexer(2n-to-1) multiplexer (MUX) is a selector circuit with 2n inputs and one output. Its function is to allow only one input line to be fed at the output. The line to be fed at the output can be selected using some selector inputs. To select one of the 2n possible inputs, n selectors lines are required (Fig. 3). An MUX can be used to implement a given function. A complex function can be implemented in many multiplexers connected together. The function is first broken down into small pieces. Each piece is then implemented on a

multiplexer [11]. The multiplexers will then be connected to build the given function.The devices manufactured by Actel are based on a structure similar to traditional gate arrays and are based on multiplexers. Fig. 4 illustrates the logic block in the Act 3 and shows that it comprises an AND and OR gate that are connected to a multiplexer based circuit block.

A. Logic block based on LUT

A look-up table (LUT) is a group of memory cells, which contain all the possible results of a given function for a given set of input values.The LUT can compute any function of n inputs by simply programming the lookup table with the truth table of the function we want to implement [5]. As shown in the Fig. 5 if we wanted to implement F=AB+C function with our 3-input LUT (often referred to as a 3-LUT),

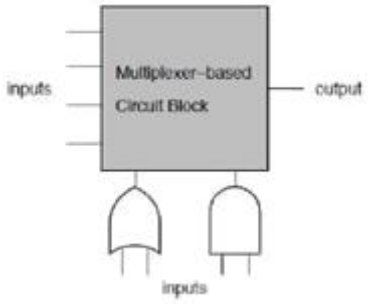

Fig. 4. The Actel basic computing blocks uses multiplexers as function generators

we would assign values to the lookup table memory such that the pattern of select bits chooses the correct row's "answer". The output values will be located in SRAM memory bits. Therefore complicated functions with a large number of inputs can be implemented by combining several lookup tables together.As Fig. 6 shown, in some applications we need to store the output bit of LUT, the D-Flip flop will attach at the end. A D flip-flop (DFF) is included to provide state-holding elements. This DFF can be bypassed when not needed, by selecting the appropriate multiplexer input behind the flip-flop [6].

B. Configurable logic block (CLB)

The basic computing block in the Xilinx FPGAs consists of an LUT with variable number of inputs, a set of multiplexers, arithmetic logic and a storage element (Fig. 7).The XC4000 features a logic block (called a Configurable Logic Block (CLB) by Xilinx) that is based onlook-up table (LUT) and has 2000 to 15000 gates.The Fig. 7 shows CLB design in traditional FPGA. The typical FPGA has a logic block with one or more 4-input LUT(s), optional D flip-flops (DFF), and some form of fast carry logic. The flip-flop can be used for pipelining, registers, state holding functions for finite state machines, or any other situation where clocking is required [6].The XC4000 interconnect is arranged in horizontal and vertical channels. Each channel contains some number of short wire segments that span a single CLB (the number of segments in each channel depends on the specific part number), longer segments that span two CLBs and very long segments that span the entire length or width of the chip [4].Except for the Virtex 5, all LUTs in Xilinx devices have four inputs and one output. In the Virtex 5 each LUT has six

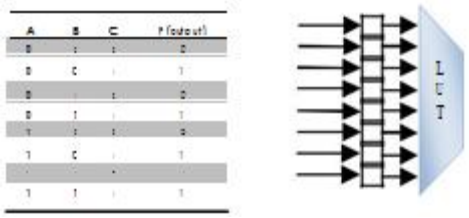

Fig. 5. a 3-LUT schematic

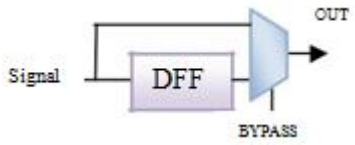

Fig. 6. D flip-flop with optional bypass

inputs and two outputs. The LUT can be configured either as a 6-input LUT, in which case only one output can be used, or as two 5-input LUTs, in which case each of the two outputs is used as output of a 5-input LUT [11].

IV. GRANULARITY

The logic block is defined by its internal structure and granularity. The structure defines the different kinds of logic that can be implemented in the block, while the granularity defines the maximum word length of the implemented functions. The functionality of the logic block is obtained by controlling the connectivity of some basic logic gates or by using LUTs and has a direct impact on the routing resources. As the functional capability increases, the amount of logic that can be packed into it increases.A collection of CLBs, known as logic cluster, is described with the following four parameters [8]; the size of (number of inputs to) a LUT, the number of CLBs in a cluster, the number of inputs to the cluster for use as inputs by the LUTs and the number of clock inputs to a cluster (for use by the registers).

Thus, the size and complexity of the basic computing blocks is referred to as the block granularity. All the reconfigurable platforms based on their granularity are distinguished into two groups, the fine-grain and coarse-grain

systems [2] [8].A number of reconfigurable systems use a granularity of logic block that we categorize as medium-grained.Medium-grained logic blocks may be used to implement datapath circuits of varying bit widths, similar to the fine-grained structures. However, with the ability to perform more complex operations of a greater number of inputs, this type of structure can be used efficiently to implement a wider variety of operations [6].

A. Fine-grain & Coarse-grain

In fine-grained architectures, the basic programmed building block consists of a combinatorial network and a few flip-flops. A fine-grain array has many configuration points to perform very small computations, and thus requires more data bits during configuration.The fine-grain programmability is more amenable to control functions, while the coarser grain blocks with arithmetic capability are more useful for data path operations. This type of logic block is useful for an encryption and image processing applications. The GRAP [Hauser and Wawrzynek 1997] and Chimaera [Hauck et al. 1997] are two

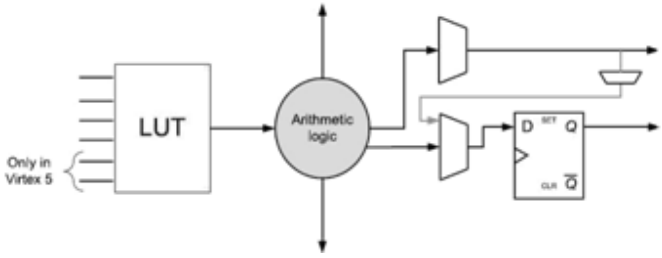

Fig. 7. a basic logic block of Xilinx FPGAs [11].

Platforms that are based on Fine-Grain Reconfigurable Devices. The CHESS [Marshall et al. 1999], RaPiD architecture [Ebeling et al. 1996] and The RAW project [Moritz et al. 1998] are some example of coarse-grained reconfigurable platform [6].

V. RECONFIGURABLE COMPUTING HARDWARE DEVICE ARCHITECTURE

Programming points, which may be based on anti-fuse, Flash, or SRAM technology, controls the logic and routing elements in reconfigurable computing hardware. In this section we discuss these technologies and specify the similarities and contrasts.

A. SRAM Technology:

The major advantage of this technology is that FPGAs can be programmed (configured) indefinitely. We just need to change the value into the SRAM cells to realize a new connection or a new function. Moreover, the device can be done in-circuit very quickly and allow the reconfiguration to be done on the fly [6] [11].

B. Anti-fuse technology

Anti-fuse is a programmable chip technology that creates permanent, conductive paths between transistors. In contrast to "blowing fuses" in the fusible link method, which opens a circuit by breaking apart a conductive path, the anti-fuse method closes the circuit by "growing" a conductive via. Anti-fuses are suitable for FPGAs because they can be built using modified CMOS technology [4].

C. EPROM, EEPROM, and FLASH

This class of non-volatile programming technology uses the same techniques as EPROM, EEPROM and Flash memory technologies. An EEPROM or EPROM transistor is used as a programmable switch for CPLDs by placing the transistor between two wires in a way that facilitates implementation of wired-AND functions.

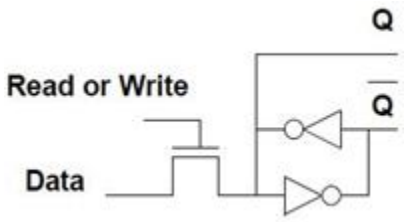

Fig. 8. a programming bit for SRAM-based FPGAs

Table 1- Programming technology properties summary

	SRAM	Flash	Anti-fuse
Volatile?	Yes	No	No
Reprogrammable?	Yes	Yes	No
Area (storage element size)	High	Moderate	Low
Manufacturing process?	Standard CMOS	Flash process	Anti-fuse need special development
In system programmable?	Yes	Yes	No
Switch capacitance	~1-2fF	~1-2fF	< 1fF
Switch resistance	~500-1000 Ω	~500-1000 Ω	~20-100 Ω

Usually the architecture also uses some general routing resources to realize longer connections [8].Like in symmetrical arrays, the macro cells are arranged on a two-dimensional array structure such that an entry in the array correspond to the coordinate of a given macro cell. The difference between the symmetrical array and the sea-of-gate is that there is no space left aside between the macro cells for routing. In hierarchical based FPGAs, macro cells are hierarchically placed on the device. Elements with the lowest granularity are at the lowest level hierarchy. They are grouped to form the elements of the next level [10] [11].

Most current FPGAs are of the two-dimensional variety. This allows for a great deal of flexibility, as any signal can be routed on a nearly arbitrary path. However, providing this level of routing, flexibility requires a great deal of routing area. It also complicates the placement and routing software, as the software must consider a very large number of possibilities. One solution is to use a more one-dimensional style of architecture. Here placement is restricted along one axis. With a more limited set of choices, the placement can be performed much more quickly and Routing is also simplified. One drawback of one-dimensional routing is that if there are not enough routing resources for a specific area of a mapped circuit, and then the routing of the whole circuit becomes actually more difficult than on a two-dimensional array that provides more alternatives [6] [8] [13].

VI. RECONFIGURATION MODELS

This section presents recon figuration models that are as follows: static and dynamic reconfiguration, single context and multiple context architectures that will be described as follows:

Fig. 11.a) Principle of static reconfiguration b) Principle of dynamic reconfiguration.

A. Static and dynamic reconfiguration

Static reconfiguration (often referred as compile time reconfiguration) is the simplest and most common approach for implementing applications with reconfigurable logic. Static reconfiguration involves hardware changes at a relatively slow rate. It is a static implementation strategy where each application consists of one configuration. The distinctive feature of this configuration is that it consists of a single system-wide configuration. Prior to commencing an operation, the reconfigurable resources are loaded with their respective configurations. Once operation commences, the reconfigurable resources will remain in this configuration throughout the operation of the application. Thus hardware resources remain static for the life of the design whereas static reconfiguration allocates logic for the duration of an application, dynamic reconfiguration (often referred to as run time reconfiguration) uses a dynamic allocation scheme that re-allocates hardware at run-time. This is an advanced technique that some people regard as a flexible realization of the time/space trade-off. It can increase system performance by using highly optimized circuits that are loaded and unloaded dynamically during the operation of the system as depicted in Fig. 11-b.In this way system flexibility is maintained and functional density is increased [14] [15].The dynamic reconfiguration has two main design problems. The first is to divide the algorithm into time-exclusive segments that do not need to run concurrently. The second problem is to co-ordinate the behavior between different configurations, i.e. the management of transmission of intermediate results from one configuration to the next [16].

B. Single context and Multi-context architecture

Although single context architectures can typically be reconfigured only statically, a run-time reconfiguration onto single context FPGA can also be implemented. In this type of FPGA, configuration information is loaded into the programmable array through a serial shift chain. Many commercial FPGAs are of this style, including the Xilinx 4000 series [7], the Altera Flex10K series, and Lucent's Orca series.

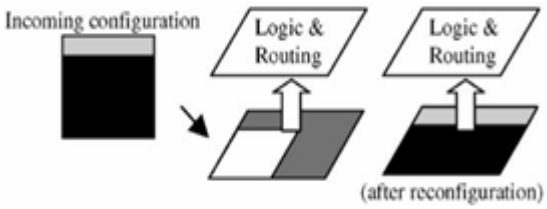

Fig. 12. single context dynamically reconfigurable architecture

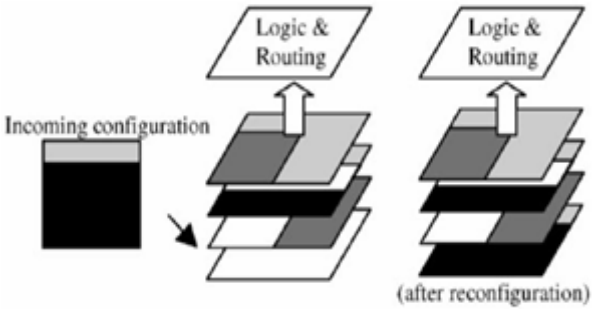

Fig. 13- multi-context dynamically reconfigurable architecture

This type of FPGA is therefore more suited for applications that can benefit from reconfigurable computing without run-time reconfiguration. A single context FPGA is depicted in Fig. 12. A multi-context architecture includes multiple memory bits for each programming bit location. These memory bits can be thought of as multiple planes of configuration information [3, 15]. Only one plane of configuration information can be active at a given moment, but the architecture can quickly switch between different planes, or contexts, of already-programmed configurations. Multi-context devices have two main benefits over single-context devices. First, they permit background loading of configuration data during circuit operation, overlapping computation with reconfiguration. Second, they can switch between stored configurations quickly-some in a single clock cycle-dramatically reducing reconfiguration overhead if the next configuration is present in one of the alternate contexts [14] [17] [18].

C. Partial reconfiguration architecture

In some cases, configurations do not occupy the full reconfigurable hardware, or only a part of a configuration requires modification. In both of these situations a partial reconfiguration of the reconfigurable resources is desired, rather than the full reconfiguration supported by the serial architectures mentioned above. When configurations do not require the entire area available within the array, a number of different configurations may be loaded into otherwise unused areas of the hardware (fig. 14). Partially runtime reconfigurable architectures can allow for complete reconfiguration flexibility such as the Xilinx 6200 [18], or may require a full column of configuration information to be reconfigured at once, as in the Xilinx Virtex FPGA [19]. For example, in a filtering operation in signal processing, a set of constant values that change slowly over time may be reinitialized to a new value, yet the overall computation in the circuit remains static [6].

VII. RUN-TIME RECONFIGURATION

Run-time reconfiguration is defined as the ability to modify or change the functional configuration of the device during operation, through either hardware or software changes. This concept is known as runtime reconfiguration (RTR). Run-time reconfiguration is based upon the concept of virtual hardware, which is similar to virtual memory. Here, the physical hardware is much smaller than the sum of the resources required by each of the configurations. Therefore, instead of reducing the number of configurations that are mapped, we instead swap them in and out of the actual hardware as they are needed [6]. The primary advantages of run-time reconfiguration in devices are reduced power consumption,

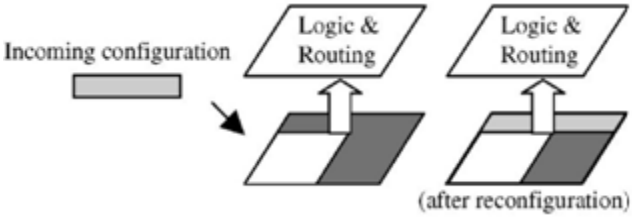

Fig. 14- Partial dynamically reconfigurable architecture

hardware reuse, obsolescence avoidance, and flexibility. The costs of run-time reconfigurability are in design and implementation complexity-both in architecture definition and in coding and test.

VIII. CONCLUSION

Research in architecture of computer systems has always been a central preoccupation of the computer science and computer engineering communities. This survey presented the basic technology that serves as the foundation of reconfigurable computing also we introduced the basic idea of lookup table computation, explained the need for dedicated computational blocks, and described common interconnection strategies. Finally, we tied it all together with brief overviews of two popular commercial architectures. Understanding the concept presented here will therefore help to understand better and faster the changes that will be made on the devices in the future.

REFERENCES

[1] I. Kuon and J. Rose, "Measuring the Gap Between FPGAs and ASICs", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), Vol. 26, No. 2, p.p 203-215, Feb. 2007.

[2] M. Platzner. “Reconfigurable Computer Architectures,” Stanford University, 1998.

[3] Thijs van As and S. Krijgsman, “Reconfigurable Architectures: A survey of design and Implementation Methods,” Faculty of Electrical Engineering, Mathematics & Computer Science Delft University of Technology.

[4] S. Brown and Jonathan Rose, “Architecture of FPGAs and CPLDs: A Tutorial”, IEEE Design & Test, Volume 13, Issue 2, Page 42-57, 1996

[5] S. Hauck andAndr´eDeHon “Reconfigurable computing: the theory and Practice of FPGA-based Computation,”

[6] K. Compton and S. Hauck. Reconfigurable Computing: A Survey of Systems and Software. ACM Computing Surveys, 34(2):171– 210, June 2002.

[7] “Xilinx,” http://www.xilinx.com.

[8] S. Vassiliadis and D. Soudris (eds.), “Fine- and Coarse-Grain Reconfigurable computing”.

[9] E.Mirsky and Andr´e DeHon. MATRIX: “A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources,” FCCM’96-IEEE Symposium on FPGAs for Custom Computing Machines April 17-19, 1996.

[10] I. Kuon, R. Tessier and J. Rose, “FPGA Architecture: Survey and Challenges,” Foundation and Trends in Electronic Design Automation, Vol. 2, no. 2, pp. 135-253, 2007.

[11] C. Bobda, “Introduction to Reconfigurable Computing: Architectures, Algorithms, and Applications”, Springer, 2007.

[12] Clive. “Max”. Maxfield, “ The Design warrior’s Guide to FPGA,” ISBN 0-7506-7604-3, ELSEVIER Publication, 2004

[13] S. Hauck,“The roles of FPGAs in Reprogrammable Systems”, in Proc. IEEE 86, 4, pp. 615–638, 1998.

[14] N. S. VOROS and K. MASSELOS “System Level Design of Reconfigurable Systems-on-Chip,” ISBN-10 0-387-26103-6, Springer, 2005.

[15] Khatib J (2001) Configurable Computing (http://www.geocities.com/siliconvalley/pines/6639/fpga).

[16] Hutchings BL, Wirthlin MJ “Implementation Approaches for Reconfigurable Logic Applications,” Brigham Young University, Dept. of Electrical and Computer Engineering. 1995

[17] DeHon A “DPGA Utilization and Application”, In: Proceedings of ACM/SIGDA International Symposium on FPGAs, pp 115-121, 1996

[18] S. Trimberger and D. Carberryand A. Johnson and J. Wong “A Time-Multiplexed FPGA”, In: Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines,pp 22-29, 1997.

[19] Xilinx Inc. (1999) VirtexTM: Configuration Architecture Advanced Users’ Guide

[20] T. Miyamori and K. Olukotun, “A Quantitative Analysis of Reconfigurable Coprocessors for Multimedia Applications,” In: Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines, pp 2-11, 1998.

[21] RD. Witting and P. Chow, “OneChip: An FPGA Processor with Reconfigurable Logic,”. In: Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines, pp 126-135, 1996.

[22] R. Razdan and K. Brace andMD. Smith,“PRISC Software Acceleration Techniques”, Proceedings of the IEEE International Conference on Computer Design, pp 145-149, 1994.

[23] S. Hauck and TW. Fry andMM. HoslerandJP. Kao, “The Chimaera Reconfigurable Functional Unit”, Proceedings of the 5th IEEE Symposium on Field Programmable Custom Computing Machines, pp 87-96, 1997.