A configuration concept for a massively parallel FPGA architecture

Sandeep Kumar[1], Christof Paar[1], Jan Pelzl[1], Gerd Pfeiffer[2], Manfred Schimmler[2]

[1] Horst Görtz Institute for IT Security, Ruhr University Bochum, Germany

{kumar,cpaar,pelzl}@crypto.rub.de

[2] Institute of Computer Science and Applied Mathematics, Faculty of Engineering, Christian-Albrechts-University of Kiel, Germany

{gp,masch}@informatik.uni-kiel.de

Abstract – Today's computer hardware are highly optimized general purpose architectures but unavoidable inefficient in terms of special purpose scenarios. The grade of efficiency usually has to face a business economical consideration to choose between special versus general purpose hardware. The emerging technology of programmable logic devices enables a new kind of architecture for massively parallel systems based on programmable logic devices. For such systems the traditional way of chip configuration is not useful anymore. This paper presents a concept for configuring massively parallel reconfigurable architectures. Furthermore a proof of concept is given by applying the configuration scheme to an reconfigurable architecture consisting of 120 chips.

Keywords: parallel hardware, cost-optimized architecture, FPGA configuration

1 Introduction

While in most cases standard hardware comes into operation, there always have been few but important problems demanding application specific hardware. The latter may be applied in the domain of Printed Circuit Boards (PCB) or in the chip domain as Application Specific Integrated Circuit (ASIC) or both of them. Since a couple of years, there is a new additional alternative: the Field Programmable Gate Array (FPGA) which is a reconfigurable logic device that has become standard hardware on the one hand. On the other hand the architectural freedom is saved. Hence FPGAs are well suited as building block for special purpose hardware in the field of compute intensive applications. In analogy to standard computer clusters parallel algorithms preferably run on parallel architectures.

The idea to use massively parallel FPGA architectures for solving compute intensive applications requires a technology supporting an unrestricted number of reconfiguration cycles, because every implemented application leads to a unique configuration. High density SRAM based architectures are supporting this feature. Also they are a proper choice in terms of computational performance. Such SRAM based FPGAs are without configuration after power on. Due to this fact a massively parallel system composed of this kind of chips has to

be configured after power on every time. For single and medium scale FPGA architectures the configuration may be taken for granted. Massively parallel FPGA architectures are characterized by a high device count demanding an appropriate configuration concept. Section 2 addresses the extent of the problem while section 3 addresses a brief introduction to a recent massively parallel FPGA architecture where the configuration concept was implemented. A conclusion in section 4 outlines the general problem and the here presented solution for configuring massively parallel FPGA architectures.

2 The problem

One option for configuring a large number of FPGAs is connecting all devices to a JTAG [1] boundary scan chain. Due to bit-serial communication the time of configuration is unnecessarily high, particularly with regard to supported 8-bit parallel configuration interfaces for all common FPGA types. For this reason JTAG is not considered furthermore, although it potentially is a feasible configuration technology in many cases.

As design guideline a massively parallel FPGA architecture should support the following features in order to be well manageable:

- broadcast of a single configuration to all devices in parallel
- selection of particular devices for configuration
- use fastest configuration interface to full capacity
- minimize any additional resources which is used for configuration only like nets, routing and components

[1] JTAG: *Joint Test Action Group*

All of this requirements are hard to meet because they are partly mutually exclusive. Hence a trade-off has to be found as carried out in the following subsections.

2.1 Addressing

Meeting the requirement for enabling particular as well as broadcast configuration, demands a powerful address logic. Since inevitable every parallel architecture comes with a communication scheme, it is obvious to use the present address logic for configuration too. So the lower data bus and the chip select net become dual use signals.

2.2 No Feedback

The standard process of configuration uses two types of feedback information represented by two output signals on FPGA site:

- *BUSY* indicates a stall in order to enable flow-control
- *DONE* indicates a successfully configured FPGA

Supporting this signals on a massively parallel architecture requires additional discrete components for driving the signals as well as additional nets which have to be routed - all used for the time of configuration only. By simple software workarounds the two feedback signals are made redundant at nearly maximum configuration speed. The successful configuration can be checked by the bus controller as explained in subsection 2.3. By this mechanism a iterative learning sequence of configurations at different speeds is able to find the nearly maximum configuration speed without stalls. Although this calibration process is time intensive, ideally it needs to be done once per life time of a system.

2.3 Configuration Check

A configuration concept without use of feedback signals demands an automatic configuration check by a central controller instance which might be the master bus controller. As the custom implementation at FPGA site must have a client bus controller in order to enable communication at all, the validating instance can utilize this. A simple test pattern broadcasted to every FPGA can be read out afterwards from one FPGA after another. This validation mechanism with time complexity O(n) is a considerable alternative to the hardware support of feedback signals.

3 COPACOBANA

This section presents in brief the design and realization of the COPACOBANA machine [1], which is optimized for running crypt analytical algorithms. This acronym stands for *Cost-Optimized Parallel Code Breaker* because the primary design goal was to produce a re-programmable low-cost design which can be realized for less than 10,000 $ [2] which is applicable for attacking the Data Encryption Standard (DES) in less than nine days. The realized architecture outperforms conventional computers by several orders of magnitude. Fully configured, COPACOBANA hosts 120 low-cost FPGAs [3]. In this configuration, the COPACOBANA hardware is able to perform an exhaustive key search of DES at a rate of more than 2^{35} keys per second, yielding an average search time of less than nine days. For this, the high-speed DES engine design of the Université Catholique de Louvain's Crypto Group was used.

[2]The currency is US dollar. The total cost is composed of the material and production costs of its components. Note that the NRE design costs for the layout and for programming the FPGAs are not included.

[3]Xilinx *XC3S1000-5FT256C*

3.1 Architecture

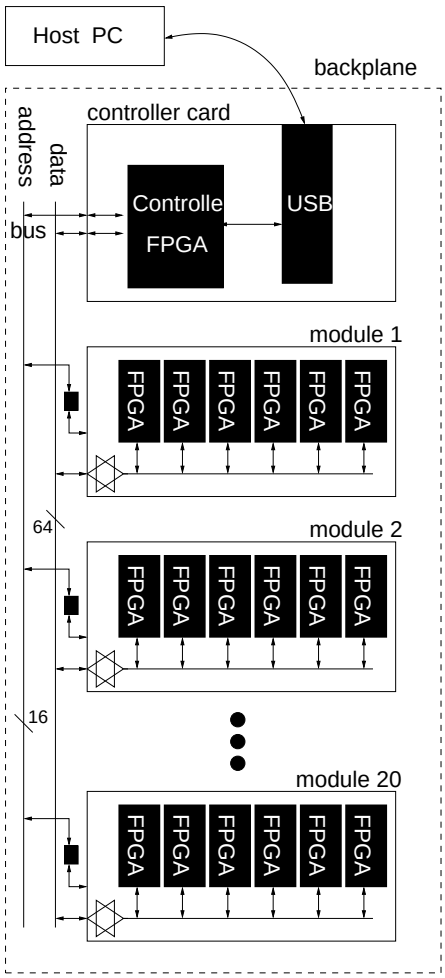

Figure 1: Architecture of COPACOBANA

The design of COPACOBANA is depicted in Figure 1. In terms of cost-optimization a lean hardware was built in order to support minimum requirements like power supply, FPGA configuration und intercommunication. COPACOBANA consists of

- *FPGA modules* for the actual implementation of the application,
- a *backplane*, connecting all FPGA modules to a common data bus, address bus, and power supply,
- and a *controller card*, as bus controller and connection to a host-PC via USB.

The backplane hosts 20 FPGA-modules at 6 devices each and a controller card. All modules are connected by a 64-bit data bus and a 16-bit address bus. This single master bus is easy to control because no arbiter is required. Interrupt handling is totally avoided in order to keep the design as simple as possible. If the communication scheduling of an application is unknown in advance, the bus master will need to poll the FPGAs.

3.2 Configuration

The configuartion concept of COPACOBANA is depicted in figure 2 showing signals and components which are responsible for configuration. The bottom part represents a slot related cutout of the bus while the upper part depicts the beginning of an FPGA-module showing one of six FPGAs. This existing hardware is mentioned here as proof of concept because ist was designed in accordance with the above presented configuration concept for massively parallel FPGA architectures.

The right site of the FPGA shows the unconnected feedback signals $\overline{INIT}$, $DONE$ and $BUSY$. The left site shows input signals according to the *slave parallel* configuration [4] [3] scheme of Xilinx Spartan-3 FPGAs [2]. A particular chip is addressed directly by the active-low chip select $\overline{CS}$ signal and indirectly by the address decoder [5] which enables the output OE of the data bus 3-state transceiver. This decoder also propagates the read signal $RD\overline{WR}$ at proper addresses, otherwise it is LOW so driving the data bus from the backplane bus into the FPGA-module local bus. The addressing for configuration uses the present bus system without need for any additional

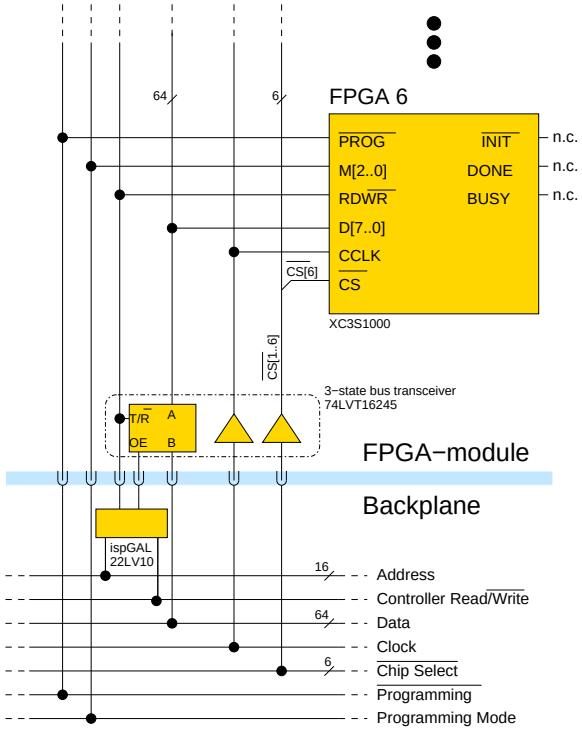

Figure 2: Configuration of COPACOBANA

ressources. This hardware was tested and brings a comfortable configuration at a lean hardware.

4 Conclusion

In this paper a configuration concept for a massively parallel FPGA architecture was demonstrated as trade-off between effort and usability. Furthermore such a system was presented in brief in order to point out how the concept was successfully implemented. As conclusion some well known configuration aspects are worth to think about for massively parallel scenarios. By disabling some features and use of proper workarounds the overall advantage justifies the concept.

[4] The 3.3V Configuration of Spartan-3 FPGAs was applied in order to enable the dual use function of D[7..0].

[5] The address decoder is implemented as Lattice *ispGAL22LV10*.

References

[1] S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, M. Schimmler: "Breaking Ciphers with COPACOBANA - a Cost-Optimized Parallel Code Breaker", *Conference on Special-purpose Hardware for Attacking Cryptographic Systems* (SHARCS) 2006.

[2] "Spartan-3 FPGA Family: Complete Data Sheet", *Xilinx Inc.*, www.xilinx.com, March 4, 2005.

[3] K. Goldblatt: "The 3.3V Configuration of Spartan-3 FPGAs", *Xilinx Inc.*, www.xilinx.com, Application Note 453 (v1.0), 2005.