# RECOBUS-BUILDER – A NOVEL TOOL AND TECHNIQUE TO BUILD STATICALLY AND DYNAMICALLY RECONFIGURABLE SYSTEMS FOR FPGAS

*Dirk Koch, Christian Beckhoff, and Jürgen Teich*

Hardware/Software Co-Design, Department of Computer Science
University of Erlangen-Nuremberg, Germany
email{dirk.koch, teich}@cs.fau.de

## ABSTRACT

In this paper, we present the ReCoBus-Builder tool chain that simplifies the generation of dynamically reconfigurable systems to almost a push-button process. The generated systems provide one or more resource areas that will be used by different partially reconfigurable modules at runtime. It is possible to integrate multiple partially reconfigurable modules into the same resource area at the same time and these modules can communicate via a fixed bus infrastructure or dedicated point-to-point links with other parts of the system. This allows building encapsulated modules that will be integrated into the system by linking together bitstreams at runtime. We will demonstrate that bitstream linking can further be used to speed up the design process of static only systems by eliminating long synthesis runs or place and route steps, when only small portions of a design are exchanged.

## 1. INTRODUCTION

Partial runtime reconfiguration allows to reduce monetary cost and power consumption by using some FPGA resources for different modules with mutually exclusive functionality over time. However, runtime reconfiguration is still exotic in commercial systems, and it comes along with a resource overhead for providing a configuration interface and a communication infrastructure that is suitable to integrate modules into a system by partial reconfiguration at runtime. Furthermore, there is a lack of adequate design tools to support the generation of systems that are capable to use partial reconfiguration as an integrated feature of the system.

In this paper, we will contribute the design of runtime reconfigurable FPGA based systems with a novel technique for providing point-to-point communication links between the static part of a system and some reconfigurable modules. We named this technique *I/O bars*. I/O bars allow to establish point-to-point links for reconfigurable modules in a very flexible manner at runtime. Furthermore, we will present our tool *ReCoBus-Builder* automating all additional design steps required to build systems supporting partial runtime reconfiguration.

The paper will continue with an overview of the communication infrastructure upon which the ReCoBus-Builder will integrate modules to a system. Next, in Section 3 we will present the ReCoBus-Builder tool chain step by step. An example system demonstrating the capabilities of our proposed techniques is revealed in Section 4. Finally, we will conclude the paper in Section 5.

## 2. COMMUNICATION INFRASTRUCTURE FOR DYNAMICALLY RECONFIGURABLE SYSTEMS

If we want to build systems with the capability to integrate multiple modules of different size in a flexible manner, we require sophisticated communication structures. A lot of related practical work assumes that only one resource area or multiple resource areas are provided by the system and modules must fit into such a resource area [1, 2]. In contrast to this model, we want to partition the complete reconfigurable area with a fine grid in order to place modules with different requirements efficiently to the device.

The communication in most typically FPGA based systems can be classified in shared memory communication over *buses* or dedicated *point-to-point links*. Newer techniques like networks on a chip [3] have not yet demonstrated to be efficiently implemented on present FPGAs.

This work is based on a communication infrastructure that provides buses as well as point-to-point links for integrating partially reconfigurable modules at runtime. This allows to exchange modules in a reconfigurable system offering the following main features:

i) *Direct interfacing*: Reconfigurable modules have a direct interface to the on-chip memory bus, other modules, or I/O pins.
ii) *Module relocation*: Modules can be placed freely within the span of the reconfigurable resource area.
iii) *Flexible widths*: Modules of different sizes can be connected to the system with a *fine module grid*.
iv) *Multiple instances* of the same module are allowed.
v) *Low logic overhead* to implement the communication infrastructure.
vi) *High performance*: The communication bandwidth and the latency can compete with static only systems.

### 2.1. Generic FPGA Architecture Model

Before we can understand the specials of a communication infrastructure for reconfigurable systems, we have to look closer at the underlying FPGA architecture (see Figure 1).

978-1-4244-1961-6/08/$25.00 

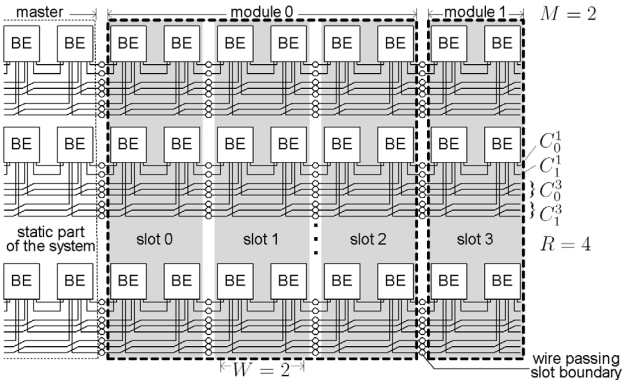


**Fig. 1**. Simplified FPGA architecture model.

An FPGA consists of a two dimensional regular grid of *basic elements* (BE) containing the switch and logic resources. The logic is typically based on one or a few clustered look-up tables (LUTs) inside the BEs. Wires of different span ($S$) between the BEs carry out the distribution of signals. For each span, there exists in each BE a number of output wires (counted by $i$) per span forming a *channel* $C_i^S$. To simplify the figure, we illustrated only some horizontal routing wires. The wires may be arranged in any desired horizontal or vertical fashion. Onto such an architecture, we will map our communication infrastructure in a regular manner. The system offers $R$ resource slots that are each $W$ BEs wide. Modules can have different sizes that are a multiple of $W$ BEs wide. Up to $M$ modules may be connected to the bus at the same time. Beside the resource slots, the reconfigurable SoC will contain a static part that may control the bus. The static part can also communicate with the modules connected to the bus.
For Xilinx FPGAs that are best suitable to implement complex dynamically reconfigurable systems, the BEs are called CLBs (configurable logic block) that cluster four so called slices, where each slice contains two look-up tables. The Xilinx Virtex-II and Spartan-3 families provide channels with a span $S$ of two (*double lines*) and six (*hex lines*).

### 2.2. Buses for Runtime Reconfigurable Systems

When building a communication infrastructure for dynamically reconfigurable systems, we have to ensure that a specific signal is routed within the same channel along the complete reconfigurable area and that it is always assigned to the same channel index $i$ when crossing its resource slot boundary. This ensures that modules can access this specific signal regardless to the present resource slot position. Inside the resource slot, we have to ensure that the complete routing and also the logic is designed completely uniformed. Figure 2 reveals the basic concept to build such a regular structure for a read data line.

Buses following these design rules have been presented in [4] and [5]. The latter work describes the so-called ReCoBus that can provide a resource slots with a grid as fine as one CLB column wide, when implemented on a Xilinx Virtex-II or Spartan-3 FPGA. In addition, a ReCoBus has only little resource overhead as compared to a traditional static only system and the low latency of the bus allows high throughput. The ReBoBus-Builder is a tool for building such a ReCoBus and it further supports the integration of the dynamic partial resources into the static part of the system.

### 2.3. I/O Bars

Beside the communication over the ReCoBus, reconfigurable modules may require links to I/O pins or point-to-point connections to other modules in the system or within the reconfigurable part of the system. Related approaches [1, 6] are either not suitable to connect modules in a fine grid or they suffer a massive resource overhead. As we want to build the resource slots as narrow as possible for increasing the placement flexibility, we can not tolerate to waste logic in a resource slot that will not require a connection. We solved this issue by reserving a horizontal set of wires that are aligned in parallel to the ReCoBus. These wires are routed in a similar regular fashion as the ReCoBus itself. We call such a set of wires an *I/O bar*.

Only at the resource slots that require a connection to a particular bar, we will instantiate additional logic to provide connection terminals to the modules that have to access this bar. An example for an I/O bar is illustrated in Figure 3. Note, that we are able to simply route through the incoming bar wires or that we can cut the routing track in order to allow a module to read the incoming data, process the data, and send the results further towards the bar. Consequently, the bar technique is ideal for signal processing with streaming data.

The I/O bar approach is very efficient to implement on Xilinx FPGAs as shown in Figure 4. By using the look-up tables and the separate usable flip-flops, we are able to provide up to eight module input ports together with eight module output ports within the same CLB. Note that we provide the double amount of connection terminals as compared to the so called *bus macros* as proposed by Xilinx [7].

## 3. RECOBUS-BUILDER DESIGN FLOW

We built the ReCoBus-Builder tool to simplify all additional design steps required to implement runtime reconfigurable

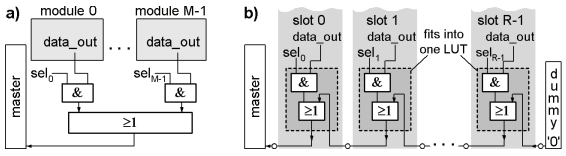


**Fig. 2**. a) Common implementation of a master read data signal in a multiplexer-based bus. b) *Distributed read multiplexer chain* suitable for partially reconfigurable systems. Only one module or resource slot is allowed to be selected at a point of time.

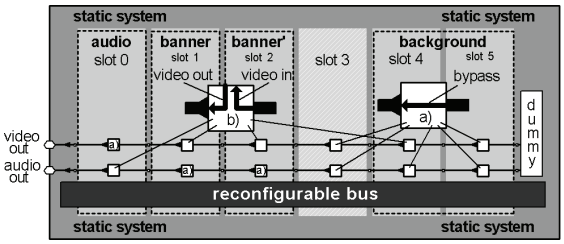


**Fig. 3**. Examplary system with an `audio` module that is connected in slot 0 to the audio bar and three video modules (`banner`, `banner'`, and `background`) that are connected in the slots 1, 2, and 4 to the video bar. `Banner` and `banner'` are two instances of the same module that manipulate the incoming video data and send the results further to the left hand side towards the video output pins.

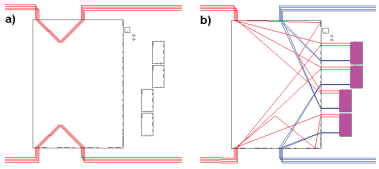


**Fig. 4**. FPGA-Editor view of an I/O bar transferring eight signals from right to left. a) Bar connection point in route through mode. b) Bar connection by cutting the routing track into an incoming and an outgoing part.

systems on FPGAs. In this section, we will guide through all steps of our flow that is illustrated in Figure 5.

### 3.1. Communication Protocol Specification

Before starting the entire design process, we have to define a physical specification of the bus protocol. In particular, this specification includes the data word width, the address range, the maximum number of interrupt lines, the amount of control signals, and the maximum number of master and slave modules connected to the reconfigurable bus. In addition, we have to define the signal levels of the control signals and the timing of all bus signals. A ReCoBus is able to implement all established bus protocols including AMBA, CoreConnect, AVALON, or Wishbone. The bus may contain pipeline registers in order to enhance the throughput. If the system provides I/O bars for the communication of the reconfigurable modules with other modules or I/O pins, we also have to specify the protocols for all these I/O bars.

A physical specification of a ReCoBus or some I/O bars can be reused by multiple designs. Such specifications are independent to the target FPGA, and the same specification may be used to build systems for different FPGA families. Currently we support all Xilinx Virtex-II /II Pro FPGAs and also all Spartan-3 devices. The ReCoBus specification process itself is supported by a GUI inside the ReCoBus-Builder tool and the process can be carried out within a few minutes. After this, we can automatically build an RTL model and according VHDL instantiation templates of the ReCoBus and the I/O bars. This model allows to simulate the complete system including the ReCoBus and I/O bars

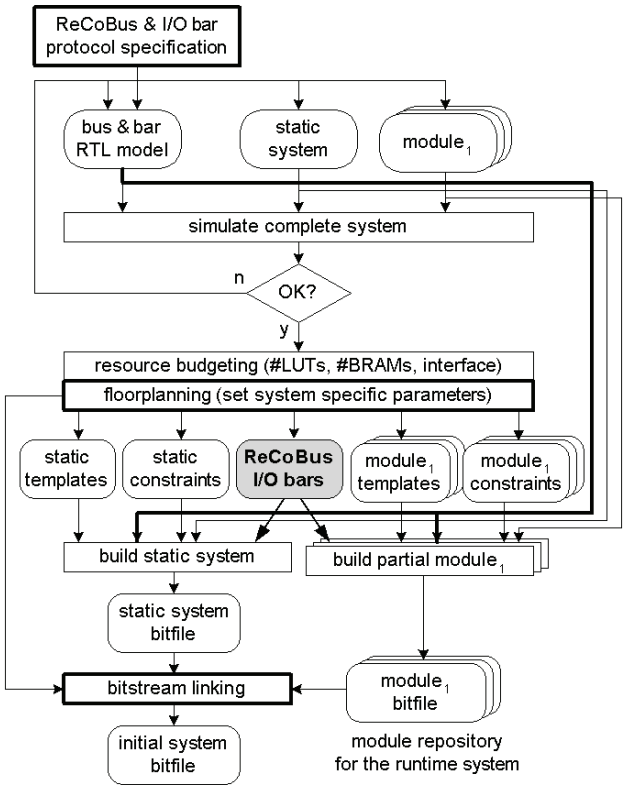


**Fig. 5**. ReCoBus-Builder design flow. The processes in the fat boxes are performed by our tools, while all other symulation, synthesis and place and route steps are based on external tools.

that link all partial reconfigurable modules within the system. Note that the simulation typically contains more modules (or instances of some modules) as the final system at runtime where modules share FPGA resources over time. Due to space limitations, we have to omit a serious discussion about the simulation of runtime reconfigurable systems.

### 3.2. Floorplanning

If the system simulation passes all tests, we can run a preliminary synthesis step in order to budget the resource requirements for the particular reconfigurable modules. Based on this initial resource budgeting, we start the floorplanning step in which we select the target FPGA and where we define the bounding boxes for the individual partially reconfigurable modules. The bounding boxes must be set to fulfill all logic (look-up tables) and memory (block RAMs) requirements. In addition, the bounding box must provide sufficient resource slots at the ReCoBus in order to connect all module bus signals to the ReCoBus. Note that a ReCoBus allows to take a benefit if small modules also contain small interface sizes (lower width of address or data signals).

The floorplanning phase is the most important step when building partially reconfigurable systems. In this step, the

system is split into a static part and a dynamically reconfigurable part containing the FPGA resources that are shared by multiple modules over time. Furthermore, the communication infrastructure between these parts is set to fixed positions. This allows that the individual reconfigurable modules can be built in also individual synthesis processes, and the according place and route steps are forced to maintain exactly the same resources for the module-to-module communication or the links to some I/O pins. Note that the design process of different modules can be easily delegated to multiple design teams that can work in parallel.

All these processes are also provided by the PlanAhead tool [8] distributed by Xilinx Inc.. However, PlanAhead (in combination with the present partial design flow [7]) can only assign one module exclusively to one reconfigurable area within the FPGA at a point of time. Thus, it is not simply possible to replace one large module by multiple small ones within the same reconfigurable area at runtime. The main reason for this restriction is a lack of sophisticated communication techniques that provide a flexible communication infrastructure for dynamically partial reconfigurable systems. PlanAhead provides only point-to-point communication between a static part and a dynamically reconfigurable part of the system by so called *bus macros*[1]. A bus macro generator that creates macros compatible with PlanAhead is presented in [9].

Opposed to this, the ReCoBus-Builder uses the flexible ReCoBus and I/O bar communication infrastructure that allows to integrate multiple reconfigurable modules into one shared area at runtime. A ReCoBus can be parameterized to provide alternatively only slave ports or both, master and slave ports. The shared reconfigurable area is divided into multiple small chunks called resource slots. It is possible to specify the height $H$, width $W$, and number $S$ of the resource slots in terms of CLBs. Beside the position of the ReCoBus and the I/O bars, we can optionally select the routing and logic resources used to implement the ReCoBus and the I/O bars. This allows to influence the density of the module interfaces in terms of bus signals per CLB. The density has impact on the place and route process. However, we found no congestion problems if we limit the density to less than one half of the available logic resources within a CLB. All these setting can be adjusted comfortable in the ReCoBus-Builder GUI as revealed in Figure 6.

### 3.3. ReCoBus Generation

After specifying the system specific parameters, the ReCoBus-Builder generates a so called *hard macro* containing the complete logic and routing of the ReCoBus that provides the communication between the static part of the system and the reconfigurable modules. A hard macro is a module containing logic, placement, and routing information that will be maintained if the macro is instantiated in a design. Hard-

[1]The name *bus macro* is misleading, as these macros provide only point-to-point links. We use the phrase *ReCoBus* for our monolithic macro containing a real bus with connections to multiple (reconfigurable) modules at the same time. In addition, we named our flexible technique providing multiple point-to-point connections with the phrase *I/O bar*.

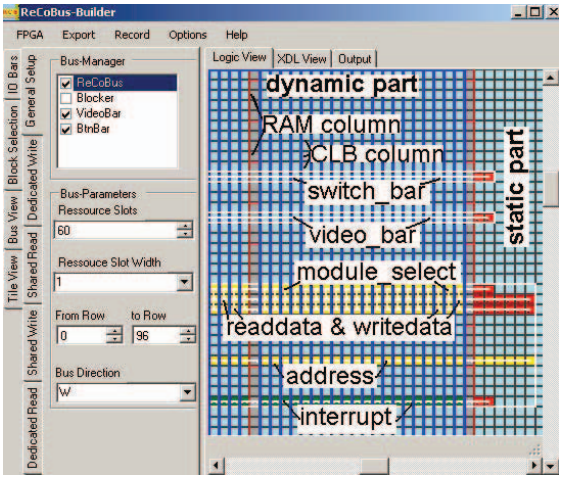


**Fig. 6**. ReCoBus-Builder GUI.

macros require no logic synthesis step and will be placed on the FPGA prior to the final place and route step that connects the hard macro with the rest of the design. The complete hard macro is generated by the use of the *Xilinx design language* (XDL) that gives designers among other low level details full access to all FPGA logic resources (e.g., the look-up tables) and all routing wires available in a particular Xilinx FPGA. In addition, we use an FPGA device description specified in the XDL language to read in all device specific information of the target FPGA that is required to build a ReCoBus hard macro. For instance, we automatically derive the positions of the block RAM columns in order to generate regular structured hard macros even if the reconfigurable area contains RAM columns. Note that partially reconfigurable modules may use such RAM columns. Another example for device specific information that we derive from the XDL device description is the set of available routing wires provided by a particular FPGA. We found that available routing wires vary even within the same FPGA family. For instance, the largest Xilinx Spartan-3-FPGA (XC3S5000) provides so called 'hex lines' within all CLBs that are not available in the CLBs of the smallest device (XC3S50) within this device family. The ReCoBus-Builder considers all this information in order to build an optimized ReCoBus hard macro.

We say that a ReCoBus macro is optimized if it uses the minimum amount of slices within the resource slots that are connected with minimum routing cost. Thus we pack as much logic inside the slices as possible to provide the bus logic and the connection to the reconfigurable modules. For instance, we may use both look-up tables inside the slices for building the logic of a resource slot for two read multiplexer chains (for two bits) of the read data bus in one slice (see also Figure 4). Independently, we can use the two additional available flip-flops to provide a connection for two write data signals from the static part of the system to the reconfigurable modules connected in the same slot. With this packing, we can provide the double amount of signals per slice as compared to the slice macros proposed in the Xilinx partial design flow [7]

A ReCoBus is built by firstly placing the slices according to the desired resource slot width and specified density within the area assigned for the resource slots. The slices are placed in a regular manner along the resource slots such that their relative positions are the same for all slots. Then, we use a breadth-first search in order to find the routing paths between two consecutive resource slots and the internal routing within a resource slot. We will continue the search in order to find all possible paths that require just the minimum number of wires between the allocated slices. We will then choose the path leading to the lowest propagation delay. In the case that multiple paths have the same delay, we will select the path containing wires providing the lowest connection grade in order to leave the wires with higher connection grade to implement the partially reconfigurable modules. The ReCoBus-Builder is capable to solve routing conflicts that may occur if the ReCoBus is densely built. The paths found between the first two resource slots and inside the resource slots are then copied to all other resource slots in order to provide a homogenous routing over all resource slots and the borders to the static part of the system. Finally the slice logic is set according to the determined routing and an interface is generated for the static part of the system. The complete ReCoBus hard macro generation runs automatically as a push button process.

Beside the ReCoBus hard macro, the ReCoBus-Builder can also generate VHDL instantiation templates. Note that the ReCoBus macro follows the same interface definition to the static part of the system as provided by the RTL simulation model.

Most common, the ReCoBus will map slaves directly into the system address space and in the case of masters, the ReCoBus will allow these modules to directly access the address space. In the case of multiple masters, we have to include an arbiter in the static part of the system that directs the access to the bus. The ReCoBus-Builder does not generate an arbiter together with the ReCoBus, but it can provide all dedicated signals between multiple masters and an arbiter that may implement any required policy to grant individual masters access to the bus.

### 3.4. I/O Bar Generation

Beside the ReCoBus hard macro we can specify the dimension and positions of the I/O bars. Multiple bars can be included into the system at different horizontal aligned positions that are located parallel to the ReCoBus. We can define if and where an I/O bar has a module connection point. Connection points are only required for modules that demand access to a particular bar. If no I/O bar connection is required, we will not waste any logic resources.

Each bar is exported as an individual hard macro, as described in the last paragraph for the ReCoBus. In each resource slot, the ReCoBus-Builder tool ensures that we always use the same equivalent wire to route the signal further to the succeeding slots, regardless if an I/O bar contains a connection or not. Consequently, modules can be freely placed within the reconfigurable resource area.

### 3.5. Bitstream Generation and Linking

Beside the ReCoBus and I/O bar hardmacros , we can generate VHDL files that instantiate modules following the ReCoBus communication interface protocol specification. Together with some additional constraints related to the system clock and area constraints, these VHDL files can be directly used by the Xilinx synthesis tools to generate complete bitstreams. The Xilinx place and route tool has some problems to fulfill area constraints. As a workaround, the ReCoBus-Builder can generate so called *blocker macros* that can be used to prohibit the usage of logic and routing resources at specific CLB positions.

As we can constrain the timing behavior of the ReCoBus and the I/O bars, it is possible to generate the different partial modules completely independent from the static part of the system. When we place and route the partial modules, we locate them at the most critical position that is furthest away from the static part of the system. Thus, we can directly crop the partial modules out of the complete bitstream in order to link the initial system. For this purpose, we implemented a *bitstream linker* that can compose together partial modules in a two-dimensional fashion (refer also to [10, 11]). In addition, we perform some design rule checks and we can set parameters (e.g., the base address) for the partial modules that are part of the initial system.

As the communication infrastructure requires only low resource overhead by providing high flexibility and low latency, the RoCoBus-Builder tool flow is also suitable to implement completely static systems. As the interface of the partial modules is completely encapsulated, a re-synthesis step of a particular module will not influence the timing of other parts of the system. In addition, our tool flow allows to parallelize the synthesis as well as the place and route process. Furthermore, the partitioned design flow reduces massively the memory consumption. The bitstream linking itself is performed within a few seconds.

## 4. EXAMPLE SYSTEM

In order to demonstrate the capabilities of the ReCoBus and the I/O-bar communication infrastructure, we implemented a system with the tool flow presented in the last section. An FPGA-editor view of the system is revealed in Figure 7. Beside a softcore CPU with a memory controller and other peripherals, the system provides a resource area containing 60 resource slots, each being 1 CLB column wide. In each resource slot, a module with an 8-bit wide interface could be connected to the ReCoBus. By the use of 4 consecutive resource slots it is possible to integrate modules with a full sized 32 bit slave interface into the system. The ReCoBus follows the Wishbone protocol and is directly connected to the softcore CPU in the static part of the system. Note that this system is the first providing a 1 CLB placement grid and also the number of individual resource slots is the highest one reported for an FPGA based system. The best competing solution provides 16 resource slots that are 4 CLB columns wide on a Virtex-II 4000 device [4].

Some modules can connect to a video I/O bar and the

video output of the system is depicted in Figure 8. A video module must be at least two resource slots wide for connecting to the I/O bar, because this interface is spread over two resource slots. However, this restricts not the module placement. The complete system runs at a clock frequency of 50 MHz. As the communication infrastructure is pipelined with one register, it would allow the double clock speed if implemented on a Xilinx Virtex-II 6000-6 device.

We implemented a set of different test modules including some video overlay modules (to test the video bar streaming capabilities), some crypto modules, and an FIR-filter. The latter one was built once by the use of look-up tables as well as once by additionally utilizing the dedicated multipliers available on all Virtex-II devices (see Figure 7). The implementation alternatives lead to more placement freedom at runtime. All test modules are routable within their smallest resource slot width according to the logic requirements of the particular modules. Only the RC5 crypto core required to widen the module width by an extra resource slot in order to get all signals routed within the module bounding box.

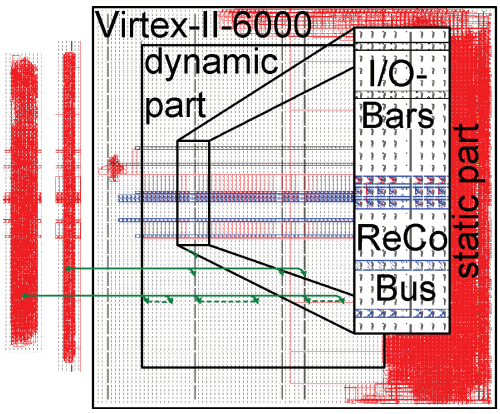


**Fig. 7**. Reconfigurable system with a static part containing a softcore CPU and a dynamic part with 60 resource slots. The two modules on the left hand side implement both the same FIR filter. The smaller one utilizes the dedicated multipliers provided in the Xilinx Virtex-II FPGA while the other one is built completely by look-up table resources.

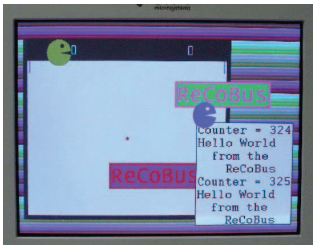


**Fig. 8**. Video output of the reconfigurable system. Each of the six objects (e.g., a text box) on the screen is created by an individual module. Each module receives the video stream from the I/O bar that is then overlaid with the particular object and sent further over the I/O bar to the next module.

## 5. CONCLUSION

In this paper, we presented the ReCoBus architecture and the novel I/O bar communication technique for integrating partially reconfigurable modules into a system at runtime. In addition, we introduced the tool ReCoBus-Builder that automatically generates the communication infrastructure for such systems. In addition the tool supports the design of reconfigurable systems down to the final bitstream generation. A test system demonstrated that our tool flow is capable to build systems with extreme high placement flexibility in combination with a high resource slot count. The ReCoBus-Builder tool flow can further be used to speed up the design of completely static systems by linking module bitstreams instead of merging systems on the netlist level.

## 6. REFERENCES


[1] H. Walder and M. Platzner, "A Runtime Environment for Reconfigurable Operating Systems," in *Proc. of the 14th Int. Conf. on Field Programmable Logic and Application (FPL'04)*, 2004, pp. 831–835.

[2] P. Sedcole, B. Blodget, T. Becker, J. Anderson, and P. Lysaght, "Modular dynamic reconfiguration in Virtex FPGAs," *IEE*, vol. 153, no. 3, pp. 157–164, 2006.

[3] L. Benini and G. D. Micheli, "Networks on Chips: A New SoC Paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, Jan. 2002.

[4] J. Hagemeyer, B. Kettelhoit, M. Koester, and M. Porrmann, "Design of Homogeneous Communication Infrastructures for Partially Reconfigurable FPGAs," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'07)*, Las Vegas, USA, 2007.

[5] D. Koch, C. Haubelt, and J. Teich, "Efficient Reconfigurable On-Chip Buses for FPGAs," in *Proc. of the 16th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2008)*, Palo Alto, CA, USA, Apr. 2008.

[6] M. Hübner, C. Schuck, M. Kühnle, and J. Becker, "New 2-Dimensional Partial Dynamic Reconfiguration Techniques for Real-time Adaptive Microelectronic Circuits," in *Proceedings of the IEEE Symp. on Emerging VLSI Technologies and Architectures*, Washington, DC, USA, 2006, p. 97.

[7] P. Lysaght, B. Blodget, J. Mason, J. Young, and B. Bridgford, "Invited Paper: Enhanced Architecture, Design Methodologies and CAD Tools for Dynamic Reconfiguration of Xilinx FPGAs," in *Proc. of the 16th Int. Conf. on Field Programmable Logic and Application (FPL'06)*, Aug 2006, pp. 1–6.

[8] Xilinx, 2008, `http://www.xilinx.com/ise/optional_prod/planahead.htm`.

[9] C. Claus et al., "An XDL-based busmacro generator for customizable communication interfaces for dynamically and partially reconfigurable systems," in *Works. on Reconfigurable Computing Education*, Porto Alegre, Brazil, May 2007.

[10] B. Blodget, P. James-Roxby, E. Keller, S. McMilla, and P. Sundararajan, "A Self-reconfiguring Platform," in *Proceedings of International Conference on Field-Programmable Logic and Applications (FPL)*, Lisbon, Portugal, 2003, pp. 565–574.

[11] E. L. Horta, J. W. Lockwood, and S. T. Kofuji, "Using parbit to implement partial run-time reconfigurable systems," in *FPL '02: Proceedings of the 12th International Conference on Field-Programmable Logic and Applications*. London, UK, 2002, pp. 182–191.