

Analysis of the XC6000 Architecture for Embedded System Design

Karlheinz Weiß¹, Ronny Kistner¹, Arno Kunzmann¹, Wolfgang Rosenstiel^{1,2}

¹Forschungszentrum Informatik an der Universität Karlsruhe (FZI)

Haid-und-Neu-Str. 10-14, 76131 Karlsruhe, Germany

²Universität Tübingen, Sand 13, 72076 Tübingen, Germany

Abstract

Novel FPGA architectures combined with new implementation methods influence the widespread area of embedded system design. Current research results based on the Global Run Time Reconfiguration (RTR) method show improvements to the functional density of up to 500% in contrast to the widely used Compile Time Reconfiguration (CTR) method. In addition, the RTR method applied to a partial reconfigurable architecture, like Xilinx XC6000, promises further enhancements. This paper analyses different implementation methods in order to exploit the resources of specific FPGA architectures. The development of an ATM diagnostic monitor serves as a realistic application to analyse these methods. We will analyse the differences between the XC4000E/EX and XC6000 FPGA architecture, based on their use of the same CTR implementation method. Additionally, applying Local RTR to XC6000 leads to a further benefit of about 20% in contrast to the CTR method. The evaluation process is based on the FZI internal rapid prototyping environment, which is best suited for sophisticated ATM applications.

1 Introduction

Most of today's existing technical applications are controlled by so-called Embedded Systems¹. Many different application areas exist that demand their own embedded system architecture. Therefore, a common definition of embedded system does not exist [1].

Asynchronous Transfer Mode (ATM) is an example of advanced communication technology, whose basic embedded system architecture consists of application specific software running on a microcontroller as well as application specific hardware. This hardware part communicates with the microcontroller and the environment via complex high speed interfaces and can be implemented using Field Programmable Gate Arrays (FPGAs) and Application Specific

¹This work was supported in part with funds of the Deutsche Forschungsgemeinschaft under reference number 3221040 within the priority program "Design and design methodology of embedded systems".

Integrated Circuits (ASICs). The rapid progress of FPGA architectures in conjunction with new implementation methods influences this kind of embedded system architectures and offers system designers powerful features to enhance performance and decrease system costs.

2 Previous Work

In order to give an overview of state-of-the-art technology, we must distinguish between FPGA chip architectures (reconfigurable and partially reconfigurable) and the different design methodologies (CTR and RTR) used to implement hardware modules with these FPGAs.

• Reconfigurable and Partially Reconfigurable FPGA Architectures

The term "reconfigurable FPGA" means, that the configuration data for realizing specific hardware modules can be continually changed without any damage to the FPGA device. These FPGAs typically use SRAM cells to store the configuration data for the appropriate logic and routing information. Commonly used FPGA devices like the widely introduced XC4000E/EX family [2] or the Altera 8000 family [3] employ a serial download link and need between 1ms and 10ms for each configuration. These devices have achieved the status of a world wide standard. If even a small hardware module in an active device is changed, the entire configuration data must be erased and completely reloaded using the new configuration data.

In contrast, a partially reconfigurable FPGA offers the possibility of changing unused hardware modules in an active device, without disturbing other, currently active hardware modules during run-time. Only the unused hardware module can be replaced with new, currently needed modules via a 32 bit, high speed parallel programming interface while the rest of the chip continues its operation. Therefore, the reconfiguration time is dramatically reduced down to a few microseconds. This

new capability is offered by the novel Xilinx XC6000 [4] family, the Atmel AT6000 [5] family and the National Semiconductors CLAy [6] chip.

- **Compile Time Reconfiguration (CTR) Methodology**

The basic characteristic of CTR is that the configuration bitstream must be loaded onto the FPGA device before an operation commences. The configuration remains unchanged during the run-time of the application. The design flow is like an ASIC design and sophisticated development tools are available. It is the most commonly used implementation method for reconfigurable FPGAs and is not bound to any particular device family. That means, this method can be used for the widely introduced reconfigurable (e.g. XC4000E/EX) FPGA families and for the new partially reconfigurable FPGA architectures (like XC6000).

- **Run Time Reconfiguration (RTR) Methodology**

RTR allocates and reallocates FPGA resources at run-time. Consequently, one FPGA device can implement different hardware modules. These modules are part of an application and can be changed dynamically while the application is running. Hutchings and Wirthlin [7] further divide RTR into Global RTR and Local RTR.

Global RTR is used for applications, where necessary hardware modules can be partitioned into time exclusive parts running in a sequential manner. In this case, an application starts operation with one hardware module (typically using all of the FPGA resources) and changes during run-time to another module to support another task of the application. Only one module is active at any given time. As described in [8], Global RTR can increase the functional density of an neural network by 500% in contrast to a CTR implementation, due to the elimination of currently unused hardware modules. The Global RTR method is not tied to a specialized FPGA architecture and can be applied to any reconfigurable FPGA device. Due to the fact that devices with a parallel programming interface (like the XC6000) are able to perform a complete reconfiguration process very fast (by about a factor of 1000), such devices will become more interesting for a large group of Global RTR applications.

Local RTR changes this time exclusive concept and enables the exchange of different subsets of hardware modules, which occupy their own portions of the chip area, at any time. This leads to a more fine-grained partially reconfigurable system architecture and to a more efficient use of FPGA resources. This implementation method requires conclusive a partially reconfigurable FPGA device as described above.

This paper analyses the features of the XC6000 architecture and its related design methods in contrast to the

Xilinx XC4000E/EX family. Our approach consists of two steps: In step one, the CTR method is applied to both the XC4000E/EX and the XC6000 devices. This work, - One method applied to two different architectures-, shows the advantages based mainly on the different FPGA architectures. The second step applies Local RTR to the XC6000 architecture. This step, - One architecture applied to two different methods-, shows the additional benefits based on the new implementation method, which can exploit the partial reconfiguration capability of the XC6000 family. This analysis can help designers recognize the new features XC6000 FPGAs offer for their daily work and can help researchers focus on improving design tools and methodologies. The experimental results included in this paper were obtained through an appropriate industrial application.

The main part of the paper is organized as follows: Section 3 gives a brief functional description of our ATM diagnostic monitor. Section 4 introduces the FZI internal rapid prototyping environment for embedded ATM applications. Section 5.1 describes a hardware architecture for the ATM diagnostic monitor example, suitable for CTR implementation. Section 5.2 describes a hardware architecture suitable for Local RTR implementation and section 5.3 contains the comparative results that were obtained. Section 6 offers some concluding remarks.

3 Functional Description of the ATM Diagnostic Monitor

In this section, we briefly describe our ATM diagnostic monitor which serves as a basic and practice relevant example for our analysis. Figure 1 depicts a typical part of an ATM network. A PC endowed with an ATM network card is connected to an ATM switch via a physical medium. The physical medium could be either a coax cable or a fibre optic wire. The real communication bandwidth is subdivided into up to 4096 Virtual Paths labelled VP0..VP4095. Each Virtual Path is further subdivided into 65536 Virtual Channels, labelled VC0..VC65535. The transfer method is serial, with typical baudrates of 155 Mbit/sec. The smallest unit of an ATM message is called a cell. An ATM cell is composed of 53 bytes, 5 header bytes and 48 bytes for the payload. It's associated Virtual Path and Virtual Channel is coded with 28 bits together with other traffic information in the header.

The magnifying glass in figure 1 symbolizes the main feature of the monitor. In faulty networks, designers want to detect errors that occur during communication.

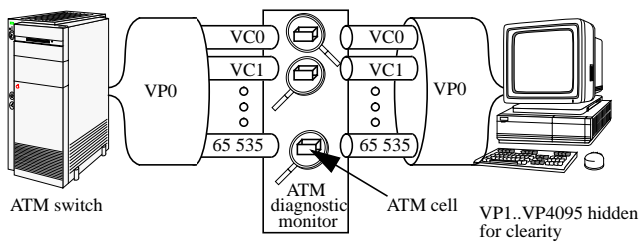


Figure 1. Functioning of the ATM diagnostic monitor

The first step is to observe all Virtual Paths and their Virtual Channels looking for faulty ATM cells. If faulty ATM cells are discovered, they must be identified and counted in relation to the associated Virtual Path and Channel. In most cases, faulty ATM cells appear sporadically, not constantly. Therefore, the monitor must be able to observe the ATM communication traffic over all Virtual Paths and Channels simultaneously for an extended period of time. The second step is to analyse the faulty ATM cell which hint at possible errors, like CRC Check Errors or Packet Length Errors. A more detailed description of ATM can be found in [10].

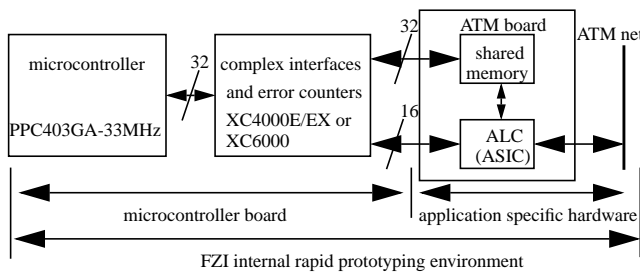


Figure 2. Abstract model for an embedded system suitable to realize the ATM diagnostic monitor

In order to transform the previous requirements into a real embedded system architecture, we need three basic component classes during the implementation process. These component classes are a microcontroller, an ASIC and a FPGA. An abstract model of the embedded system is given in figure 2. This architecture is realized using the FZI internal rapid prototyping environment introduced in section 4.

- A microcontroller tends to be very flexible, but is slow in contrast to specialized hardware. Microcontrollers are used to implement an application specific software algorithm. The ATM diagnostic monitor is based on a 32 bit PowerPC Embedded Controller [11] (IBM-PPC403GA). It executes the initialization process for the FPGA and the ASIC and sets up common data structures in the shared memory during booting. During the run-time of the application, the microcontroller is only able to con-

trol the permanent receiving process of the ASIC via the complex interface and the partial reconfiguration process of the XC6000 FPGA due to hard real time conditions.

- A customized ASIC executes its assigned task very quickly but is inflexible and cannot deal with changes in requirements. Nevertheless, the reception process of ATM cells at 155 Mbit/sec is a very difficult task, which needs specialized VLSI chips. Therefore, an Adaption Layer Controller (ALC) [12] is used. This chip is designed to receive ATM cells on all Virtual Paths and Channels, convert them to user data (this process is called reassembly) and perform dedicated error checking in real time. The ALC-ASIC records the reassembled user data onto shared memory, from which the microcontroller can read the data via a 32 bit data path through the complex interface. Additionally, it records the respective Virtual Path and Channel number (28 bits) and the error status (4 bit) via appropriate bit fields in the shared memory. Therefore, the ALC needs a dedicated shared data structure in the shared memory. This data structure must be permanently updated during the receiving process with a special algorithm running on the microcontroller and using most of its computation resources. This hardware was implemented on a special ATM board that can be connected to the microcontroller board (explained in section 4) via a 96 pin expansion header.
- A FPGA is used to connect the microcontroller interface and the two different interfaces of the ATM board. These include a 16 bit interface to the registers located in the ALC, a 32 bit interface to the shared memory, an arbiter which controls access to the shared memory, address decoders and registers for intermediate results. This part of the hardware is a complex interface, which is implemented in the FPGA device.

A further reason for the use of FPGAs is due to the following requirement: If the ATM diagnostic monitor is notified about faulty ATM cells, these cells must be recognized and counted. Therefore, the monitor has to provide dedicated Error Counters; one for each Virtual Channel as symbolized by a "magnifying glass" in figure 1. If the ATM board is notified about ATM cells at 155 Mbit/sec, the ALC evaluates the ATM cells, records the error status, the Virtual Path and Channel number in the shared memory for further use and generates interrupts every 2.73 μ s (about 90 PPC403 clock cycles). This real time condition is too quick for the microcontroller to be able to realize these Error Counters in software in addition to the update algorithm described above. Therefore, we need additional FPGA hardware to provide for the Error Counters in hardware, which should make a width of 32 bits. In the worst case scenario, this enables an

maximum observation time of 192 minutes, before a counter overflow occurs. The situation, where thousands of Virtual Channels transmit faulty ATM cells, is very unlikely and can not be implemented - since thousands of 32 bit Error Counters would be required - with a realistic amount of FPGA resources. We limited the monitor to simultaneously observe “as many Virtual Channels as possible”. If more Virtual Channels are faulty than associated Error Counters are available, an overflow bit is set. The amount of Error Counters implemented depends greatly on the implementation method and target architecture. As shown in Section 5, this problem is suitable for analysis of the XC6000 features in contrast to the XC4000E/EX devices and the different implementation methods.

4 FZI Internal Rapid Prototyping Environment

Many different environments for rapid prototyping exist. Most of them connect the PC ISA [16] or PCI [17] bus via an interface to partially reconfigurable FPGAs and are suitable for more fundamental CTR and RTR research projects. A few environments [15] [14] are principal very flexible, but because of the high degree of flexibility are not suitable for an embedded system like our ATM diagnostic monitor. This is due to by the large expansion of the resulting system leading to the loss of real time capability. Therefore, we have developed a new rapid prototyping environment.

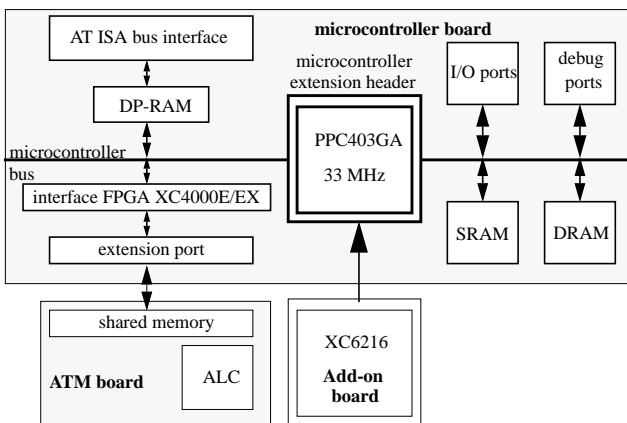


Figure 3. FZI internal rapid prototyping environment

A distinguishing feature of an embedded system in the area of communication applications is the fact that they are composed of few, highly integrated VLSI components with complex on-chip functionality. These embedded system architectures typically use three key components: A 32 bit microcontroller and FPGAs, enlarged with different ASICs. In order to meet these requirements, we have designed the

new FZI internal rapid prototyping environment depicted in figure 3, which is best suited to deal with this kind of embedded systems. This environment provides all the key components mentioned above in a flexible, but still compact architecture to guarantee high clock speeds. This enables a real time emulation, which is very close to the final target system. A detailed description can be found in [13].

The FZI internal prototyping environment consists of a microcontroller board and different add-on boards. The microcontroller board serves as a base platform and provides the whole infrastructure commonly used for most embedded systems in the described application area, i.e. the microcontroller core and FPGAs. The following points describe how the ATM diagnostic monitor application introduced in section 3 is composed via the FZI rapid prototyping environment, using the three key components mentioned above:

- The PPC403GA microcontroller needs additional devices to operate, which make up the “microcontroller core”. The 512kByte SRAM bank can be used as fast memory for time critical program code. Additionally, two 4MByte DRAM simm are available for large program code and FPGA configuration data. A 2kByte Dual Port RAM (DPRAM) is used as an interface to the PC AT ISA bus and acts primarily as a download link for program and FPGA configuration data from the cross development PC. Serial and debug ports are available to support different software and hardware debugger. This microcontroller executes the software algorithm described in section 3.
- A dedicated interface FPGA (pin grid array for different XC4000E/EX devices) connects the microcontroller bus signals at one side to a 96 pin extension header at the other side. It serves as a reconfigurable interface device to connect application specific hardware with a dedicated interface behavior to the microcontroller. The connection of the ATM board is shown in figure 2 and 3 and its needed interface is described in section 3. Additionally, this FPGA is used for the CTR implementation of Error Counters on XC4000E/EX devices, which will be explained in section 5.1.
- All microcontroller signals are connected to a microcontroller extension header surrounding the PPC403GA embedded controller chip, which acts as a master. It can be used to add a slave- or coprocessor which is closely connected to the master. This feature is used to add a small Xilinx XC6216 board, which acts as a partially reconfigurable coprocessor and is used for the CTR and Local RTR implementations explained in section 5.1 and 5.2.

address locations inside the appropriate compare unit. The number of function units for the resulting compare unit decreases, 28 XOR gates and a 28 bit local bus (the connections between one of the XOR-inputs with the VP+VC register, as described above) are no longer needed. A 28 bit bus forwards the current Virtual Path and Channel number from the central logic to the compare unit, a one bit bus signals a valid error condition and a one bit bus gives feedback to the central logic. This feedback signal indicates that an Error Counter for the current Virtual Path and Channel is present and no additional Error Counter must be instantiated. In this case, the central logic suppresses the request to install the appropriate partially reconfigurable input gates inside the compare unit via the microcontroller.

5.3 Implementation Results

At the beginning of section 5, we explicitly mentioned the term “users point of view”. That means, we maximize the number of Error Counters. Therefore, the suitability of a FPGA family for our application can be measured by the potential maximum number of Error Counters, which run simultaneously. This is what we call hardware functionality.

In the first approach described in section 5.1, the CTR architecture was developed using VHDL for design entry and the Xilinx place & route tools for backend. Table 1 shows the results of the implementation of the Error Counters (EC) on XC4000E/EX devices. Using 4020E and 4025E devices, the CLB utilization is limited to about 65%. If the utilization rises to more than 65%, the design can not be implemented due to routing restrictions. The XC4000EX devices has significantly more routing resources and the XC4028EX is able to realize up to 20 Error Counters with 99% CLB utilization. The maximum flip flop utilization

#EC	10	12	14	16	18	20	24
CLBs	505	611	724	817	898	1023	1219
Device	4020 E	4025 E	4028 EX	4028 EX	4028 EX	4028 EX	4036 EX
CLB Utili.	64%	59%	70%	79%	87%	99%	94%
FF Utili.	29%	27%	31%	35%	39%	44%	43%

Table 1. FPGA resources for CTR implementation on XC4000E/EX family

of only 44% shows that the limiting factor for the implementation of additional Error Counters is not the amount of available storage elements, but the routing resources, mainly consumed by the large on chip busses labelled b1 and b2 in figure 4.

The CTR implementation on the XC6000 architecture enables the implementation of a maximum of 20 Error Counters on a XC6216 chip. As mentioned above, the

XC6000 architecture benefits from direct access to all application flip flops without any additional routing resources (the busses b1, b2 and the address logic in figure 4 are not needed). The same amount of Error Counters in the XC4000E/EX family can be achieved with the XC4028EX device. In the data sheets [4][2], Xilinx gives a typical gate range for XC4028EX from 18,000 to 50,000 and for the XC6216 between 16,000 to 24,000. If we use an average value of 20,000 for XC6216 and 32,000 for XC4028EX, the XC6216 enables a 60% higher hardware functionality (the utilization of both XC4028EX and XC6216 comes near to 100% with 20 Error Counters).

In the second approach described in 5.2, we applied Local RTR to the XC6216 architecture. As shown in figure 6 and appendix A, each Error Counter needs a chip area of 10x16 function units. Therefore, the number of Error Counters rises from 20 to 24, which leads to an additional increase in the hardware functionality of 20%, in contrast to the CTR method on the same XC6216 chip.

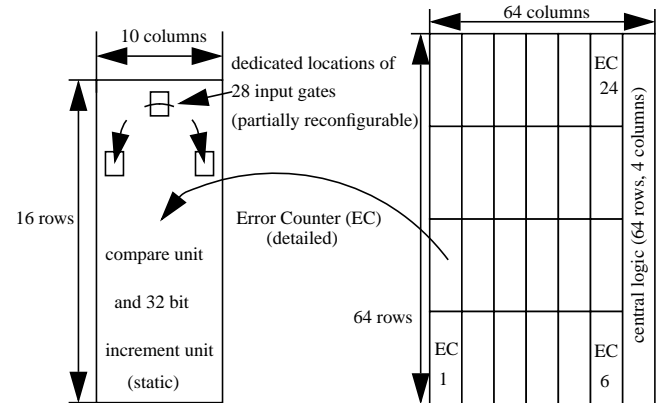


Figure 6. Chip partitioning for using the Local RTR method on XC6216

Due to the lack of appropriate design tools for the Local RTR method, the design flow is very complicated and needs much assistance from the designer, in contrast to the well automated CTR method. Much research work has to be done in the future to enable an easy-to-use Local RTR design flow.

6 Conclusions

In this paper we have evaluated the features of the novel XC6000 architecture in contrast to the XC4000E/EX family and their related implementation methods. For this evaluation, we used our recently completed ATM diagnostic monitor project as a realistic industrial example. Our experimental results show, that the XC6000 FPGA provides significantly more hardware functionality than the XC4000E/EX chips. The obtained improvement means that:

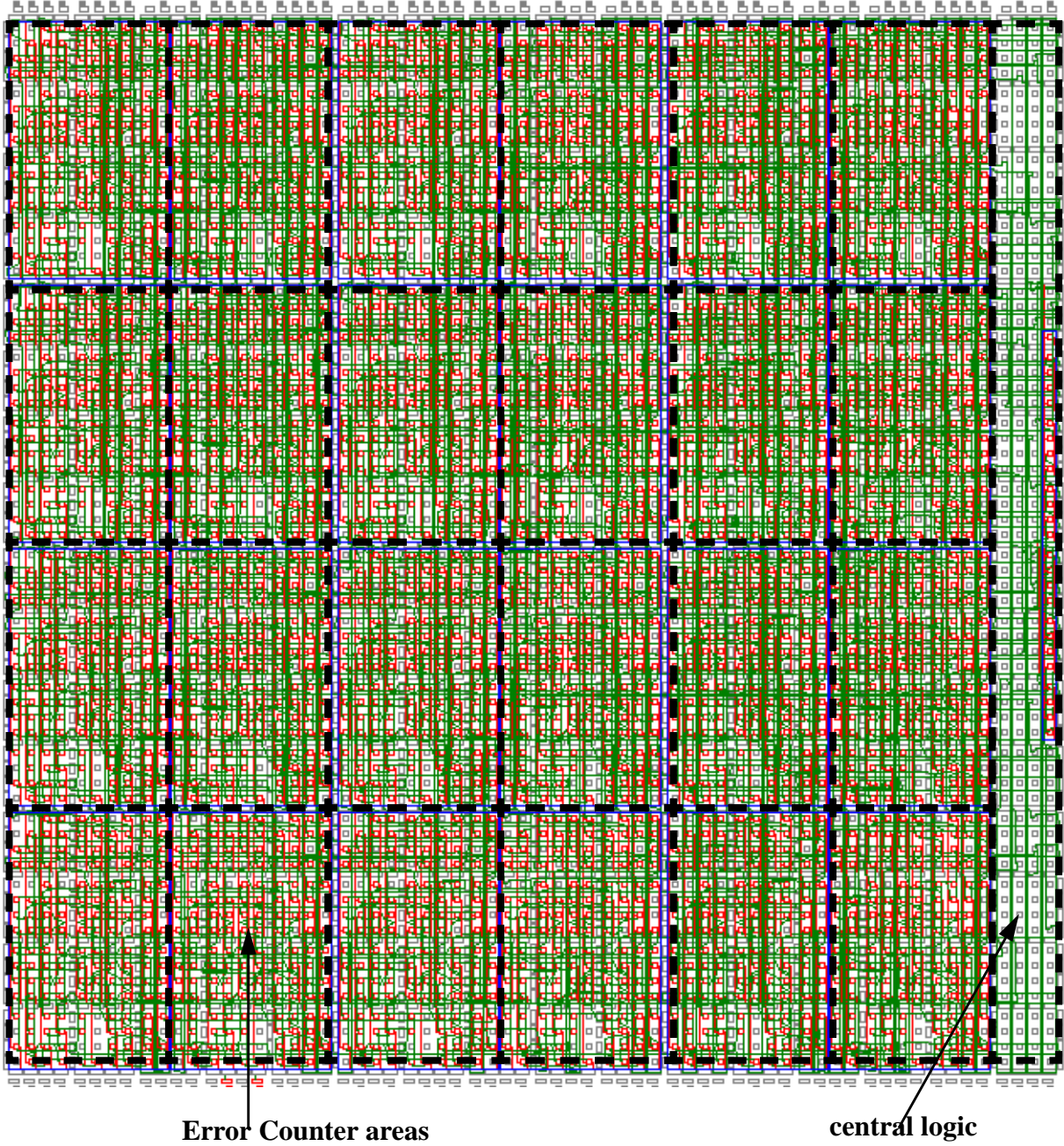
- Using the same CTR implementation method and changing the FPGA architecture from XC4000E/EX to XC6000 chips will improve the hardware functionality for gates by about 60%. For our investigated application, one XC6216 chip with an average gate count of 20,000 can replace a XC4028EX chip with an average gate count of 32,000. This improvement is mainly due to the XC6000 open architecture, which enables access to all application flip flops without any additional routing resources.
- Using the same XC6000 architecture and changing the implementation method from CTR to Local RTR will improve the hardware functionality on additional 20%. This improvement is mainly caused by constant propagation, which leads to a reduction in the consumption of function units and routing resources inside the Error Counters. This enables the XC6216 chip to realize the same functionality as a XC4036EX chip.

Our example has shown, that the XC6000 architecture contributes significant advantages to applications, which transmit many intermediate results between the hardware modules which reside in an FPGA and a microcontroller. Such applications can also be found in areas like industrial communication networks, digital signal processing or picture compression.

References:

- [1] W. Wolf: *Hardware-Software Co-Design of Embedded Systems*. Proceedings of the IEEE, Vol. 82, No.7, July 1994.
- [2] Xilinx: *Data book and application notes*. <http://www.xilinx.com>.
- [3] Altera: *Data book and application notes*. <http://altera.com>.
- [4] Xilinx: *XC6000 Field Programmable Gate Arrays*. 1996.
- [5] Atmel: *Configurable Logic: Design & Application Book*. 1993-1994.
- [6] National Semiconductor: *Configurable Logic Array (CLAy)*. Data Sheet, 1993.
- [7] B. Hutchings, M. Wirthlin: *Implementation Approaches for Reconfigurable Logic Applications*. Proceedings of 5th International Workshop of Field-Programmable Logic and Applications, Oxford 1995.
- [8] J. Eldrege, B. Hutchings: RRANN: *The Run-Time Reconfiguration Artificial Neural Network*. IEEE Custom Integrated Circuits Conference, San Diego, pp 77-80, 1994.
- [9] M. Wirthlin, B. Hutchings: *Improving Functional Density Through Run-Time Constant Propagation*. ACM-Symposium on Field Programmable Gate Arrays, 1997.
- [10] W. Goralski: *Introduction to ATM-Networking*. McGraw-Hill Series on Computer Communications, 1995.
- [11] *PPC403GA Embedded Controller - User Manual*. IBM Corporation 1994.
- [12] *MB86687A - Adaption Layer Controller (ALC)*. Data Sheet Edition 2.0, FUJITSU 1996.
- [13] A. Hergenhan, K. Weiß: *User Manual: Microcontroller Design Environment*. (postscript) <http://www.fzi.de/weiss.html>, FZI 1997.
- [14] S. Hauck, G. Borriello, C. Ebeling: *Springbok: A Rapid-Prototyping System for Board-Level Designs*. ACM/SIGDA 2nd. International Workshop on Field-Programmable Gate Arrays, Berkley, Feb. 1994.
- [15] G. Koch, U. Kebschull, W. Rosenstiel: *A Prototyping Environment for Hardware/Software Codesign in the COBRA Project*. Third International Workshop on Hardware/Software Codesign, 1994.
- [16] M. Wirthlin, B. Hutchings: *A Dynamic Instruction Set Computer*. IEEE Workshop on FPGAs for Custom Machines, pp. 99-107, 1995.
- [17] W. Luk, N. Shirazi, P. Cheung: *Modeling and Optimizing Run-Time Reconfigurable Systems*. IEEE Symposium on FPGAs for Custom Computing Machines, pp. 167-176, 1996.

Appendix A:



XC6216 chip layout with Local RTR implementation method