Elementary Block Based 2-Dimensional Dynamic and Partial Reconfiguration for Virtex-II FPGAs

Michael Hübner, Christian Schuck, Jürgen Becker
Universitaet Karlsruhe (TH), Germany
http://www.itiv.uni-karlsruhe.de/
{huebner,schuck, becker}@itiv.uni-karlsruhe.de

Abstract

The development of Field Programmable Gate Arrays (FPGAs) had tremendous improvements in the last few years. They were extended from simple logic circuits to complex Systems-on-Chip which enable the integration of complete microcontroller systems and their peripheral devices. Virtex-II FPGAs from Xilinx provide the possibility of dynamic and partial reconfiguration. This can be taken advantage of to substitute inactive parts of a hardware system and adapt the complete chip to a different requirement of an application while run-time. Existing approaches allow reconfiguration of slot based systems while run-time. Unfortunately such systems suffer from the fact, that fixed sized reconfigurable slots are not completely utilized by all functional blocks. Therefore a new 2-dimensional approach is necessary to optimize the placement of functions on the reconfiguration area for the FPGA. Benefit is a reduced chip size which leads to a reduction of power dissipation. This paper describes the method and procedure to include a 2-dimensional placement of reconfigurable blocks and the integration to a run-time system.

Keywords: dynamic reconfiguration, online routing, Virtex, 2-dimensional Placement

1. Introduction

Xilinx Virtex FPGAs offer the possibility of dynamic and partial run-time reconfiguration. This feature is used in new approaches by outsourcing configuration data which makes it possible to use FPGAs with smaller configuration memory and consequently smaller chip size. Thus it is possible to save costs and reduce power consumption since not actually used modules of a complete system do not allocate configuration memory and corresponding power consuming hardware [1]. Nevertheless power dissipation during reconfiguration has to be considered [2]. The authors of [3] raise this issue with the main focus on energy saving and basis for new design methodology.

When designing a dynamic and partial reconfigurable system on an FPGA it has to be made certain, that no signal lines of a module cross the border to another functional block. During reconfiguration, such a signal line might cause a malfunction or a short-circuit, which destroys the FPGA. Because of this it is necessary to implement interfaces which are used as fixed routing resources. These interfaces, called BUS Macros, are placed in the same position for each functional block. Connecting the modules with signal lines on the same position grants the option, to substitute a module by another. Figure 1 shows the hardware reconfigurable system described in [2].

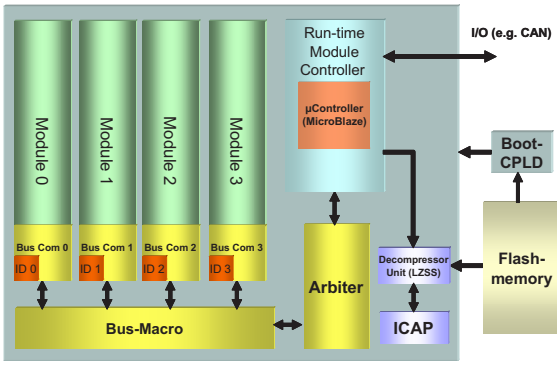

Figure 1. Dynamic reconfigurable system

In previous work, a method for dynamic and partial reconfiguration was presented in [1] and [4]. These systems have in common, that the reconfigurable area is slot based. Complete rectangular shaped areas with a fixed size can be substituted. This leads to a smaller design space in comparison to a 2-dimensional approach. The idea of integrating a Network-On-Chip was described

1-4244-0054-6/06/$20.00

in [3] and this work will describe a step forward to this new approach.
Actual and future work will support systems which have a more flexible character. Exploitation of adaptivity with an optimized utilization of chip resources is the goal for future reconfigurable systems.
The paper is organized in the following manner: Section 2 describes the basic methods and organization of the configuration memory. Section 3 describes the read-, modify writeback method for 2D-placement. In section 4 the necessary basic hardware for the approach is presented. The system integration is described in section 5. For the software support the management API is presented in section 6. Finally the paper is closed in section 7 with the conclusions.

2. Basic Methods and Configuration Memory

An existing IP-Core will be used to enable the access to the internal reconfiguration memory of the FPGA. This core allows both to read configuration data as well as writing data through an internal configuration access port [8]. Therefore configuration of the chip can be read, manipulated and finally written back to the memory. With a new approach it is possible to modify any rectangular shaped area within the configuration memory. This needs a special process which will be described in this work. Xilinx provides the JBits tool which enables the manipulation of existing configuration bitstreams. Java classes allow the access to previous stored bitstreams with a development computer. The goal of this work is to introduce an autonomous system integrated on an on-chip processor. Here the well introduced MicroBlaze processor is used. To reduce the overhead for software manipulating the bitstream, an own C-based approach was developed which has its paradigm in the JBits tool. Our software is optimized for the necessary functions while JBits provides a facility of functions for more manipulation possibilities.
Basic principle of this work is the Xilinx Virtex-II product family which includes 11 devices. The number of system gates range from 40K with the XC2V40 to 8M with the biggest device XC2V8000. Virtex-II FPGAs belong to the family of fine grained reconfigurable architectures with a variety of configurable elements like logic and routing resources. This enables a flexible and fine-grained possibility of adaptation while run-time. This is exploited if dynamic and partial reconfigurability is used for run-time adaptive systems of the future. Support for this is the internal reconfiguration access port (ICAP) which allows the access to the configuration memory without external wiring or additional devices. In the following chapter the most important and for this work fundamental criteria of the architecture and its structure will be discussed.

2.1. Virtex-II Architecture

Virtex-II FPGAs consist, like the most Xilinx FPGAs, of four basic elements which are built up in a regular array structure. The CLB-blocks (Configurable Blocks) build up the kernels of the device. They include the combinatorial logic and the register resources. Internal, a CLB- block is made up of 4 similar slices. Each slice includes two configurable Look-up-Tables (LUT), two registers and two multiplexers. To provide sequential logic, slice outputs can be connected to inputs. The LUTs can be also configured as memory (RAM) or shift register. Through the switch matrixes, adjacent to the CLB- blocks, each logic block has access to the routing resources of the FPGA. The routing resources run in horizontal and vertical direction between each switch matrix. Different length build up a set of optimal routing resources for connecting the utilized logic element on the chip (see figure 2).

- Long Lines are bi-directional circuits which overstretch the complete height and width of the module.
- Hex Lines are unidirectional circuits which route the signals to every 3rd and 6th block in all four directions. The signal can either be gripped in the middle and at the end of the circuit.
- Double Lines are unidirectional circuits which route the signals to every next and next but one block in all four directions. The signal can either be gripped in the middle and at the end of the circuit.
- Direct Connect Lines are unidirectional circuits which route signals to the neighbouring horizontal, vertical and diagonal blocks.
- Fast Connect Lines connect CLB internal outputs of LUTs with inputs of LUTs. The continuous structure of

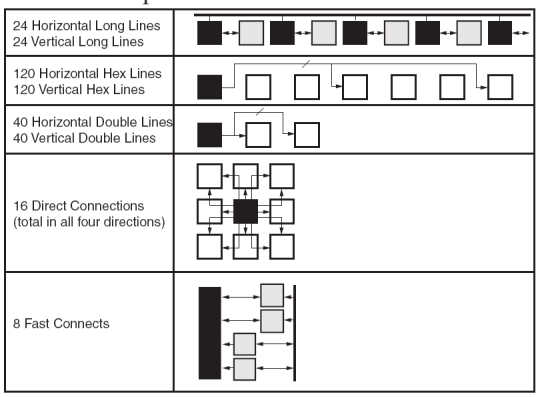

Figure 2. Hierarchic Routing-Resources [9]

CLB-blocks and routing- resources in the centre of the module is interrupted by the vertically running BRAM- and multiplier-columns. These are also connected by the global routing-network.

The outer brim is formed by IO-blocks which are used as a connection to the peripheral devices. Additionally there are blocks with resources for he clock-management which have to be dealt with separately. Further details on the architecture of Virtex-II can be found in [9].

2.2. Memory Architecture

All characteristics of the configurable FPGA-elements are controlled by memory cells which have to be initialized after a voltage has been impressed. This kind of memory is referred to as the configuration memory of the FPGA.

Writing into the configuration memory is accomplished via a bitstream which contains information about internal configuration logic and data meant for the configuration memory.

The configuration memory is organized in frames with a width of 1 bit which stretch across the entire height of the FPGA (figure 3). A frame is the smallest addressable unit of the configuration memory which means that all manipulations have to be carried out on frame basis. Configuration frames cannot directly be related to a defined hardware unit. In fact they set up different physical units alongside a narrow FPGA column. The length of a frame depends directly on the number of CLB-columns and therefore varies depending on the type of module inside the product line. The frames are classified into 6 different groups: CLB, IOB, IOI, GCLK, BRAM and BRAM_INT, which can roughly be assigned to the

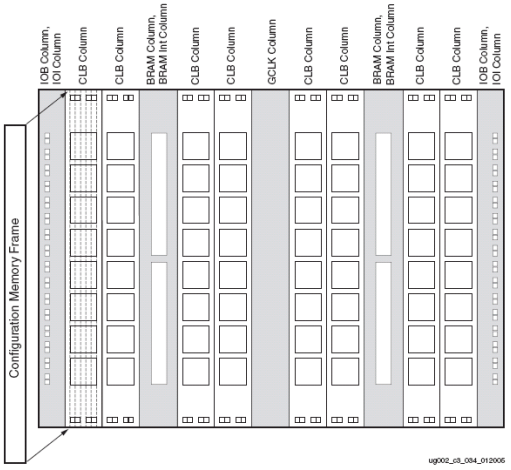

Figure 3. Virtex-II Configuration-Frame-Mapping [9]

physical resources. IOB- frames set up the IO-blocks at the left and right brim. The IO-blocks at the top and at the bottom are controlled by the matching CLB-frames. These are also responsible for the entire CLB-configuration, the routing-resources and the IOB-registersat the top and bottom brim. Similar to the IOB-frames the IOI-frames account for the configuration of the registers at the left and right brim. By using the BRAM-frames it is possible to set the content of the block-RAM memory. The BRAM_INT-frames set up the remaining BRAM-characteristics and the multipliers. The GCLK-frames in the centre are responsible for the global clock-network. The configuration memory area of a physical FPGA column consists of different frames and depends on the type of resource it contains (see table 1).

This way 22 configuration frames can be assigned to a physical CLB-column for example.

Table 1. # minor frames per column

Type	IOB	IOI	CLB	BRAM	BRAM INT	GCLK
# Frames	4	22	22	64	22	4

The addressing of the single frames is carried out by an unambiguous 32-bit address which consists of `Block Address´ (BA), `Major Address´ (MJA) and `Minor Address´ (MNA).

Table 2. frame addressing

X	BA	MJA	MNA
31-27	26-25	24-17	16-9

The configuration memory is divided into three types of blocks which are assigned to the columns with the appropriate resources as follows:

- Block Address (BA) 0: GCLK, IOB, IOI, CLB
- Block Address (BA) 1: BRAM
- Block Address (BA) 2: BRAM_INT

Figure 4 shows the memory map at Major-Frame-Level.

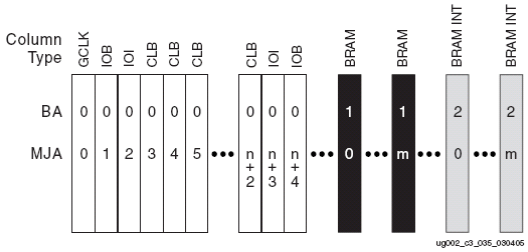

Figure 4. Memory map Major- Frame-Level [9]

It is a function of the number of existing CLB-columns (n) and the number of existing BRAM-columns (m). A Major-Frame therefore matches a physical column of the

FPGA. The frames which are part of this column are addressed by the Minor-Address. It ranges from 0 to the number of frames -1, e.g. from 0 to 21 for a CLB-column.

The manufacturer of Xilinx gives no particulars about a detailed mapping of a special FPGA-resource on single bits inside a configuration frame. Hence it is not possible without any difficulty to read the configuration out of an LUT or a routing-resource from the referring frames.

2.3. ICAP-Interface

Virtex-II FPGAs employ different modes and interface to access the configuration logic. In the following paragraph the Internal Configuration Access Port (ICAP), which was employed in this work for configuration, is briefly introduced. The ICAP-interface is on-chip. It provides reading as well as writing access on the configuration memory for the hardware implemented on the FPGA. The communication protocol employed matches the SelectMap- Interface in Slave-Mode which is not dealt with here in detail however, as a completed IP-Core has been utilized in this work which already implements the protocol.

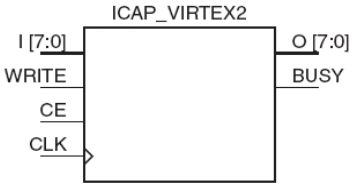

Figure 5. ICAP–Interface [8]

Figure 5 shows the ports of the ICAP-Primitive. Unlike the SelectMap-Interface it employs separate unidirectional data lines with a width of 8 bit for the input when the configuration is written and the output when it is read out. Using an ICAP-interface one has to bear in mind that the configuration-modes configuration-pins of the FPGAs M0-M2 are not set on JTAG `Boundary Scan´ Mode (101), as this setting disables the ICAP-Interface.

3. Read-Modify-Writeback Method

As already explained in the basic principles the Xilinx Virtex-II FPGAs permit a dynamic reconfiguration of the components of the configuration memory while run-time of the remaining hardware, the so-called partial dynamic reconfiguration. Due to architectural features of the internal configuration logic (see paragraph 2.2) it is however only possible to address the configuration memory on frame basis and thus solely read or write configuration data frame by frame.

As a frame stretches across the entire height of the module it follows that at all times all resources which are in one FPGA-column are affected by this process of reading or writing a frame. However Xilinx FPGAs hold the characteristic of `Glitchless Switching´ which means that when transcribing the up-to-date configuration with an identical configuration the function of the affected resources is not influenced. These conditions provided, single FPGA resources can be manipulated through the `Readback-Modify-Writeback´ method, without interfering with the remaining resources contained in the column. Therefore the corresponding configuration frame is read out of the configuration memory with the aid of a readback-sequence. This is followed by the manipulation of the bits inside the readback frame which are assigned to the FPGA-resource. The other bits remain unmodified. Finally the partly changed frame is written back into the configuration memory with a write-sequence. The problem in this operation is the fact that the manufacturer Xilinx gives no particulars about the addressing of single resources inside a frame.

4. Basic Hardware System

The substructure of the systems hardware is the Virtex-II FPGA product line by Xilinx. With the aid of the software package `Xilinx Platform Studio´ (XPS) (see [5] and [6]) provided by the firm complex micro-controller systems can easily be developed. Apart from micro-controllers these mainly contain buses as well as functional units which are attached to buses like memory-controller or timer. The single components are part of the software package and are provided as complete IP-cores. These are merged; the newly developed system is then synthesized and loaded onto the FPGA as a bitstream.

To begin with a basic system was created which holds the following basic functions:

- enable access to the internal ICAP -Interface
- provide user interface to interact with the system
- allow access to an external data memory
- permit software debugging

The most important necessary units and their function will be shown in the following paragraph.

4.1. OPB-HwIcap-Core

The OPB-HwIcap-Core by Xilinx forms the interface to the internal configuration-interface ICAP (see paragraph 2.3). The module internally consists of an OPB-controller which works as interface for the OPB-bus, a ICAP-controller to regulate the data- and control-flow between ICAP-Primitive, Block-RAM and OPB-controller and a Block-RAM memory. As can be gathered from figure 6 the exchange of configuration packages (data) is carried out by the OPB-controller (i.e. processor) and ICAP-primitive through the BRAM-memory. With a

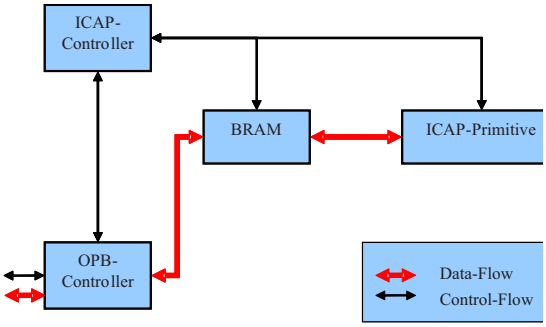

Figure 6. OPB-HWICAP-Core

magnitude of 2KB it is dimensioned in a way that it can at least store one configuration-frame.

A configuration frame read back by the ICAP-Primitive can therefore be directly modified by the processor inside the BRAM-memory for example, before it is finally written back. The software driver included by the firm provides functions to write and read data between the processor and BRAM-memory as well as between BRAM-memory and ICAP-primitive.

5. System Integration

In contrast to a first online peripheral approach with JBits (not described here), the following approach tries to put the Read-, -Modify- Writeback process into practice as on-chip solution on the FPGA, i.e. on the MicroBlaze Soft-Core-Processor which is implemented there. As a suitable possibility to communicate with the ICAP-interface the driver of the HwIcap-module, which was already employed and tested in the peripheral approach, can be used. The task is now mainly restricted to deriving the addressing of the single FPGA-resources and therefore enable the manipulation of the single resources.

The conceptional goal of this work was to exchange rectangular shaped configuration areas with each other with the aid of the Read-, -Modify- Writeback process or shift or swap them to a different area. The smallest rectangular shaped configuration area which can be exchanged with any other rectangular shaped area is a CLB-block with a related routing-matrix. This combination forms a so-called elementary-block (E-block). A subtler granularity e.g. on the slice-level, is not possible without restrictions. Single slices inside a CLB-block are different and therefore cannot be exchanged with each other. As a consequence of the different multiplexers the configuration of a slice 0 cannot be transferred to one of a slice_1 for example. Larger rectangular areas always consist of several elementary-blocks. Therefore it is sufficient for the conceptional formulation to deal with the addressing of elementary-blocks, i.e. CLB-block plus Routing-Matrix inside a frame or a group or frames. Due to missing documentation the addressing of E- blocks within a frame was developed first. Afterwards the methods which allow the optional shifting or exchanging of rectangular shaped areas were realized in C-Code.

5.1. Elementary Block Approach

The basic idea behind the approach is to group single E-blocks to rectangular shaped modules and to provide an API to swap and shift the modules on the base of a homogenous FPGA area. Besides that an API function to load previously stored modules from an external memory was implemented. A key property of the modules is that they are self contained i.e. no routing wires are crossing the modules borders.

To store a module, in a first approach, the complete configuration frames, containing the relevant module information are stored sequentially, in a bitstream like manner like explained in next section.

5.2. Storage of Modules

In order to store rectangular shaped modules in an external memory the actual module data that is included within different configuration frames is extracted from the .bit file of the complete design. This is done by a small tool based on the Xilinx JBits API. In the first step the tool takes the filename of the .bit file, the lower left E-block coordinates and the size of the module in height and width and cuts of the corresponding set of configuration frames. In a second step, to omit the placement information, a scaling of the module data is done by down shifting the module within the frames. Finally in a third step the tool writes the single frames sequentially to a new .flh file which in turn can be stored in the external memory. Another small tool was written to transfer the prepared .flh module data to the flash memory via a serial connection.

6. Configuration Management API

Based on the E- block approach we developed a new configuration management system which is able to load rectangular shaped modules from an external module repository, place them to an appropriate position onto the reconfigurable chip area and hook them up to an existing bus system. The system can be triggered by a serial interface from the outside as well completely on chip from a controlling application via its API functions. For a seamless integration the serial interface implements a protocol to transfer setup parameters and module data to consistent memory.

Similar to the approach described in [2] the proposed system uses a slot- based concept. The systems lifecycle is based upon two phases, the initialisation phase and the

runtime phase. In the initialisation phase of the system the reconfigurable chip area is partitioned into a static and dynamic part, whereas the dynamic part consists of distributed configuration slots like shown in figure 7. The system configuration parameters like, quantity, position, and size of the configuration slots and the width of the used routing module are stored in an external memory. In a second step the routing module and the data of the reconfigurable modules are uploaded through the serial interface as well. In this way the system is highly flexible and can be tailored to the application specific needs of various partial and dynamically reconfigurable designs.

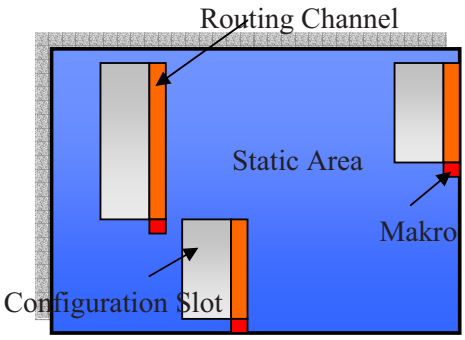

Figure 7 System partitioning example

Once the initialisation phase is completed the system is ready for reconfiguration of the slot structure. The dynamic modules stored in the flash repository can be inserted into the slots. A detailed description of the used online routing mechanism, the basic structure of the software and the simplified system design flow trough the use of the API are described in the next chapters.

6.1. Online Routing

To enable the feature of online routing the reconfigurable area of the design was divided into vertical configuration slots similar to the approach described in [7]. The key property of such a slot is that the underlying FPGA architecture is homogenous in both vertical and horizontal direction and that it is identical for each slot. This allows rectangular shaped modules to be placed into any of the configuration slots regardless of its vertical position within the slot. It is also possible to place multiple smaller modules, that don't use the complete slot height, on top of each other into one configuration slot, as long as the sum of the module heights does not exceed the slot height.

The modules in the repository are self contained with respect to their used routing and CLB resources. In order to communicate with the outer world each module has a LUT-based communication interface in the lower right corner of the module. Via its interface the module can be dynamically connected during runtime to a LUT-based communication primitive.

The communication primitives are located adjacent to each configuration slot in so called "vertical routing channels", which can be seen in figure 8. Each slot has its own routing channel attached to it which covers the whole height of the slot. This ensures that the modules have access to the communication primitives, no matter on which vertical position they are placed within the slot. A hard wired macro, which builds the bridge to static area, is placed on the bottom. In the initial configuration the slots and the vertical routing channels at the left are completely empty (figure 8 a). All configurable elements are loaded (placed) into this area form configuration management system, running on the MicroBlaze processor [7] at runtime.

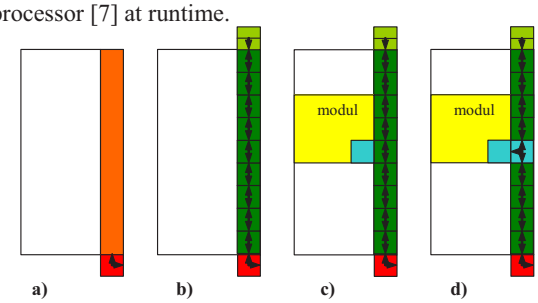

Figure 8 Online Routing Procedure

The online routing is based on only two types of communication primitives, which can be seen in Figure 9. Type-I primitives route the signals from the previous to the next Type-I or Type-II primitive in a vertical direction (up/down) along the routing channel. Type-II primitives do the same as Type-I primitives but in addition they also route the signals to the left in horizontal direction. The height of a Type-I primitives is one CLB-row. The width of Type-I and Type-II primitives and the height of a Type-II primitive depends on the number of signals to route. Per CLB-row/column height/width it is possible to route 8 signals.

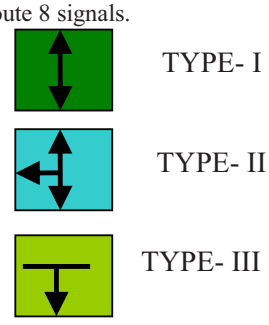

Figure 9 Routing Block Types

When the configuration management system is initialised all vertical routing channels are filled up with Type-I primitives and a special closure primitive on top of each channel, see figure 8 b). It ensures that the dangling

inputs of the last Type-I primitive have a defined low-level. As with the modules the routing primitives are stored in the flash repository and can be placed into the routing channels just as the modules can be placed into the slots.

After a module is placed into one of the slots Figure 8 c), it needs to be connected to the routing primitives. This is done by just replacing the Type-I primitive, which is adjacent to the modules interface, with a Type-II primitive. As stated before the Type-II primitives also routes the signals to the left and therefore it hooks up to the modules interface. The module is connected as described in Figure 8 d). This is the same for each module regardless of its position within the slots.

Unrouting a module is done in the opposite way. The Type-II primitive, which was connected to the module interface, is replace by a Type-I and the area where the module was placed is overwritten with a blank configuration. Routing and unrouting a module by swapping Type-I with Type-II primitives and vice versa does not effect the other modules as the primitives are designed in a way that they feature glitch less switching of the Virtex-II configuration logic.

6.2. Simplified System Design Flow

Building a partially reconfigurable system is extremely simplified through the use of the new configuration management API compared to the proposed "modular design flow" from Xilinx. The reason is that the area of the static design parts in no longer touched by reconfiguration of the dynamic parts. The use of long horizontal connection macros which involves a waste of chip area and serious timing problems is also no longer necessary. This chapter gives a brief overview which steps are necessary to create partially reconfigurable designs with the use of the new API.

All components of the static design and the reconfigurable modules are implemented with the Xilinx XTS/ISE tool chain. The partitioning of the system (compare section 5) is done through use of area- and configprohibit- constraints within the .ucf file. As a basic rule all instantiated components need to be locked down. The static ones to the static part and the dynamic modules to the bottom position within one of the slots. Every slot must contain just one module at that time. The rest of the slot area is blocked by a configprohibit- constraint. As there are usually more dynamic modules than slots in the design this process needs to be repeated until every module was instanciated at least once. The single elements of the online routing system (chapter 6.1) are generated through a LUT base prototype macro as shown in figure 10. To obtain the smallest possible granularity of the module position within the slots, double lines are used as routing resources.

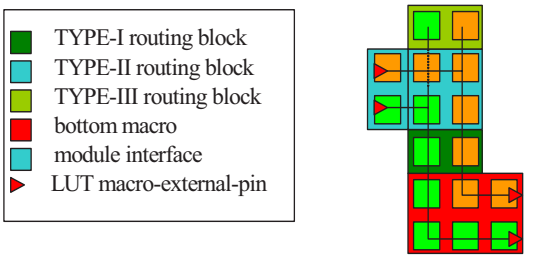

Figure 10 Prototype Routing Macro Example

The key property of the macro is its homogeneity, so the single elements can be combined with each other like described in section 6.1. The routing macro is also added to the design and instantiated once for every configuration slot. Its placement is fixed with a lock constraint in the .ucf file to a position, so that the bottom macro part is just below the routing channel. On the one side the macro-external- pins are connected to the module interface and on the other side to a application specific bus system.
Once all components of the system, the static once, the dynamic reconfigurable modules and the routing prototype macros are connected together the system is synthesized and a .bit file of the entire design is created. On the base of this design the introduced tool chain is used to extract the routing blocks and the dynamic reconfigurable modules which are converted to a flash compatible .flh file respectively (chapter 5.2). After that the .flh files are transferred to flash and the .bit file of the entire design is loaded into FPGA configuration memory.

6.3. Software Structure

The software of the configuration management API is implemented in C and made to run on a MicroBlaze processor It is build of a hierarchic modular concept (figure 11). Therefore it is easy to adapt and extend, whereas for most application only the flash driver needs to be adapted.

The bottom layer is comprised of the device drivers and utility modules which directly access the hardware or extend the driver functionality respectively. On the layer above are the manager modules. Every manager module is responsible for a specific task during system runtime.

The ICAP manager module operates on the ICAP interface and covers the implementation of the proposed RMW- method. The flash manager module organizes all consistent data in flash memory and grants access to them. The central element is the configuration manager module that handles the reconfigurable slot structure and the proposed online routing capability. It also provides all the API- functions to control the reconfiguration process

and to obtain the system status information. The serial interface module is only needed during the initialisation phase to transfer the system parameters and the module data. It can be omitted during runtime phase if the system is controlled by a host application on chip. In the case of an external controlling host application it forward the incoming commands to the appropriate manager module. The API is the basis for extension of existing run-time systems which are presented in [2].

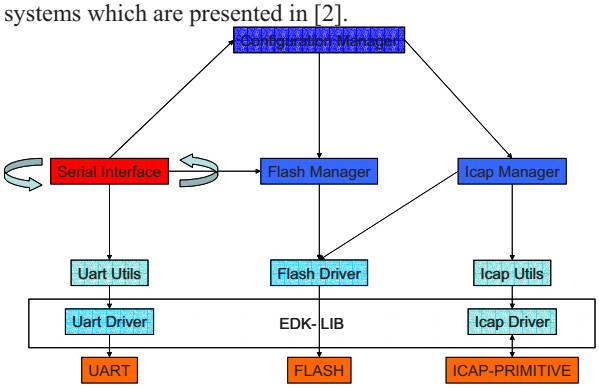

Figure 11 Modular Software Concept

7. Conclusions and Future Work

The paper discussed new perspectives and corresponding approaches for today's microelectronic embedded and processor solutions incorporating on-demand reconfigurable datapath allocations on suitable granularities in real-time. The focus of the paper has demonstrated substantially new dynamic and partial reconfiguration techniques for 2D FPGA placement and routing adaptation for today's fine-grain Xilinx devices. State of the art solutions of partial and dynamic reconfigurable systems provide the substitution, of fixed rectangular shaped blocks of hardware modules while other parts stay in operation. The circumstance caused by the FPGA architecture and its reconfiguration mechanism, forces to substitute parts of a design in complete columns. Therefore, the reconfiguration of variable rectangular shaped hardware modules is a challenging task since configuration of hardware in the same column, doesn't has to be affected while this process. The approach introduced here demonstrated variable two-dimensional placement of hardware modules to Xilinx Virtex-II FPGAs, which offers an optimized utilization of such kind of devices, in comparison to the traditional partial reconfiguration approach with fixed geometry column-based topologies of module slots.

In summary, the transparent and easy programmable integration of finegrained reconfigurable architectures into today's and future microelectronic solutions for embedded systems and for universal as well as application-tailored processor architectures demonstrates very beneficial perspectives for short time-to-market pressure, for low cost and power consumption, and for additional options in fault tolerance and risk management.

8. References

[1] J. Becker, M. Hübner, M. Ullmann: "Power Estimation and Power Measurement of Xilinx Virtex FPGAs: Trade-offs and Limitations", SBCCI03, Sao Paulo, Sep. 03

[2] J. Becker, M. Hübner, M. Ullmann: "Real-Time Dynamically Run-Time Reconfiguration for Power-/Cost-optimized Virtex FPGA Realizations", VLSI03, Darmstadt, Sep. 03

[3] L. Benini, G. De Micheli: "Networks on Chip: A New Paradigm for Systems on Chip Design", Date 02, March 3~7, Paris France

[4] J.C. Palma, A. Vieira de Melo, F. G. Moraes, N. Calazans, "Core Communication Interface for FPGAs", Proceedings of 15th Symposium on Integrated Circuits and Systems Design (SBCCI), 2002,Porto Alegre BRAZIL

[5] http://www.xilinx.com/ise/design_tools/

[6] http://www.xilinx.com/ise/embedded/edk.htm

[7] M. Huebner, T. Becker, J. Becker "Real-Time LUT-Based Network Topologies for Dynamic and Partial FPGA Self-Reconfiguration", SBCCI04, Brasil

[8] B. Blodget, S. McMillan: "A lightweight approach for embedded reconfiguration of FPGAs", DATE´03, Munich Germany

[9] Xilinx, Virtex-II Platform FPGA User Guide UG002(v2.0), March, 23 2005