The Erlangen Slot Machine: A Dynamically Reconfigurable FPGA-Based Computer *

Mateusz Majer, Jürgen Teich, Ali Ahmadinia, and Christophe Bobda

University of Erlangen-Nuremberg
Hardware-Software-Co-Design
Am Weichselgarten 3, 91058 Erlangen, Germany

Abstract. Computer architects have been studying the dynamically reconfigurable computer [1] for a number of years. New capabilities such as on-demand computing power, self-adaptiveness and self-optimization capabilities by restructuring the hardware on the fly at run-time is seen as a driving technology factor for current research initiatives such as autonomic [2, 3] and organic computing [4, 5]. Much research work is currently devoted to models for partial hardware module relocation [6] and dynamically reconfigurable hardware reconfiguration on e.g., FPGA-based platforms. However, there are many physical restrictions and technical problems limiting the scope or applicability of these approaches. This led us to the development of a new FPGA-based reconfigurable computer called the *Erlangen Slot Machine*. The architecture overcomes many architectural constraints of existing platforms and allows a user to partially reconfigure hardware modules arranged in so-called *slots*. The uniqueness of this computer stems from a) a new slot-oriented hardware architecture, b) a set of novel inter-module communication paradigms, and c) concepts for dynamic and partial reconfiguration management.

1 Introduction

Growing capacities provided by FPGAs as well as their partial reconfiguration capabilities have made them the ultimate choice for reconfigurable computing platforms. Partial reconfiguration is useful a) to increase the flexibility in computation and b) for efficiency reasons by time-sharing the resources on the device. It requires run-time loadable modules that are typically pre-compiled and stored as bitstreams, which will then be used to reconfigure the device, i.e., allocate space for the subsequent execution the module. Several models and algorithms for on-line placement have been developed in the past, see e.g., [7, 8, 6]. However, these algorithms are limited by two main factors. First of all, the model assumptions are often not realistic enough for implementation on real hardware or require very tedious processes. Second, the development process of modules

* Supported in part by the German Research Foundation (DFG), SPP 1148 (Rekonfigurierbare Rechensysteme) under contract TE163/14-2 and by Xilinx Inc.

is subject to many restrictions that make a systematic development process for partial reconfiguration difficult.

Until now, no FPGA-based platform on the market provides a solution to the problems of design automation for dynamically reconfigurable hardware modules and their efficient and flexible relocation. The purpose of the *Erlangen Slot Machine (ESM)* [9–12] is to overcome many of the deficiencies of existing FPGA-based reconfigurable computers by providing:

- A new flexible FPGA-based reconfigurable platform that supports relocatable hardware modules arranged in so-called *slots*.
- Tool support for the development of run-time reconfigurable computation and communication modules using new inter-module communication paradigms.
- A powerful reconfiguration manager which enables various preprocessing stages for fast bitstream manipulation. We call the preprocessing stages *plug-ins*. For example, a relocation plug-in can be selectively activated before a bitstream is uploaded to the FPGA.

Reconfiguration times in the range of seconds [13] are not sufficient for applications that require a fast reaction to external events. Our hardware reconfiguration manager described in this paper is the foundation for reconfiguration times in the range of milliseconds. For example, these fast reconfiguration times will allow a seamless switching of video filters in a video pipeline processing architecture.

This paper is organized as follows: In Section 2, we summarize the major problems of today's reconfigurable computers with respect to allowing partial reconfiguration and unrestricted module relocation. Section 3 provides an overview of the Erlangen Slot Machine. Moreover, we show the advantages of our platform in comparison to related work in this area and give an overview of target application domains for this platform. One major issue still solved unsatisfactorily is inter-module communication. In Section 4, we therefore provide different new paradigms for inter-module communication of relocatable hardware modules. Section 5 presents application domains for the Erlangen Slot Machine, such as video and audio streaming. In Section 6, we present the concept of our reconfiguration manager as well as different workload scenarios. These scenarios show the enhanced flexibility of our concept for different application domains. In Section 7, we conclude our work and provide an outlook of future work.

2 Drawbacks of existing systems

Despite the announcement made by several companies in the last couple of years about the design and production of new and mostly coarse-grained reconfigurable chips [14–18], the dominant part of today's reconfigurable computing platforms are still fine-grained and FPGA-based. The growing capacities provided by FPGAs as well as their partial reconfiguration capabilities have made them the ultimate choice. Xilinx FPGAs [19] combine the advantages of large capacity and

the ability to support partial reconfiguration. The Virtex series offers enough logic for efficiently implementing applications with high demand of resources, e.g., arising in video, audio and signal processing as well as in other fields like automotive applications.

There are, however, many open problems concerning module relocation: One particular problem is, for example, in order to connect a module to other modules and/or pins, signals are often required to pass through other modules. We call those signals used by a given module and crossing other modules *feed-through signals*. Using feed-through lines to access resources has, however, two negative consequences, as illustrated in Figure 1:

- Difficulty of design automation: Each module must be implemented with all possible feed-through channels needed by other modules. Because we only know at run-time which module needs to feed through a signal, many channels reserved for a possible feed-through become redundant.
- Relocation of modules: Modules accessing external pins are no longer relocatable, because they are compiled for fixed locations where a direct signal line to these pins is established.

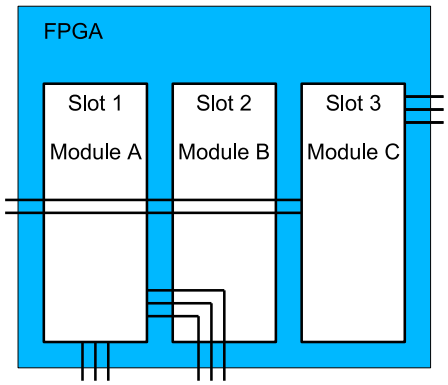

Fig. 1. The feed-through line problem with relocatable modules. Placing a new module B into slot two requires that the new module provides all feed-through lines needed by slot one and three. This fact disables any module relocation and makes it impossible to place modules with different feed-through requirements into the other slots.

Many FPGA-based reconfigurable platforms such as [20–25] offer various interfaces for audio, video capturing and rendering and for communication. However, each interface is connected to the FPGA using dedicated pins at fixed locations. Modules with access to a given interface such as a VGA (video graphics adapter, see Figure 2) must be placed in the area of the chip where the FPGA signals are connected, thus making a relocation impossible. Until now, no platforms on the market provide a solution to these problems.

Before we present in detail the architecture of the Erlangen Slot Machine and some required factors for innovation, we summarize most of the important problems limiting the use of partial and dynamic reconfiguration on current existing FPGA-based reconfigurable computers:

1. **Limitation of partial reconfiguration support on actual FPGAs:** Very few FPGAs allowing partial reconfiguration exist on the market. These few FPGAs, like the Virtex series by Xilinx [19], impose nonetheless some restrictions on the least amount of resources that can be reconfigured at a time, for example column-wise reconfiguration.
2. **I/O-pin dilemma:** Most of the existing platforms include I/O peripherals like video, RAMs, audio, ADC (analog to digital converter) and DAC (digital to analog converter) connected at fixed pins of the device. As a consequence of these pin constraints, reconfiguration may be difficult or even impossible. Another problem related to pins is that the pins belonging to a given logical group like video, and audio interfaces are not situated closely to each other. On many platforms, they are spread around the device. A module accessing a device will have to feed many lines through many different components. This situation is illustrated in Figure 2: Two modules (one of which is a VGA module) are implemented. The VGA module uses a large number of pins at the bottom part of the device and also on the right hand side. Implementing a module without feed-through lines is only possible on the two first columns on the left hand side. The effort needed for implementing a reconfigurable module on more than two columns together with the VGA module is very high. Celoxica [20], XESS [21], Nallatech [22], and Alpha boards [23] all exhibit the same limitations. On the XF-Board [24, 25] from ETH Zurich, the peripherals are connected to one side of the device. Each module accesses I/Os through an operating system (OS) layer implemented on the left and right part of the device. Many other existing platforms like the RAPTOR board [26], Celoxica RC1000 and RC2000 [20] are PCI systems that require a workstation for operation. The use in stand-alone systems as needed in many embedded systems is not possible.
3. **Inter-module communication dilemma:** Modules placed at run-time on the device typically need to exchange data among each other. Such a request for communication is dynamic due to run-time module placement. Dynamically routing signal lines on the hardware is a very cumbersome task. For efficiency reasons, new communications paradigms must be investigated to support such dynamic connection requests, for example packet-based DyNoCs [27] or principles of self-circuit routing.
4. **Local memory dilemma:** Modules requiring large amounts of local memory cannot be implemented since a module can only occupy the memory inside its physical slot boundary. Storing data in off-chip memories is therefore the only solution. However, existing FPGA-based platforms often have only one or two external memory banks and their pin connections are spread loosely over the borders of the FPGA.

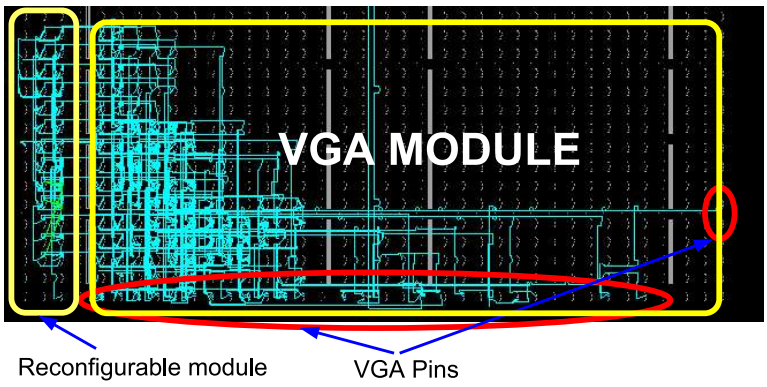

Fig. 2. Pin distribution of a VGA module on the RC200 platform. It can be seen that the VGA Module occupies pins on the bottom and right FPGA borders. In consequence, only a narrow part on the left side is available for dynamic module reconfiguration.

With these limitations in mind, we designed a new FPGA-based reconfigurable computer called the *Erlangen Slot Machine* (ESM). Its architecture circumvents all of the above problems and will be described next.

3 The Erlangen Slot Machine

The main idea of the Erlangen Slot Machine (ESM) architecture is to accelerate application development as well as research in the area of partially reconfigurable hardware. The advantage of the ESM platform is its unique slot-based architecture which allows the slots to be used independently of each other by delivering peripheral data through a separate crossbar switch as shown in Figure 3. We decided to spend an off-chip crossbar in order to have as many resources free on the FPGA for partially reconfigurable modules. The ESM architecture is based on the flexible decoupling of the FPGA I/O-pins from a direct connection to an interface chip. This flexibility allows the independent placement of application modules in any available slot at run-time. As a result, run-time placement is not constrained by physical I/O-pin locations as the the I/O-pin routing is done automatically in the crossbar, and the I/O pin dilemma is thus solved in hardware.

3.1 Architecture overview

The ESM platform (see Figure 3) is centered around an FPGA serving the main reconfigurable engine and an FPGA realizing the crossbar switch. They were separated into two physical boards (see Figure 12) called BabyBoard and MotherBoard and are implemented using a Xilinx Virtex-II 6000 and a Xilinx Spartan-II 600 FPGA. Figure 3 shows the slot-based architecture of the ESM consisting of the Virtex-II FPGA, local SRAM memories, configuration memory

and a reconfiguration manager. The top pins in the north of the FPGA connect to local SRAM banks. These SRAM banks thus solve the problem of restricted intra-module memory, in the case of video applications, for example. The bottom pins in the south connect to the crossbar switch. Therefore, a module can be placed in any free slot and have its own peripheral I/O-links together with dedicated local external memory. Each slot of up to 6 slots can access each a local SRAM bank.

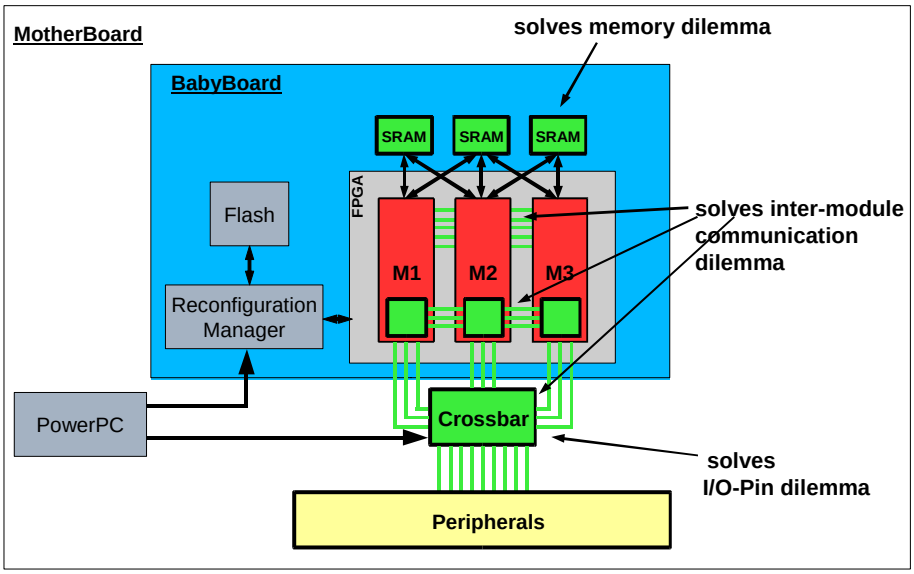

Fig. 3. ESM Architecture overview. The architecture of the BabyBoard is refined in Figure 4. The MotherBoard is shown in Figure 5.

3.2 The BabyBoard

Computation and Reconfigurable Engine The reconfigurable engine of the ESM computer is a printed circuit board that features a Xilinx Virtex II-6000 FPGA from Xilinx, several SRAMs and a reconfiguration manager implemented on another FPGA. Due to the restriction[1] in the reconfiguration of Virtex-II FPGAs, we adapted our architecture to match the following properties:

– **Solving the I/O-pin dilemma:** Run-time placement of modules on a reconfigurable device, in this case the FPGA, is done by downloading a partial bitstream that implements the module on the FPGA. This requires a relocation that places a module in a location different from the one for which it

[1] The reconfiguration can be done only in chunks of full columns.

was synthesized. Relocation can be done only if all the resources are available and structured in the same way in the designated placement area at compile-time. This includes also the I/O-pins used by the module. For example, a module compiled for slot 0 might then be allocated to slot 3 at run-time. We solved the I/O-pin dilemma on the ESM by avoiding fixed connections of peripherals to the FPGA. As shown in Figure 4, all the bottom pins from the FPGA are connected to an interface controller realizing a crossbar and implemented itself using a Xilinx Spartan-II FPGA. At run-time, it connects FPGA pins to peripherals automatically based on the slot position of a placed module. This I/O-pin rerouting principle is done without reconfiguration of the crossbar FPGA. This makes it possible to establish any connection from one module to peripherals dynamically.

– **Solving the memory dilemma:** Memory is very important in applications like video streaming in which a given module must exclusively access a picture at a time for computation. However, as we mentioned earlier, the capacity of the available BlockRAMs in FPGAs is limited. External SRAM memory is therefore added to allow storage of large amounts of data by each module. To allow a module to exclusively access its external memory bank, 6 SRAM banks are connected at the north border of the FPGA. In this way, a module will connect to peripherals from the south, while the north will be used for temporally storing computation data. According to the 6 memory banks which can be connected on the top, the device is divided into a set of elementary slots called *micro-slots* A to V (see Figure 4). In order to use an SRAM bank in the north, a module must have at least a width of three micro-slots (creating slots S1 to S6). The *Erlangen Slot Machine* owes its name from this arrangement of reconfigurable slots. This modular organization of the device simplifies the relocation, primary condition for a viable partially reconfigurable computing system. Each module moved from one slot to another will encounter equal resources. The architecture of the BabyBoard is illustrated in more detail in Figure 4.

The Reconfiguration Manager Apart from the main FPGA, the BabyBoard also contains the configuration circuitry. This consists of a CPLD, a configuration FPGA (a small Spartan II FPGA) implementing the reconfiguration management (Section 6) and a Flash, see Figure 4.

– The CPLD is used to download the Spartan-II configuration from the Flash upon power-up. It also contains board initialization routines for the on-board PLL and the Flash.
– The reconfiguration management is implemented on the Spartan-II FPGA. This device contains a circuit to perform module relocation while loading a new partial module bitstream. Its architecture and functionality will be described in details in Section 6.
– The Flash provides a capacity of 64 MBytes, thus enabling the storage of up to 32 full configurations or of a few hundred partial module bitstreams.

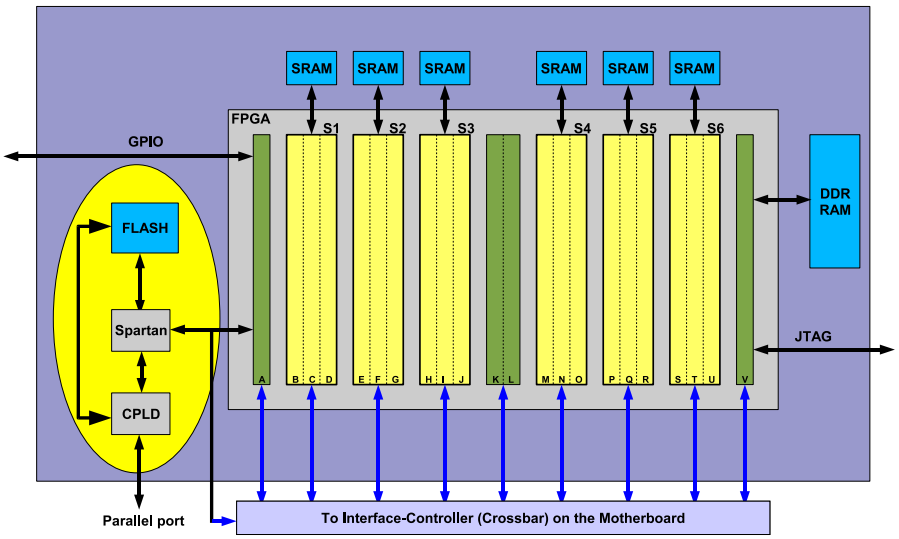

Fig. 4. Architecture of the ESM BabyBoard. Slots A to V denote micro-slots that provide the module and reconfiguration granularity. Three consecutive micro-slots define a macro-slot. Each macro-slot (S1 to S6) can access one full external SRAM bank. In terms of slice count, a micro slot occupies 1536 slices (4 CLB columns) on the FPGA. Slots A, K, L and V are special micro-slots as slots A and V interface external pins and slot K, L contain BlockRAM.

Memory Six SRAM banks of size 2 MB each are attached to the board on the north of the device, thus providing memory space to the six macro-slots (denoted as S1 to S6 in Figure 4) for temporal data storage. The SRAMs can be also used for shared memory communication between neighbor modules, e.g., for streaming applications. They are connected to the FPGA in such a way that the reconfiguration of a given module will not affect the access to other modules.

Debug Lines Debugging capabilities are offered through general purpose I/O provided at regular distances between the micro-slots. A JTAG port provides debug capabilities for the main FPGA, the CPLD and the Spartan-II.

3.3 The MotherBoard

The MotherBoard provides programmable links from the FPGA to all peripherals for multimedia and communication such as IEEE1394, USB, Ethernet, PCMCIA, Video and Audio-I/Os. The physical connections are established at run-time through a programmable crossbar implemented statically on a Spartan-II chip on the MotherBoard. Video capture and rendering interfaces as well as high speed communication links also exist on the MotherBoard on which the BabyBoard is mounted through four connectors (see Figure 12). A PowerPC

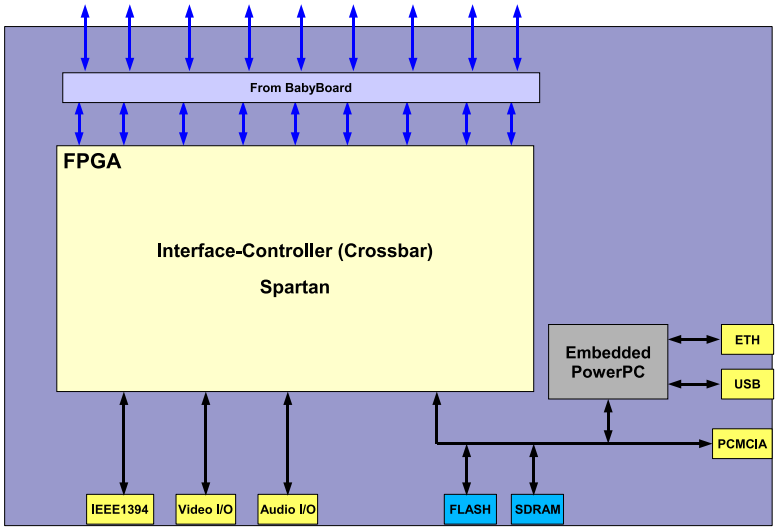

Fig. 5. Architecture of the ESM MotherBoard. The PowerPC is them main controller of the ESM system and running Linux. Its memory bus is connected directly to the crossbar for memory-mapped communication with the reconfiguration manager on the BabyBoard.

processor (MPC875) is the core of the MotherBoard. It is used to control the complete ESM. In particular, it manages the dataflow on the MotherBoard as well as the interfaces to the external world, e.g., Ethernet and USB. Upon start-up, one can log-in into the ESM just as for a full Linux-based computer system. The PowerPC of the ESM is used for application development or for testing and controlling the dynamic reconfiguration, e.g., operating system functions for module management.

3.4 Tool Flow

For an automated generation of partial reconfigurable modules and their communication infrastructure we are developing SlotComposer.

This tool is used for an automated communication and synthesis flow infrastructure generation for partially reconfigurable hardware modules. Using the existing Xilinx PR Tool Flow our SlotComposer inserts slice-based bus macros between adjacent modules. Based on designers specification SlotComposer connects partially reconfigurable modules to the Reconfigurable Multiple Bus (RMB) communication infrastructure or the crossbar. At the same time it generates all necessary constraint files and optimizes the usage and placement of bus macros and instantiates all intermediate communication signals. Moreover, all required scripts for the PR Flow are generated.

4 Inter-module Communication

One of the central limiting factors for the wide use of partial dynamic reconfiguration yet not addressed is the problem of inter-module communication. Each module that is placed on one or more slots on the device must be able to communicate with other modules. For the ESM, we provide four main paradigms for communication among different modules (see Figure 6): The first one is a direct communication using bus-macros [28] between adjacently placed modules (see Figure 6 a)). Secondly, shared memory communication using SRAMs or BlockRAMs is possible (see Figure 6 b)). However, only adjacent modules can use these two communication modes. For modules placed in non-adjacent slots, we provide a dynamic signal switching communication architecture called reconfigurable multiple bus (RMB) [29] (see Figure 6 c)). In [30] we have presented an ILP model for minimizing the communication cost for RMB slot modules. Finally, the communication between two different modules can also be realized through the external crossbar (see Figure 6 d)).

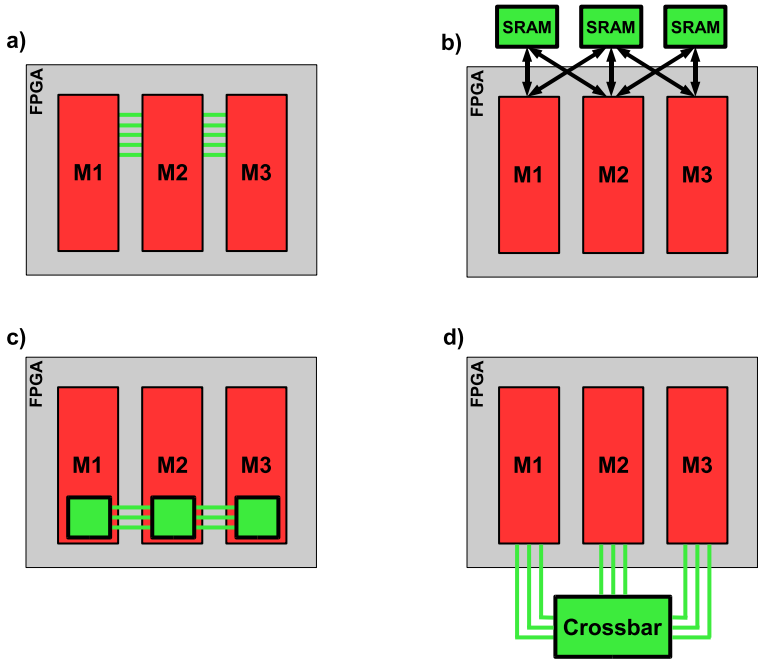

Fig. 6. Inter-module communication possibilities on the ESM: a) bus-macro, b) shared memory, c) reconfigurable multiple bus (RMB), d) external crossbar.

4.1 Communication between adjacent modules

On the ESM, bus-macros are used to realize a direct communication between adjacently placed modules, providing fixed communication channels that help to keep the signal integrity upon reconfiguration. Because eight signals can be passed for each bus-macro, the number of bus-macros needed for connecting a set of n signals between two placed modules is $n/8$.

4.2 Communication via Shared Memory

Communication between two neighboring modules can be done in two different ways using shared memory: First, dual-ported BlockRAMs can be used for implementing communication among two neighbor modules working in two different clock domains. The sender writes on one side, while the receiver reads the data on the other side. The second possibility uses external RAM. This is particular useful in applications in which each module must process a large amount of data and then sends the processed data to the next module, as it is the case in video streaming. On the ESM, each SRAM bank can be accessed by the module placed below as well as those neighbors placed right and left. A controller is used to manage the SRAM access. Depending on the application, the user may set the priority of accessing the SRAM for the three modules. In Section 5, we will present a video streaming case study that uses this way of communication.

4.3 Communication via RMB

In its basic definition, the Reconfigurable Multiple Bus (RMB) architecture [31–33] consists of a set of processing elements or modules, each possessing an access to a set of switched bus connections to other processing elements. The switches are controlled by connection requests between individual modules. The RMB is a one-dimensional arrangement of switches between N slots (see Figure 7). In our FPGA implementation, the horizontal arrangement of parallel switched bus line segments allows for the communication among modules placed in the individual slots. The request for a new connection is done in a wormhole fashion, where the sender (a module in slot S_k) sends a request for communication to its neighbor (slot S_{k+1}) in the direction of the receiver. Slot S_{k+1} sends the request to slot S_{k+2}, etc., until the receiver receives the request and returns an acknowledgment. The acknowledgment is then sent back in the same way to the sender. Each module that receives an acknowledgment sets its switch to connect two line segments. Upon receiving the acknowledgment, the sender can start the communication (circuit routing). The wired and latency-free connection is then active until an explicit release signal is issued by the sender module. The concept of an RMB was first presented in [32] and extended later in [31] with a compaction mechanism for quickly finding a free segment. However, it has never been implemented in real hardware.

Also, in our implementation [33] of the RMB on Xilinx Virtex FPGAs, we separated the RMB switches from the modules. In this way, we provide a uniform

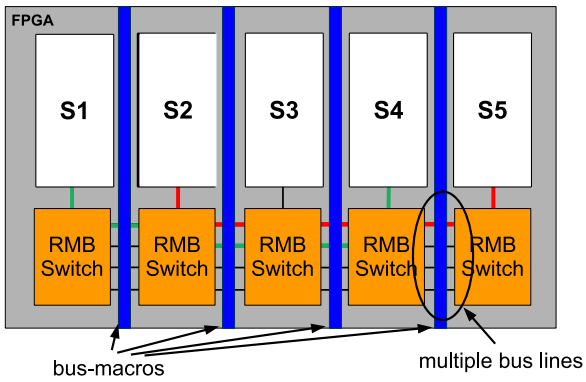

Fig. 7. FPGA implementation of the RMB for partial reconfiguration.

interface to designers for connecting modules to the multiple line switches. The implementation of the RMB structure on an FPGA Virtex II 6000 with four processors and four parallel 16 bit lines reveals an area overhead of 4% with a frequency of 120 MHz on the controller [29]. In [29], we have summarized area and data speed numbers in terms of a) different numbers of modules, b) different numbers of parallel bus segments, and c) bitwidths of each bus segment. As shown on Figure 7, bus-macros are used at the boundary of modules and controllers to insure a correct operation upon reconfiguration.

We were able to show that a module reconfiguration can take place column-wise at the same time that other modules are communicating on the chip without any signal interference. This is possible by storing the states of the RMB switches in regions of BlockRAM that are physically unaffected by partial reconfiguration.

4.4 Communication via the Crossbar

Another possibility of establishing a communication among modules is to use the crossbar. Because all the modules are connected to the crossbar via the pins at the south of the FPGA, the communication among two modules can be set in the crossbar as well.

4.5 Communication Costs

The ESM platform supports four different communication schemes. Each approach has its own properties, such as maximum bandwidth, signal delay and setup latency. The RMB is the only scheme that has a varying setup latency that is the product of the number of RMB elements to destination and the setup time of four clock cycles. Using bus macros for communication is the preferred choice, but it only works for adjacent modules. The maximum bandwidth in all communication schemes is a factor of clock speed and data bandwidth. In our experiments we assume for the ESM a global clock speed of 50 MHz. All properties are listed in Table 1.

Scheme	Data Bandwidth	Delay	Setup
BusMacro	19.2 Gbits/s	2 ns	none
RMB	6.4 Gbits/s	3 ns * CP	4 cycles * CP
Crossbar	1.8 Gbits/s	15 ns	18 cycles
SRAM	0.4 Gbits/s	20 ns	2 cycles

Table 1. Communication bandwidth and signal delay.

5 Case Study: Video and audio streaming

Video streaming can be defined as the process of performing computations on video data streams. Many video algorithms process the data stream picture-by-picture. Usually, a picture frame is transmitted pixel-by-pixel and therefore can be processed on a pixel-by-pixel basis. However, since a lot of algorithms require the neighborhood of a pixel, e.g., for filtering, often at least one complete frame must be stored and processed before the next one can be accessed. Capturing the neighborhood of a pixel is often done using a sliding window [11] the size of which varies according to the size of a neighbor region. A given set of buffers (FIFO) is used to update the window. The number of FIFOs varies according to the size of the window. In each step, a pixel is read from the memory and placed in the lower left cell of the window. Up to the upper right pixel which is disposed, i.e., output, all the pixels in the right part of the window are placed at the queue of the FIFO one level higher.
In the field of video compression, the processing is usually done in a block-by-block basis, different from the sliding window concept. However, the overall structure is almost the same.
As shown in Figure 8, the architecture of a video streaming system is usually built on a modular basis. The first module buffers with the image captured from an image source. This can be a camera or a network module which collects the picture data through a network channel, or any other source. The frames are alternately written to the SRAM banks RAM1 and RAM2 by the capture module. The second module collects the picture from RAM1 or RAM2 if this RAM module is not in use by the first module, builds the sliding windows and passes it to the third module which processes the pixel and saves it in its own memory or directly passes it to the next module. This architecture presents a pipelined computation in which the computational blocks are the modules that process the image frames. RAMs are used to temporally store frames between two modules, thus allowing a frame to stream from RAM to RAM and the processed pictures to the output.
An adaptive video streaming system is characterized by its ability to optimize the computation performed on the video stream according to changing environmental conditions. In most cases, only one module on the computation chain must be changed while the other keep running. The video capture module, for example, can be changed if we want to optimize the conversion of pixels to match the current brightness or the current landscape. It is also possible to

change the video source from camera to a new one with different characteristics. In an adaptive system, the functionality of a module on the computation path should be changed very fast without affecting the rest of the system. This can be done by providing some parameters to the module to instruct it to switch from one algorithm to the next one. However, the structures of the basic algorithms are not always the same. A Sobel filter [34], for example, cannot be changed into a Laplace filter by just changing the parameters. This is also true for a Median-operator which cannot be replaced by a Gauss-operator by just changing parameters. Network and camera require two different algorithms for capturing the pixels. In many cases, the complete module should be replaced by a module of the same size, but different in its structure while the rest of the system keeps running.

Our architecture fulfills the prerequisites for a modular pipelined and adaptive system for video streaming. In the system architecture presented before, we divided the device into slots, which each of them can implement a given module. RAMs are provided to the north of the device while the southern pins can be used by modules to communicate with the rest of the environment.

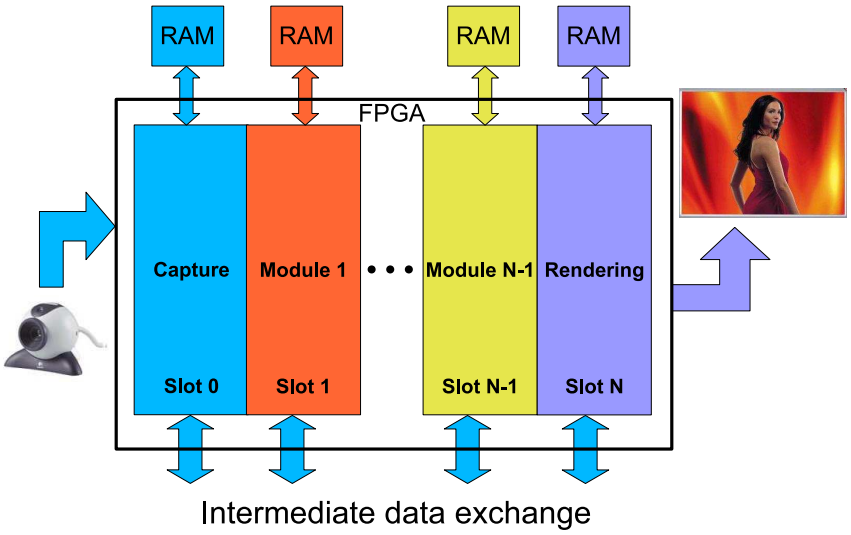

Fig. 8. A modular architecture for video streaming as implemented onto the slot-based structure of the ESM.

6 Reconfiguration manager

The ESM computer requires an operating system for the initialization of executable application modules and their run-time supervision. The main tasks

of such an operating system are a) scheduling of application modules, b) management of free slots including slot segmentation and partitoning, c) loading, unloading and relocation of application modules into slots, d) configuration of peripheral devices, e) configuration of the crossbar, and f) bitstream management.

In our view, the most-time critical operations must be executed in hardware in order to keep the reconfiguration time to a minimum. We consider the loading, unloading and relocation of modules to be the most time-critical tasks which will be therefore implemented in a dedicated hardware reconfiguration manager. All other system tasks can implemented in C and executed on the PowerPC embedded processor (see Figure 3) belonging to the ESM MotherBoard. These two parts of the operating system are linked via a simple communication bus as shown in Figure 3. This hardware/software interface is realized through a set of elementary reconfiguration instructions passed from the PowerPC to the reconfiguration manager on the BabyBoard using Memory-Mapped I/O. The reconfiguration manager must implement the following minimal set of elementary instructions[2]:

- LOAD (load bitstreams to their pre-compiled position)
- UNLOAD (unload bitstreams to deactivate a running module)
- RELOCATE_AND_LOAD (relocate bitstreams to a different slot position before loading)

Therefore, the reconfiguration manager was built in hardware and located in a Spartan-II 400 FPGA which is connected to the main FPGA via the SelectMAP interface.

During normal operation, the bitstream data will be loaded from the flash memory located on the BabyBoard (see Figure 4). However, bitstreams must be first downloaded and stored into the flash memory. Here, two methods are supported. The first method uses a parallel port interface implemented directly in the reconfiguration manager to download the configuration data from a host PC to the flash memory. The second method uses the Ethernet port of the PowerPC processor on the MotherBoard to download bitstreams from a remote host. In order to support these and also many other reconfiguration scenarios, we developed a very flexible, plug-in based reconfiguration manager architecture.

6.1 Flexible plug-in architecture

Our first implementation had a data block-oriented reconfiguration manager and consisted of a simple state machine which controlled all interfaces and operated on byte blocks. These data blocks of size 512 bytes each correspond to the page size of the flash memory device. For each primitive operation on a data block, an instruction had to be processed. When one data block was written

[2] The detailed and full instruction set syntax and syntax will be described in a forthcoming publication

from flash into the Virtex-II SelectMAP interface, two instructions had to be processed. First, the data block was read in 512 cycles from the flash device and written to an internal scratch pad. Then, the second instruction was read and the data block from the scratch pad was written to the SelectMAP interface. As all instructions were executed sequentially, the maximum upload speed of a bitstream to the FPGA was slowed down by factor two, due to the exclusive access to the scratch pad.

However, the main problem with this architecture arose when more plug-ins and extensions were to be added to the reconfiguration manager. If for example, an error correcting code (ECC) plug-in and a decompression plug-in are used additionally, then the speed degradation will increase to a factor of six, because four additional instructions are needed for reading and writing the scratch pad. This initial scenario is illustrated in Figure 9 b. An additional maintenance issue is the global finite state machine itself. Its code base had to be changed every time a new plug-in was added or removed.

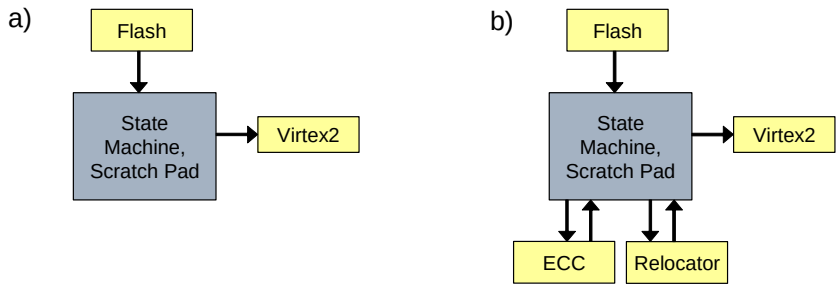

Fig. 9. Simple reconfiguration manager architecture.

Clearly, this first data block-oriented architecture is not suitable for a high performance solution, since the throughput decreases with every new attached plug-in. The main bottleneck is not the flash interface but the scratch pad-oriented data flow combined with the sequential execution of the instructions.

Based on these consolidated findings, we propose a novel architecture for the reconfiguration manager here which can upload the bitstreams into the FPGA at the speed of the flash interface. The central scratch pad was eliminated and replaced by a pipelined data flow architecture. Moreover, a) the finite state machine was replaced by a MicroBlaze microcontroller [19], and b) a data crossbar switch is employed between the plug-ins. This new architecture is depicted in Figure 10. The crossbar plug-in shown in this figure connects the reconfiguration manager control implemented now in software on the MicroBlaze to the ESM MotherBoard in order to establish the communication link to the PowerPC shown in Figure 3.

All plug-in modules are connected to two communication interfaces: The control bus connects plug-ins to the MicroBlaze for initialization and control. The

data crossbar connects to the data input and output ports of each plug-in and its connection setup is also controlled by the MicroBlaze which is programmed in assembly language.

In order to upload a hardware module from flash to the FPGA, the following sequence of steps has to be performed:

- Command is sent to the MicroBlaze to upload a bitstream to the FPGA without the use of any other plug-ins.
- Program running on the MicroBlaze connects the output of the flash plug-in to the Virtex-II plug-in input through a write into the configuration register of the data crossbar.
- Next, this program initializes the flash plug-in with the start address and length of the bitstream.
- Then, the program enables the SelectMAP interface in the Virtex-II plug-in.
- Finally, the flash plug-in is enabled and starts to read the bitstream.
- The flash plug-in sends the bitstream to the Virtex-II plug-in byte by byte as long as its ready signal is true (if not, the flash plug-in has to wait).
- While the flash and the Virtex-II plug-in are running in parallel, the MicroBlaze checks periodically if any of the plug-ins has finished its operation.
- Only if after finishing one command, the MicroBlaze can execute a new command, and, for example, reinitializes the plug-ins and the data crossbar.

If one load command has been executed and another load follows, then the procedure starts from second step, because the data crossbar has already been set. The addition of plug-ins to the reconfiguration manager is simple. Any new module must have a fixed control bus interface and a fixed data crossbar interface. With these standard interfaces, the plug-in can be directly controlled through the MicroBlaze assembly program. The data crossbar uses a parametrized HDL description which can be configured at design-time to the number of actually instantiated plug-ins.

6.2 Workload scenarios

Depending on the operating system requirements, different operations need to be performed on each bitstream. Before the bitstream is uploaded to the FPGA, it can pass through any number of additional plug-ins. The order in which a bitstream passes the plug-ins is configurable at run-time through the setup of the data crossbar switch. This allows a flexible preprocessing of the bitstream prior to being loaded. Only the number of available plug-ins in the reconfiguration manager has to be determined at design-time.

Based on the introduced reconfiguration manager architecture from Figure 10, several flows are possible. Some of these are depicted in Figure 11. In the first scenario, only a basic upload of a bitstream is performed. Therefore, the data flows from the flash plug-in output directly through the data crossbar to the Virtex-II plug-in input. If an error-correction is needed, then the flash output data can be sent to the ECC plug-in before going to the Virtex-II plug-in. This

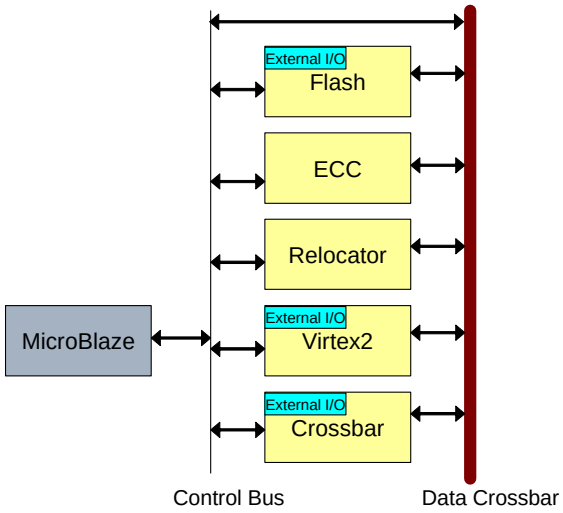

Fig. 10. Architecture of the ESM reconfiguration manager with plug-ins such as Flash, ECC, module relocator and other possible plug-ins.

case is shown in Figure 11 b). In the third scenario, the bitstream is read from the flash, error-corrected and relocated before being sent to the Virtex-II plug-in for upload (see Figure 11 c)). Here, the crossbar is configured by the microprozessor in such a way that the output of each plug-in is connected to the input of its neighboring plug-in. The fourth scenario depicted in Figure 11 d) shows how the bitstream data is delivered by the PowerPC through the MotherBoard crossbar. The bitstream is subsequently error-corrected and relocated prior to its upload.

The plug-ins that are currently implemented for the reconfiguration manager are: ECC plug-in, decompression plug-in and a relocator plug-in which can translate a bitstream on the fly to any slot location on the FPGA by directly manipulating the address offsets in the bitstream at load-time.

6.3 Implementation results

The reconfiguration manager was implemented and consists of the MicroBlaze microcontroller, parallel port interface plug-in, flash memory interface plug-in, VirtexII SelectMAP plug-in, an OPB (on-chip peripheral bus) interface implementing the control bus and the data crossbar. The control bus is a 32 bit OPB bus, while the data crossbar is an 8 bit full duplex crossbar.

The flash plug-in interface is able to sustain a data rate of 10 Mbyte/s in a conservative and tested timing setup. As the SelectMAP interface can upload bitstreams at a rate of 50 MByte/s, an additional decompression plug-in would accelerate the reconfiguration time when used on compressed bitstreams.

The final board implementation of the BabyBoard and MotherBoard is shown in Figure 12. The reconfiguration manager is implemented in the Spartan-II 400

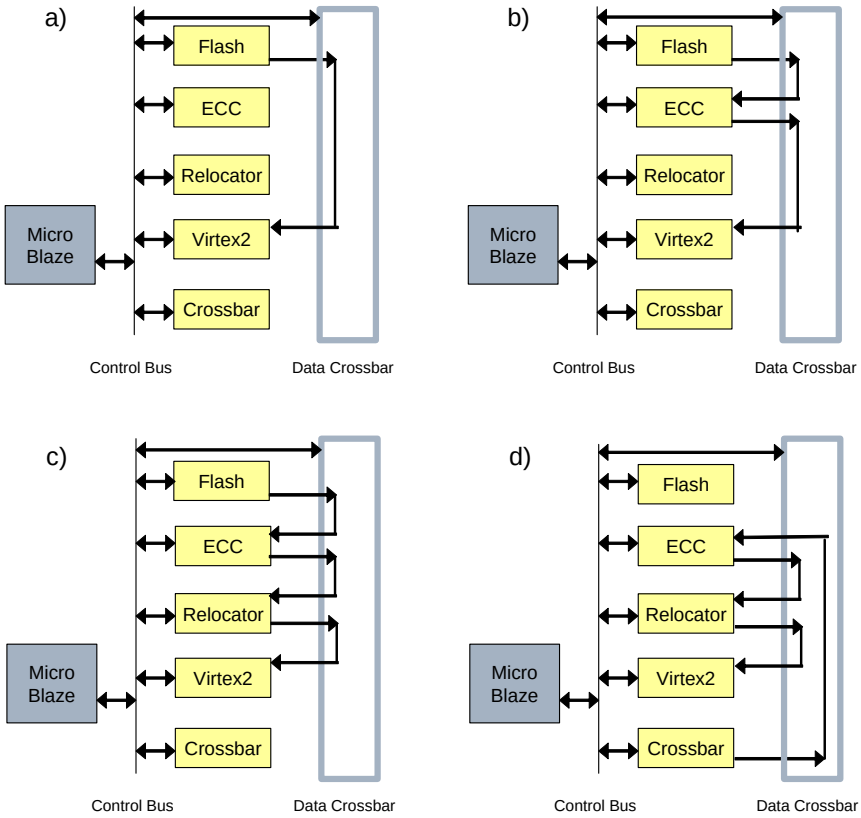

Fig. 11. Four different workload scenarios for the reconfiguration manager.

FPGA which is located close to the 64 MByte flash device and the main Virtex-II 6000 FPGA. Technical data sheets as well as software, primer applications, and user information is available at http://www.r-space.de.

The separation of BabyBoard and MotherBoard was made in order to customize the ESM architecture to other application domains such as automotive. In order to do so, a new MotherBoard could be designed to have different peripherals such as CAN, LIN, FlexRay controllers, and A/D and D/A converters.

7 Conclusions

We have presented a new dynamically reconfigurable computer architecture called Erlangen Slot Machine (ESM) that was built for reasons that many brilliant ideas for reconfigurable computers and for dynamic resource management cannot be efficiently and directly transfered using currently available technology, mainly because of I/O-pins, memory, and inter-module communication panaceas.

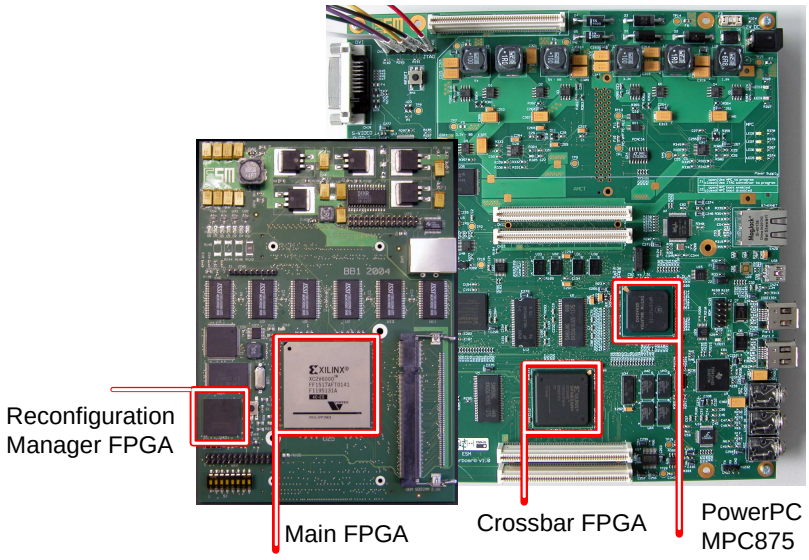

Fig. 12. Implementation of the ESM BabyBoard and MotherBoard. Technical data sheets are available at http://www.r-space.de.

The ESM is a stand-alone reconfigurable computer trying to bridge this gap by providing a) new architectural concepts to avoid the above physical problems and restrictions, b) new inter-module communication concepts, as well as c) an intelligent module reconfiguration management.

We expect this architecture to serve as a prototyping platform for reconfigurable hardware development with respect to application-development and operating system implementation for reconfigurable module management (i.e., placement and scheduling). A small series of 15 systems is currently manufactured to serve projects in the German Priority Program SPP1148 sponsored by the German Science Foundation as a prototyping platform.

Acknowledgments

Josef Angermeier, Jan Grembler, Felix Reimann, Thomas Haller, André Linarth, Peter Asemann, Christian Freiberger, Christoph Lauer and Dirk Koch have all helped with the design and implementation of the ESM hardware and software.

References

1. Schaumont, P., Verbauwhede, I., Keutzer, K., Sarrafzadeh, M.: A quick safari through the reconfiguration jungle. In: Proceedings of the 38th Design Automation Conference, Las Vegas (2001) 127–177

2. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36** (2003) 41–52
3. IBM Autonomic Computing Initiative. (http://www.research.ibm.com/autonomic/)
4. Müller-Schloer, C., von der Malsburg, C., Würtz, R.P.: Organic computing. Informatik Spektrum **27** (2004) 332–336
5. The Organic Computing Page. (http://www.organic-computing.org)
6. SPP1148 Reconfigurable Computing Priority Program. (http://www12.informatik.uni-erlangen.de/spprr/)
7. Ahmadinia, A., Bobda, C., Fekete, S., Teich, J., van der Veen, J.: Optimal routing-conscious dynamic placement for reconfigurable devices. In: Proceedings of International Conference on Field-Programmable Logic and Applications. Volume 3203 of Lecture Notes in Computer Science (LNCS)., Antwerp, Belgium, Springer (2004) 847–851
8. Bazargan, K., Kastner, R., Sarrafzadeh, M.: Fast template placement for reconfigurable computing systems. IEEE Design and Test of Computers **17** (2000) 68–83
9. Majer, M., Teich, J., Bobda, C.: ESM - the Erlangen Slot Machine. (http://www.r-space.de)
10. Bobda, C., Majer, M., Ahmadinia, A., Haller, T., Linarth, A., Teich, J., Fekete, S.P., van der Veen, J.: The Erlangen Slot Machine: A highly flexible FPGA-based reconfigurable platform. In: Proceeding IEEE Symposium on Field-Programmable Custom Computing Machines. (2005) 319–320
11. Bobda, C., Ahmadinia, A., Majer, M., Ding, J., Teich, J.: Modular video streaming on a reconfigurable platform. In: Proceedings of the IFIP International Conference on Very Large Scale Integration, Perth, Australia (2005) 103–108
12. Bobda, C., Majer, M., Ahmadinia, A., Haller, T., Linarth, A., Teich, J.: Increasing the flexibility in FPGA-based reconfigurable platforms: The Erlangen Slot Machine. In: Proceedings of the IEEE Conference on Field-Programmable Technology, Singapore, Singapore (2005) 37–42
13. Krasteva, Y., Jimeno, A., Torre, E., Riesgo, T.: Straight method for reallocation of complex cores by dynamic reconfiguration in Virtex II FPGAs. In: Proceedings of the 16th IEEE International Workshop on Rapid System Prototyping, Montreal, Canada (2005) 77–83
14. Baumgarte, V., May, F., Nückel, A., Vorbach, M., Weinhardt, M.: PACT XPP - a self-reconfigurable data processing architecture. In: ERSA, Las Vegas, Nevada (2001) 167–184
15. Silicon Hive. (http://www.siliconhive.com)
16. PicoChip. (http://www.picochip.com)
17. Elixent Ltd. (http://www.elixent.com)
18. NEC DRP Project. (http://www.necel.com/en/techhighlights/drp/)
19. Xilinx, Inc. (http://www.xilinx.com)
20. Celoxica Ltd.: RC2000 Development Board. (2004)
21. Xess Corp. (http://www.xess.com)
22. Nallatech, Inc. (http://www.nallatech.com)
23. Alpha Data Ltd.: ADM-XRC-II Xilinx Virtex-II PMC. (2002)
24. Platzner, M., Thiele, L.: XFORCES - executives for reconfigurable embedded systems. (http://www.ee.ethz.ch/~platzner)
25. Steiger, C., Walder, H., Platzner, M., Thiele, L.: Online scheduling and placement of real-time tasks to partially reconfigurable devices. In: Proceedings of the 24th International Real-Time Systems Symposium, Cancun, Mexico (2003) 224–235

26. Kalte, H., Porrmann, M., Rückert, U.: A prototyping platform for dynamically reconfigurable system on chip designs. In: Proceedings of the IEEE Workshop Heterogeneous reconfigurable Systems on Chip (SoC), Hamburg, Germany (2002)
27. Bobda, C., Ahmadinia, A., Majer, M., Teich, J., Fekete, S., van der Veen, J.: DyNoC: A dynamic infrastructure for communication in dynamically reconfigurable devices. In: Proceedings of the International Conference on Field-Programmable Logic and Applications, Tampere, Finland (2005) 153–158
28. Lysaght, P., Blodge, B., Mason, J., Young, J., Bridgeford, B.: Enhanced architectures, design methodologies and cad tools for dynamic reconfiguration of Xilinx FPGAs. In: Proceedings of 16th International Conference on Field Programmable Logic and Applications (FPL06), Madrid, Spain (2006)
29. Ahmadinia, A., Ding, J., Bobda, C., Teich, J.: Design and implementation of reconfigurable multiple bus on chip (RMBoC). Technical Report 02-2004, University of Erlangen-Nuremberg, Department of CS 12, Hardware-Software-Co-Design (2004)
30. Fekete, S., van der Veen, J., Majer, M., Teich, J.: Minimizing communication cost for reconfigurable slot modules. In: Proceedings of 16th International Conference on Field Programmable Logic and Applications (FPL06), Madrid, Spain (2006)
31. ElGindy, H.A., Somani, A.K., Schröder, H., Schmeck, H., Spray, A.: RMB - a reconfigurable multiple bus network. In: Proceedings of the Second International Symposium on High-Performance Computer Architecture (HPCA-2), San Jose, California (1996) 108–117
32. Vaidyanathan, R., Trahan, J.L.: Dynamic Reconfiguration: Architectures and Algorithms. IEEE Computer Society (2003)
33. Ahmadinia, A., Bobda, C., Ding, J., Majer, M., Teich, J., Fekete, S., van der Veen, J.: A practical approach for circuit routing on dynamic reconfigurable devices. In: Proceedings of the 16th IEEE International Workshop on Rapid System Prototyping, Montreal, Canada (2005) 84–90
34. Rafael Gonzalez and Richard Woods: Digital Image Processing. Prentice Hall (2002)