Chapter 1

INTRODUCTION TO RECONFIGURABLE HARDWARE

Konstantinos Masselos[1,2] and Nikolaos S. Voros[1]
[1] *INTRACOM S.A., Hellenic Telecommunications and Electronics Industry, Greece*
[2] *Currently with Imperial College of Science Technology and Medicine, United Kingdom*

Abstract: This chapter introduces the reader to main concepts of reconfigurable computing and reconfigurable hardware. Different types of reconfiguration are discussed. A detailed classification of reconfigurable architectures with respect to the granularity of their building blocks, the reconfiguration scheme and the system level coupling is also presented.

Key words: Reconfigurable hardware, reconfigurable architectures, reconfiguration, reconfigurable computing

1. RECONFIGURABLE COMPUTING AND RECONFIGURABLE HARDWARE

Reconfigurable computing refers to systems incorporating some form of hardware programmability–customizing how the hardware is used using a number of physical control points [2]. These control points can then be changed periodically in order to execute different applications using the same hardware. Reconfigurable hardware offers a good balance between implementation efficiency and flexibility as shown in Figure 1-1. This is because reconfigurable hardware combines post-fabrication programmability with the spatial (parallel) computation style [2] of application specific integrated circuits (ASICs), which is more efficient in comparison to the temporal (sequential) computation style of instruction set processors.

Due to the increasing flexibility requirements (e.g. for adaptation to different evolving standards and operating conditions) that are imposed by computationally intensive applications such as wireless communications,

N.S. Voros and K. Masselos (eds.), System Level Design of Reconfigurable Systems-on-Chips, 15-26.

devices need to be highly adaptable to the running applications. On the other hand, efficient realizations of such applications are required, especially in the resources they use during deployment, where power consumption must be traded against perceived quality of the application. The contradictory requirements for flexibility and implementation efficiency cannot be satisfied by conventional instruction set processors and ASICs. Reconfigurable hardware forms an interesting implementation option in such cases.

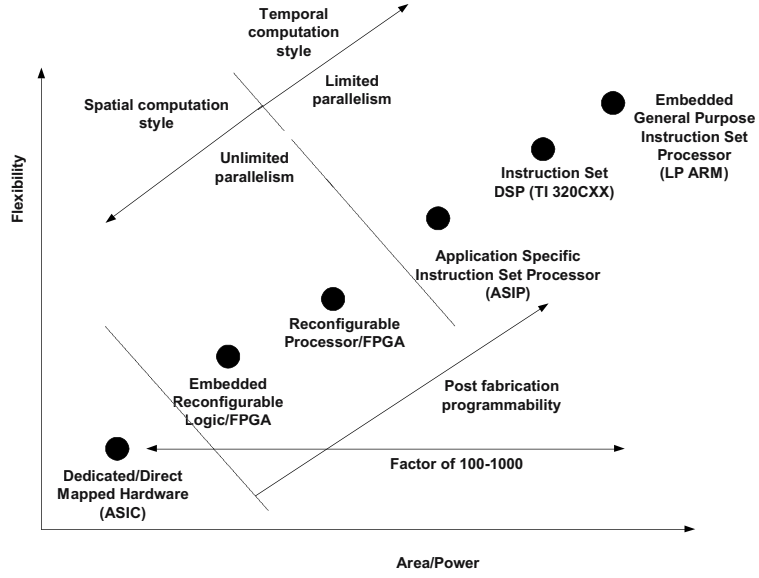

Figure 1-1. Positioning of reconfigurable hardware

There are also other reasons why to use reconfigurable resources in system-on-chip (SoC) design. The increasing non-recurring engineering (NRE) costs push designers to use same SoC in several applications and products for achieving low cost per chip. The presence of reconfigurable resources allows the fine tuning of the chip for different products or product variations. Also, the increasing complexity in the future designs adds the possibility of including design flows, which can require costly and slow redesign of the chip. Reconfigurable elements are often homogenous arrays, which can be pre-verified to minimize the possibility of having design errors. Also the post-manufacturing programmability allows correction or circumvention of problems later than with fixed hardware.

2. TYPES OF RECONFIGURATION

The next paragraphs describe different types of reconfiguration.

2.1 Logic reconfiguration

A typical logic block reconfigurable architecture contains a look-up table (LUT), an optional D flip-flop and additional combinational logic. The LUT allows any function to be implemented, providing generic 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. The combinatorial logic is usually the fast carry logic used to speed up fast carry-based computations such as addition, parity, wide AND operations and other functions. The logic blocks located at the periphery of the device can be of different architecture dedicated to I/O operations.

The logic blocks are grouped to matrices overlaid with a reconfigurable interconnection network of wires. Interconnection network reconfiguration is controlled by changing the connections between the logic blocks and the wires and by configuring the switch boxes, which connect different wires. The reconfiguration of both the logic blocks and the interconnection network is achieved by using SRAM memory bits to control the configuration of transistors. The functionality of the logic blocks, I/O blocks and the interconnection network is modified by downloading bit stream of reconfiguration data onto the hardware.

2.2 Instruction-set reconfiguration

The concept of instruction-set reconfiguration refers to the hybrid architectures consisting of microprocessor and reconfigurable logic. The key benefit is a combination of full software flexibility with high hardware efficiency. One promising approach is the reconfigurable instruction set processors (RISP), which have the capability to adapt their instruction sets to the application being executed through a reconfiguration in their hardware. The result is a reconfigurable and extensible processor architecture, which could be tailored closely to the designers' specific needs.

Through the adaptation, specialized hardware accelerates the execution of the applications. If shared resources are used in the adaptation, the functional density is also improved. By moving the application-specific data-paths into the processor, a remarkable improvement in performance compared to fixed instruction-set processors can be achieved. At the same time, designing at the level of instruction-set architecture significantly shortens the design cycle and reduces verification effort and risk. On the

other hand, new methodologies, tools and processor foundations are required. Automated extension of processor function units and associated software environment - compilers, debuggers, instruction simulators etc., are also the key points to success.

Different systems with different characteristics have been designed. Usually two main design tasks are involved:

1. What is the type of interfaces between the microprocessor and the reconfigurable logic?
2. How to design the reconfigurable logic itself?

Each of them contains many trade-offs. The common classification of the reconfigurable processors could be made according to the coupling levels of reconfigurable logic. The concept of coupling levels applies also without a reference to reconfigurable processors. As shown in Figure 1-2, there are three types of coupling levels:

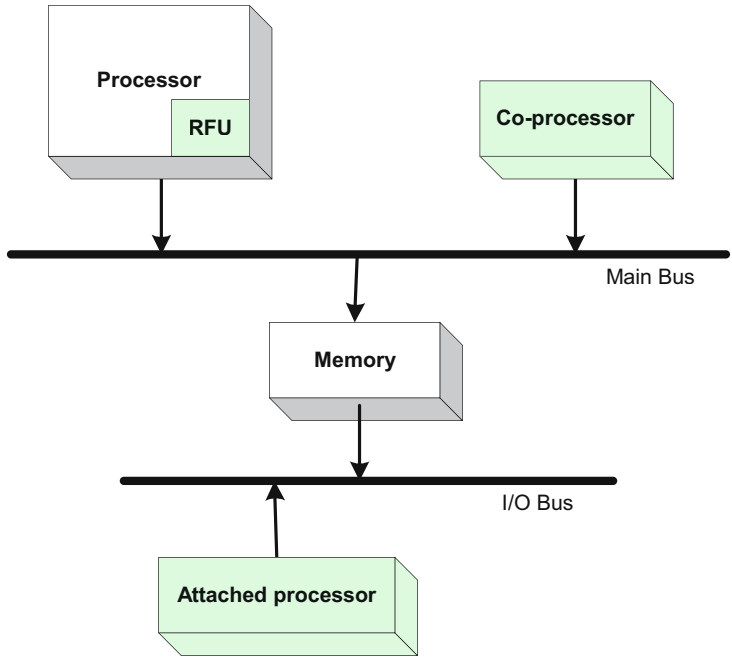

Figure 1-2. Basic coupling levels of reconfigurable logic

1. *Reconfigurable functional unit (RFU)* - the logic is placed inside the processor, the instruction decoder issues instructions to the reconfigurable unit as if it were one of the standard functional units of the processor. In this way, the communication cost is very small, so the speed could be easily increased. This is also the most promising

approach because it can be used to accelerate almost any application [1].

2. *Coprocessor* - the logic is next to the processor. Communication is done using a protocol.
3. *Attached processor* - the logic is placed on some kind of I/O bus. With the coprocessor and attached processor approaches, the speed improvement using the reconfigurable logic has to compensate for the overhead of transferring the data. This usually happens in applications where a huge amount of data has to be processed using a simple algorithm that fits in the reconfigurable logic.

2.3 Static and dynamic reconfiguration

There are two basic reconfiguration approaches: *static* and *dynamic*.

2.3.1 Static 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 main objective is to improve the performance.

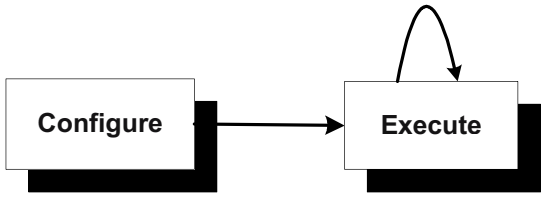

Figure 1-3. Principle of static reconfiguration

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 (or application). This is depicted in Figure 1-3. Much higher performance than with pure software implementation (e.g. microprocessor approaches), cost advantage over

ASICs in certain cases and conventional CAD tool support are the main advantages of this technology.

2.3.2 Dynamic reconfiguration

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 Figure 1-4. In this way system flexibility is maintained and functional density is increased [9].

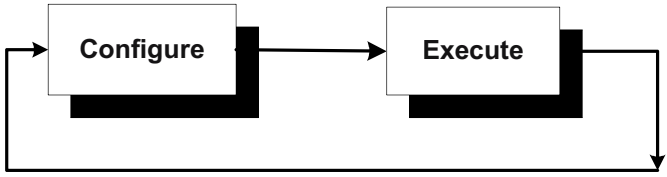

Figure 1-4. Principle of dynamic reconfiguration

Dynamic reconfiguration is based upon the concept of virtual hardware, which is similar to the idea of virtual memory. Here, the physical hardware is much smaller than the sum of the resources required by all 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.

There are two main design problems for this approach: the first is to divide the algorithm into time-exclusive segments that do not need to (or cannot) run concurrently. This is referred to as temporal partitioning. Because no CAD tools support this step, this requires tedious and error-prone user involvement. The second problem is to co-ordinate the behaviour between different configurations, i.e. the management of transmission of intermediate results from one configuration to the next [8].

3. CLASSIFICATION OF RECONFIGURABLE ARCHITECTURES

In this section reconfigurable hardware architectures are classified with respect to several parameters. These parameters are described below:

- **Granularity of building blocks** This refers to the levels of manipulation of data. In this chapter we distinguish three types of granularity: *fine-grain* which corresponds to bit-level manipulation of data, *medium grain* manipulating data with varying number of bits and *coarse-grain* granularity which implies word level operations.
- **Reconfiguration scheme** Systems can be reconfigured statically or dynamically. Dynamically reconfigurable systems permit the partial reconfiguration of certain logic blocks while others are performing computations. Statically reconfigurable devices require execution interrupt.
- **Coupling** This refers to the degree of coupling with a host microprocessor. In a *closely coupled* system reconfigurable units are placed on the data path of the processor, acting as execution units. *Loosely coupled* systems act as a coprocessor. They are connected to a host computer system through channels or some special-purpose hardware.

3.1 Classification with respect to building blocks granularity

The granularity criterion reflects the smallest block of which a reconfigurable device is made.

In *fine-grained* architectures, the basic programmed building block usually consists of a combinatorial network and a few flip-flops. The logic block can be programmed into a simple logic function, such as a 2-bit adder. These blocks are connected through a reconfigurable interconnection network. More complex operations can be constructed by reconfiguring this network. Commercially available Field Programmable Gate Arrays (FPGAs) are based on fine grain architectures.

Although highly flexible, these systems exhibit a low efficiency when it comes to more specific tasks. For example, although an 8-bit adder can be implemented in a fine-grained circuit, it will be inefficient, compared to a reconfigurable array of 8-bit adders, when performing an addition-intensive task. An 8-bit adder will also occupy more space in the fine-grained implementation.

Reconfigurable systems which use logic blocks of larger granularity are categorized as *medium-grained* [6, 7, 10, 11, 17]. For example, Garp [6] is designed to perform a number of different operations on up to four 2-bit inputs. Another medium-grained structure was designed specifically to implement multipliers of a configurable bit-width [7]. The logic block used in the multiplier FPGA is capable of implementing a 4x4 multiplication. The CHESS architecture [11] also operates on 4-bit values, with each of its cells acting as a 4-bit ALU. The major advantage of medium-grained systems when compared to the fine-grained architecture is, that they better utilize the chip area, since they are optimized for the specific operations. However, a drawback of this approach is represented in a high overhead when synthesizing operations which are incompatible with the simplest logic block architecture.

Coarse-grained architectures are primarily intended for the implementation of tasks dominated by word-width operations. Because the logic blocks used are optimized for large computations, they will perform these operations much more quickly (and consume less chip area) than a set of smaller cells connected to form the same type of structure. However, because their composition is static, they are unable to leverage optimizations in the size of operands. On the other hand, these coarse-grained architectures can be much more efficient than finer-grained architectures for implementing functions closer to their basic word size. An example of coarse-grained system is the RaPiD architecture [4].

A *very coarse granularity* is the case when the simplest logic block is based on an entire microprocessor with or without special accelerators. Examples of such architectures are the REMARC [12] and RAW [13] architectures.

3.2 Classification with respect to reconfiguration scheme

3.2.1 Statically reconfigurable architectures

Traditional reconfigurable architectures are *statically reconfigurable*, which means that the reconfigurable resources are configured at the start of execution and remain unchanged for the duration of the application. In order to reconfigure a statically reconfigurable architecture, the system has to be halted while the reconfiguration is in progress and then restarted with the new configuration.

Traditional FPGA architectures have primarily been serially programmed single-context devices, allowing only one configuration to be loaded at a time. This type of FPGAs is programmed using a serial stream of

configuration information, requiring a full reconfiguration if any change is required.

3.2.2 Dynamically reconfigurable architectures

Dynamically reconfigurable (run-time reconfigurable) architectures allow reconfiguration and execution to proceed at the same time. The different reconfigurable styles of dynamic reconfiguration are depicted in Figure 1-5 and discussed in the following paragraphs.

Single context dynamically reconfigurable architectures

Although single context architectures can typically be reconfigured only statically, a run-time reconfiguration onto single context FPGA can also be implemented. Typically, the configurations are grouped into contexts, and each context is swapped as needed. Attention has to be paid on proper partitioning of the configurations between the contexts in order to minimize the reconfiguration delay.

Multi-context dynamically reconfigurable architectures

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

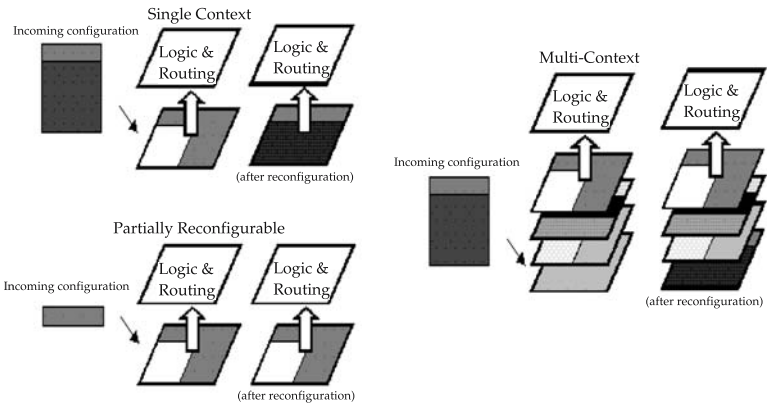

Figure 1-5. The different basic models of dynamically reconfigurable computing

quickly switch between different planes, or contexts, of already-programmed configurations. In this manner, the multi-context architecture can be considered a multiplexed set of single-context architectures, which requires that a context be fully reprogrammed to perform any modification to the configuration data. However, this requires a great deal more area than the other structures, given that there must be as many storage units per programming location as there are contexts. This also means that multi-context schemes are mainly used in coarse-grain architectures.

Partially Reconfigurable Architectures

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.

In partially reconfigurable architectures, the underlying programming layer operates like a RAM device. Using addresses to specify the target location of the configuration data allows for selective reconfiguration of the reconfigurable resources. Frequently, the undisturbed portions of the reconfigurable resources may continue execution, allowing the overlap of computation with reconfiguration. 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. Partially run-time 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].

4. COUPLING

The type of coupling of the Reconfigurable Processing Unit (RPU) to the computing system has a big impact on the communication cost. It can be classified into one of the four groups listed below, which are presented in order of decreasing communication costs and illustrated in Figure 1-6:

- RPUs coupled to the I/O bus of the host (Figure 1-6.a). This group includes many commercial circuit boards. Some of them are connected to the PCI bus of a PC or workstation.
- RPUs coupled to the local bus of the host (Figure 1-6.b).

- RPUs coupled like co-processors (Figure 1-6.c) such as the REMARC - Reconfigurable Multimedia Array Coprocessor [12].
- RPUs acting like an extended data-path of the processor (Figure 1-6.d) such as the OneChip [16], the PRISC - Programmable Reduced Instruction Set Computer [14], and the Chimaera [5].

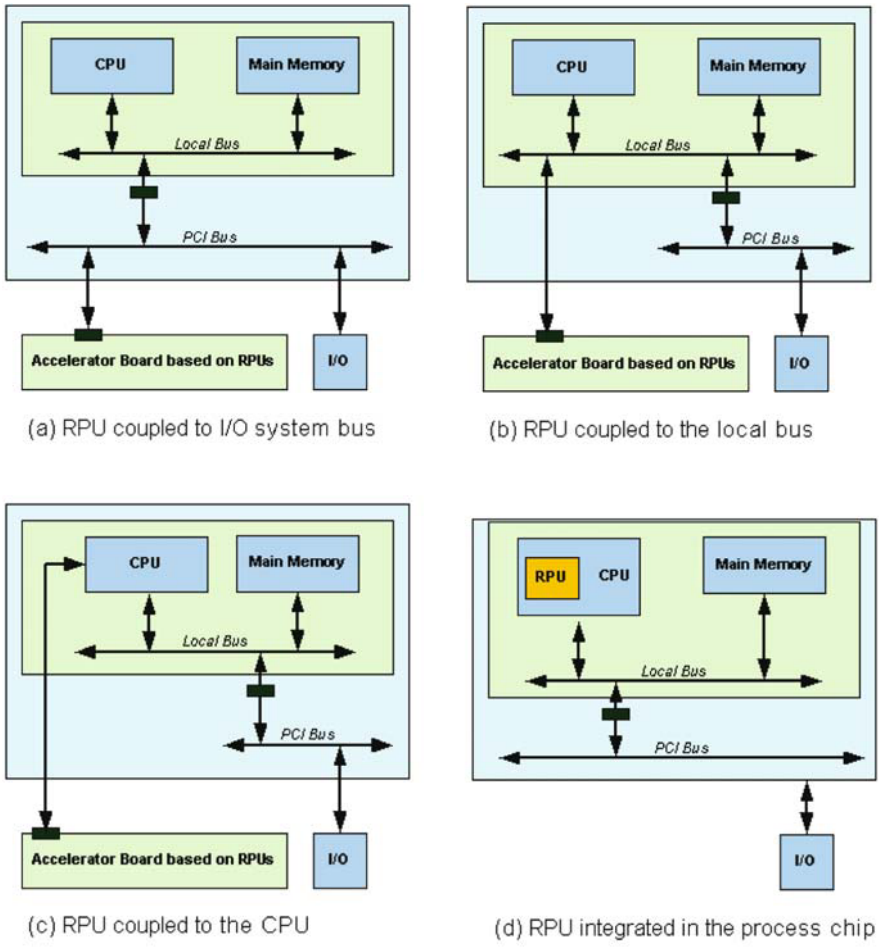

Figure 1-6. Coupling of the RPU to the host computer

REFERENCES

1. Barat F, Lauwereins R (2000) Reconfigurable Instruction Set Processors: A Survey. In: Proceedings of IEEE international Workshop on Rapid System Prototyping, pp 168-173

2. Brodersen B (2002) Wireless Systems-on-a-Chip Design. In: Proceedings of 3rd International Symposium on Quality of Electronic Design, pp 221-222
3. DeHon A (1996) DPGA Utilization and Application. In: Proceedings of ACM/SIGDA International Symposium on FPGAs, pp 115-121
4. Ebeling C, Cronquist DC, Franklin P (1996) RaPiD Reconfigurable Pipelined Datapath. In: Lecture Notes in Computer Science 1142 – Field Programmable Logic: Smart Applications, New Paradigms and Compilers, Springer Verlag, pp 126-135
5. Hauck S, Fry TW, Hosler MM, Kao JP (1997) The Chimaera Reconfigurable Functional Unit. In: Proceedings of the 5th IEEE Symposium on Field Programmable Custom Computing Machines, pp 87-96
6. Hauser JR, Wawrzynek J (1997) Garp: A MIPS Processor with a Reconfigurable Coprocessor. In: Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines, pp 12-21
7. Haynes SD, Cheung PYK (1998) A reconfigurable multiplier array for video image processing tasks, suitable for embedding in an FPGA structure. In: Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines, pp 226-235
8. Hutchings BL, Wirthlin MJ (1995) Implementation approaches for reconfigurable logic applications. Brigham Young University, Dept. of Electrical and Computer Engineering
9. Khatib J (2001) Configurable Computing. Available at: http://www.geocities.com/siliconvalley/pines/6639/fpga
10. Lucent Technologies Inc (1998) FPGA Data Book, Allentown, Pennsylvania
11. Marshall A, Stansfield T, Kostarnov I, Vuillemin J, Hutchings B (1999) A Reconfigurable Arithmetic Array for Multimedia Applications. In: Proceedings of ACM/SIGDA International Symposium on FPGAs, pp 135-143
12. Miyamori T, Olukotun K (1998) A quantitative analysis of reconfigurable coprocessors for multimedia applications. In: Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines, pp 2-11
13. Moritz CA, Yeung D, Agarwal A (1998) Exploring optimal cost performance designs for raw microprocessors. In: Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines, pp 12-27
14. Razdan R, Brace K, Smith MD (1994) PRISC Software Acceleration Techniques. In: Proceedings of the IEEE International Conference on Computer Design, pp 145-149
15. Trimberger S, Carberry D, Johnson A, Wong J (1997) A Time-Multiplexed FPGA. In: Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines, pp 22-29
16. Witting RD, Chow P (1996) OneChip: An FPGA Processor with Reconfigurable Logic. In: Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines, pp 126-135
17. Xilinx Inc. (1994) The Programmable Logic Data Book
18. Xilinx Inc. (1996) XC6200: Advanced product specification v1.0. In: The Programmable Logic Data Book
19. Xilinx Inc. (1999) VirtexTM: Configuration Architecture Advanced Users' Guide