A Bus-based SoC Architecture for Flexible Module Placement on Reconfigurable FPGAs

Andreas Oetken, Stefan Wildermann, Jürgen Teich
Department of Computer Science
University of Erlangen-Nuremberg, Germany
Email: {andreas.oetken, stefan.wildermann, juergen.teich}@cs.fau.de

Dirk Koch
Department of Informatics
University of Oslo, Norway
Email: koch@ifi.uio.no

Abstract—This paper proposes an FPGA-based *System-on-Chip* (SoC) architecture with support for dynamic runtime reconfiguration. The SoC is divided into two parts, the static embedded CPU sub-system and the dynamically reconfigurable part. An additional bus system connects the embedded CPU sub-system with modules within the dynamic area, offering a flexible way to communicate among all SoC components. This makes it possible to implement a reconfigurable design with support for free module placement. An enhanced memory access method is included for high-speed access to an external memory. The dynamic part includes a streaming technology which implements a direct connection between reconfigurable modules. The paper describes the architecture and shows the advantages in a *smart camera* case study.

I. Introduction

The costs for application-specific integrated circuits (ASICs) have steadily risen in recent years and development costs can nowadays run into multi millions of dollars. Moreover, ASICs cannot be modified once integrated into a system. To overcome these drawbacks, field-programmable gate arrays (FPGAs) provide a frequently applied alternative. FPGA-based embedded systems are of increasing importance especially in the signal and image processing domain. They offer a solution to the implementation of flexible and high-performance image processing modules [1]. For instance, intelligent embedded systems for image processing, such as smart cameras, rely on FPGA-based architectures [2], [3], [4]. One disadvantage of FPGA-based systems is that the area requirements as well as the power consumption are higher than those of ASIC architectures [5]. But FPGAs support partial reconfiguration which makes it possible to replace hardware modules at run-time and thus enables the use of smaller FPGAs.

Partially reconfiguration of hardware offers further advantages: First, *offline* design is more comfortable since only parts of the system have to be re-synthesized after alterations, and thus down-time during the design process and, as a consequence, time-to-market can be reduced. Second, the SoC can be programmed *online* during its operation. Parts of the SoC can be altered without having to shutdown or even halt the system. This enhances system customizability and maintenance. Finally, reconfigurable SoCs allow the implementation of adaptive embedded systems which incorporate so-called *self-x* properties with the goals of self-optimization to meet new requirements, self-organization of the system setup, and self-adaptation to unpredictable changes in the environment, as, e.g., shown in [6] for a self-organizing, FPGA-based smart camera.

This paper addresses one of the main challenges in this context. It presents a design to provide a system-wide communication infrastructure. This includes communication between software and hardware modules via on-chip busses, as well as between hardware modules and memory by direct memory access (DMA). The presented architecture is tailored to support run-time hardware reconfiguration. We demonstrate the usability of the proposed architecture by presenting a smart camera application as a case study.

The paper is organized as follows. Section II presents some related work. The proposed architecture is described in III. Section IV illustrates the design flow. The smart camera case study is described in Section V. The implementation and experimental results are provided in Section VI before the conclusion of this paper in Section VII.

II. Related Work

A dynamic architecture based on reconfigurable hardware for image processing applications is presented in the Auto-Vision project [7]. Here, a hardware/software co-design for driver assistance is presented [8] where image filters can be dynamically exchanged. A comparison between two platforms for implementing this algorithm is given in [9]. However, both platforms only allow to load predefined image filter modules as they use the partial reconfiguration flow from Xilinx presented in [10]. In our paper, an architecture is presented that allows a more flexible way to place hardware modules with much finer granularity. In the *Sonic-on-a-Chip* project [11], a reconfigurable system with advanced two-dimensional integration capabilities of reconfigurable modules has been developed for multimedia applications. The proposed communication architecture suffers high throughput and lacks DMA capabilities which are demanded by many video applications, including old value convolution or motion detection.

III. Architecture

This research proposes a general architecture for flexible, reconfigurable, bus-based SoCs. An example is presented in Fig. 1. The system consists of the embedded CPU sub-system and the reconfigurable part. The whole system is located within a single FPGA. Furthermore, the SoC contains an external memory for high-throughput data transfers and the communication between the system components. In the following, these components and the communication interfaces between them are presented.
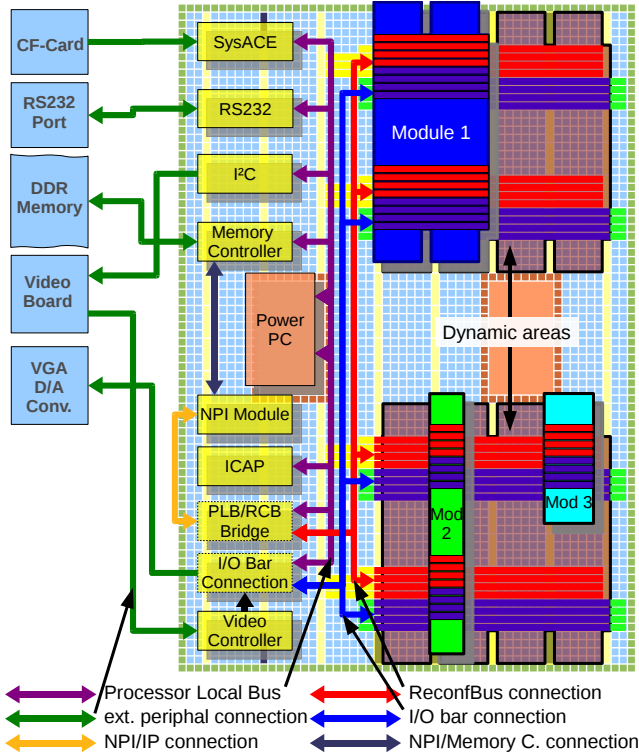
IEEE
computer
society

Fig. 1. System overview of one possible partitioning of the heterogeneous FPGA-based SoC platform consisting of CPU sub-system and reconfigurable area. Reconfigurable modules can vary in size and be freely placed, allowing a very good exploitation of the FPGA space. The ReCoBus (red) and the I/O bar (blue) are routed through the dynamic part, allowing a system-wide communication between hardware and software modules.
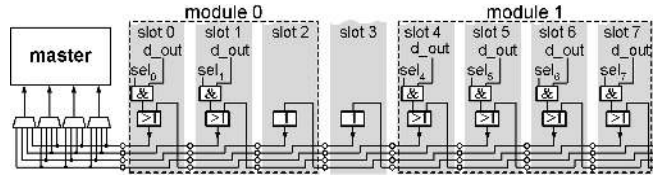
Fig. 2. ReCoBus interleaving scheme. By providing multiple independent interleaved and regular structured multiplexer chains, modules can be connected to different sized interfaces via a variable amount of consecutive bus slots. This scheme provides low latency for very fine-grained tile grids.

The I/O bar is used for streaming the data and establishing point-to-point connections between hardware modules. Note that the modules possess different sizes and that modules can be placed in a two-dimensional manner. This materially enhances FPGA utilization as modules can be fit into smaller bounding boxes that must only provide the required logic and memory resources.

1) Reconfigurable Bus: The ReCoBus [15] is an on-chip bus that is suitable for dynamically integrating hardware modules into an FPGA by partial reconfiguration. It consists of an FPGA hard macro with a regular structure, allowing partial reconfiguration at run time, even during bus transactions. The bus permits connections between master and slave hardware modules, including the communication to and from components of the embedded CPU sub-system.

The basic idea of a ReCoBus is illustrated in Fig. 2. The bus can provide all permutations of dedicated and shared as well as read and write signals from and to partially reconfigurable modules. This includes physical implementations of signals for module selection, bus arbitration, read/write control, reset generation and all kinds of data and address transfers. As depicted in the figure, the size of the bus interface may vary by using different amounts of consecutive bus slots for connecting a partial module.

The structure of the ReCoBus provides resource slots for dynamic module placement. These slots have a granularity of one Configurable Logic Block (CLB) width and, in the target implementation, 16 CLBs height. Since the signals are interleaved (cf. Fig. 2), the module width determines the maximal bandwidth for data transfers. A module with a width of one CLB column is able to read and write 8 Bit of data. Modules with a width of at least six CLB columns are then able to transfer up to 48 Bit data since the present ReCoBus can implement up to six interleaved signals. To allow burst-transactions from a reconfigurable master module to the embedded CPU sub-system, these signals can be used to transfer 32 Bit of data and up to 16 control-signals which include, for instance, information about the transfer length and synchronization. In the proposed design, four ReCoBus macros have been instantiated for implementing a two-dimensional ReCoBus architecture (cf. Fig. 1).

2) I/O bar: I/O bars provide point-to-point communication between modules which are located in the dynamic area. Details are described in [16]. The basic principle is that, in each resource slot, a module is able to read the incoming data, modify it if necessary, and pass it further to the next module.

For instance, the I/O bar can be used to stream video data between the various reconfigurable processing modules, as illustrated in Fig. 3. Here, modules can modify this stream

A. Embedded CPU Sub-system

The main purpose of the software part on the embedded CPU is to control and manage the overall system. It contains high-performance peripherals, interfaces, and other IP cores. These are, for example, a memory controller to provide access to an external RAM, a serial port interface for user commands, and a module for accessing the integrated reconfiguration interface of the FPGA. All components of the embedded CPU sub-system are connected by the main on-chip system bus, the *processor local bus* (PLB). This bus is designed for high performance, allows burst oriented high throughput modes, and is multi-master capable [12].

B. Reconfigurable Area

The FPGA area is divided into a static and a dynamic part. This is illustrated in Fig. 1. The two dark-red areas on the right top and bottom compose the dynamic part of the system. Reconfiguration is only possible in the dynamic part which contains a reconfigurable on-chip bus (*ReCoBus*[1]) and *I/O bar* communication primitives to provide a communication infrastructure for dynamically loaded hardware modules. Three partially reconfigurable modules are exemplary integrated into the dynamic area (green, dark blue and turquoise). Both primitives are provided by the framework ReCoBus-Builder [13], [14].
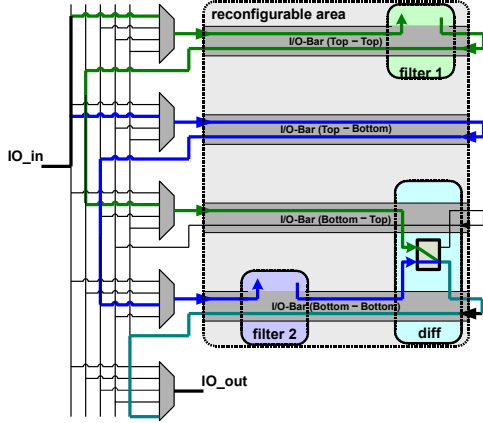
[1]name changed for blind review

Fig. 3. I/O bar consisting of multiplexer in the static part and communication lines in the dynamic part. Three filters are illustrated. Filters can read and modify the signals routed over the I/O bar.
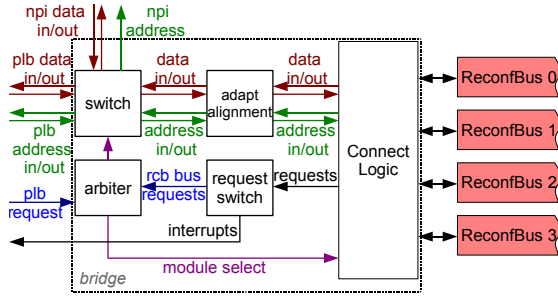

Fig. 4. Simplified overview of the PLB/RCB Bridge. In this example, the bridge connects four ReCoBus macros with the static part.
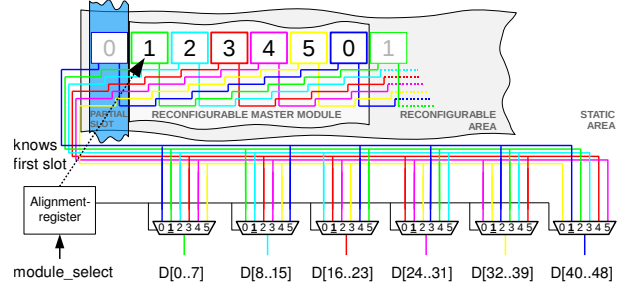

Fig. 5. The *aligment logic* multiplexes the data from a ReCoBus module depending on the module-placement. The first slot of the module (green) carries the first 8 Bit of data. The alignment register controls the multiplexer depending on the active module, specified by the module-select signals.

or generate additional output signals.

C. PLB/RCB Bridge

To allow communication between the embedded CPU sub-system and the reconfigurable part, a PLB/RCB Bridge translates the ReCoBus protocol to the PLB protocol and vice versa. Additionally, the bridge contains the arbiter for the ReCoBus, as well as the logic needed to connect to the ReCoBus macros. Handshaking coordinates the data flow between the bridge and the buses. A structural overview of the bridge is shown in Fig. 4.

The *connect logic* merges and distributes the signals from and to the ReCoBus macros and implements a common interface wrapper to the ReCoBus sub-system.

Since the request lines of a ReCoBus macro can be used as interrupt signals as well as bus requests signals, they have to be separated accordingly inside the bridge. For this purpose, the *request switch* block includes mask registers which hold information about the use of the request lines. This allows to distinguish between interrupts and bus requests.

The *arbiter* coordinates the bus access between PLB and the reconfigurable modules. It uses a round robin policy to grant fair access to the bus for master modules connected to the ReCoBus. Incoming requests from the PLB are priorized in the proposed design. This is due to the fact that the CPU sub-system is mainly performing short control transactions with the

ReCoBus modules, whereas reconfigurable modules mainly run large memory intensive burst transactions. The arbiter is able to stall data transfers initiated by hardware masters in order to allow CPU requests to be processed. This is necessary to avoid deadlocks which can emerge when a PLB master wants to access a reconfigurable module which concurrently tries to access a component of the CPU sub-system.

The *switch* shown in the upper half of Fig. 4 multiplexes the data and address signals, allowing communication between all parts of the system.

Since the ReCoBus modules can be freely placed within the reconfigurable area and the data and address signals are interleaved, these signals must be aligned correctly depending on the destination of a transaction. This is done by the *adapt alignment* block. The principle of this process is revealed in Fig. 5 for the data-out lines of a module. The first slot connected to the module always carries the first 8 Bit of data, independent of the module placement. Thus, a set of multiplexers is required to align the data into the correct order (see also Fig. 2) depending on the module placement position.

Finally, the bridge provides access to the *native port interface* (NPI) [17] of the memory using an additional hardware module, which allows high-speed data transfers between hardware masters and the memory controller without having to access the PLB bus. A comparison of the transfer speeds using PLB and NPI is given in Section VI-A.

D. Reconfigurable modules

The reconfigurable sub-system supports the integration of partial master and slave modules with variable address ranges and data widths. To allow independent master and slave access, different select addresses can be set within the ReCoBus macro with the help of *Reconfigurable Select Generators* (RSGs). A RSG is basically a look-up table that can be updated to decode a specific macro internal address vector. This process is transparent to the reconfigurable modules. Consequently, multiple instances of the same module can be integrated into the system that can be individually accessed by setting different address ranges inside the RSGs.

In order to physically implement request signals, the ReCoBus macro provides a regularily routed internal wire bundle. By selectively manipulating switch matrix entries directly inside the FPGA routing fabric, request signals from the modules can be connected to drive a particular wire of this bundle. Again, this is transparent to the entire module and
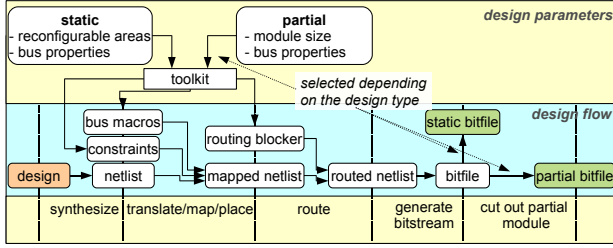
Fig. 6. Design flow.

multiple instantiations of the same module are possible by utilizing different wires of the bundle.

The ReCoBus provides a fully PLB-compliant bus interface for both master and slave operations. This makes it possible to reuse existing PLB IP cores as partially reconfigurable modules. The management of the ReCoBus resources and the corresponding bitstream manipulations are fully encapsulated in a driver API.

IV. DESIGN FLOW

To achieve partial reconfiguration as proposed in this paper, a customized design flow is necessary which is presented in Fig. 6. The flow is divided into four to five steps depending on the design type (*static* or *partial*).

The static design, which includes the embedded CPU subsystem, the bridge and the logic to connect to the ReCoBus, is a description for the entire FPGA. All routing and resource units except the ones inside the partially reconfigurable area are useable.

The partial design uses only the resources included in the area which is defined by the dimensions of a partial module inside the dynamic area.

The *toolkit* generates the macros for the ReCoBus and the I/O bar, as well as constraints to prohibit the use of resource units and a blocker for the routing resources. For the partial design, a seperate tool makes it possible to cut the used content of the module out of the bitstream.

V. SMART CAMERA CASE STUDY

An embedded design for tracking human motion is implemented as case study for analyzing the flexibility of the proposed methodology. The idea is to detect and track skin-colored image regions, which is done by applying particle filtering.

A. Particle Filtering

The basic framework is proposed in [18]. Let x_t be the state of a tracked object in a sequence of images and y_t the current measurement extracted from the camera image, then the goal is to keep track of the probability distribution $p(x_t|y_{t:1})$ of the object state given the sequence of measurements from time step 1 to t. In the particle filter framework, this distribution is represented by a set of N state samples with corresponding weights $S_t = \{\langle x_t^{(i)}, w_t^{(i)} \rangle\}$. The filtering mechanism basically consists of three steps:

In the *sampling* step, a set of hypothesis about possible object positions is generated by drawing samples from the probability distribution S_{t-1} according to their weights.
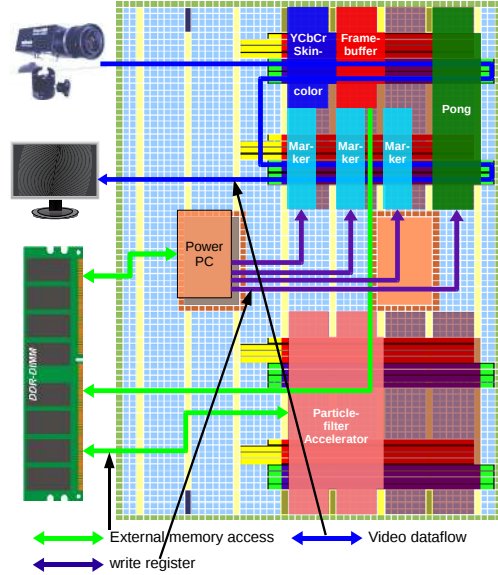

Fig. 7. Schematic overview of the implemented probabilistic smart camera case study with the hardware modules located in the reconfigurable area. Depicted is the data flow of the video stream via the IO bar, and the data flow between modules and memory.

In the *propagation* step, a position prediction is applied on this set by propagating each sample according to a motion model.

Finally, each propagated sample is evaluated based on measurements performed on the input image. The weight of each sample is set to the corresponding value of the measurement probability distribution $p(y_t|x_t)$.

B. Tracking Application

The application is based on skin color detection [19] which is performed on the input image. A so-called *region tracker* particle filter is applied for tracking the image position x_t of a skin color region. To track k regions, k of these filters have to run. The filter implements the above framework as follows: After the sampling step, each particle is propagated by using a deterministic drift followed by a Gaussian diffusion. Evaluation of a sample $s_t^{(i)} \in S_{k,t}$ is performed by counting the number of skin color pixels in a 9×9 region around the image position $x_t^{(i)}$.

To support tracking of multiple objects, a *scout* particle filter is applied to recognize new objects entering the image. First, image regions that are already tracked by a region tracker are masked in the pre-processed image. Then, particles are sampled. A fraction of the samples are not generated by drawing from $S_{scout,t-1}$, but by distributing them uniformly over the input image. Evaluation on the masked image is performed as before. Whenever the standard deviation of the samples $S_{scout,t}$ and the average sample weight exceed specific thresholds, a new region tracker is initialized at the mean position of the samples for tracking the new region.

C. Implementation

The system is implemented on a Xilinx Virtex-II XUP Pro board as illustrated in Fig. 7. The *skin color detection*

237

is implemented as a hardware slave module that reads the color values from the I/O bar and marks them as *skin* or *non-skin* by comparing them with a color template. We have implemented modules for RGB and YCbCr color spaces [19]. The classification is written as an additional signal onto the I/O bar together with the unmodified video stream.

The *framebuffer* hardware master module is implemented to store the current input image. This is done by double buffering the images in the on-chip memory via the ReCoBus using the NPI interface. We use 32 Bit for storing one pixel, with 24 Bit for the input RGB values and the remaining 8 Bit free for classification results.

The particle filtering framework is partitioned into a software and a hardware part. The set of particles is stored in the on-chip DDR memory. The software part performs the sampling and applies the motion model. The hardware part is used as a co-processor to perform the evaluation steps. All particles are loaded into the hardware module's local buffer and are then sequentially evaluated by loading the data of the image region around each particle and calculating the weight as described above. These weights are then stored together with the sample states in memory and are used for sampling in the next step.

The current implementation makes it possible to track up to 3 image regions. We have furthermore implemented a *marker* module to display the tracking result in the camera image. One marker module is used per region tracker. A simple *tennis game* is implemented on top of this application, which can be directly controlled by the hands of a person, using the results of the tracker (see Fig. 10).

VI. IMPLEMENTATION RESULTS

The proposed SoC design is implemented on a Xilinx Virtex-II XUP Pro board. The reconfigurable area is constricted by the two power PCs and thus divided into two parts. Each of the two reconfigurable parts is 24 CLB columns wide and 32 CLB rows high.

The video input signal from a video board is routed over the I/O bar and then streamed back to a VGA output. The ReCoBus is connected with the PLB/RCB bridge which offers a connection to the PLB and to the NPI, see Fig. 1.

The PowerPC is running with a clock of 300 MHz. The remaining system is clocked at 100 MHz. The reconfiguration is performed using a clock of 50 MHz. The video input format is PAL 50 Hz with a resolution of 720×576 pixels.

A. Data Transfer

The memory bandwidth may become a critical bottleneck in video applications. Looking at a video stream with a frame rate of 50 FPS and a resolution of 720×576 pixels, each represented by 3 Bytes, $62 \frac{MB}{s}$ are required to transfer all frames into the memory. Accessing the data doubles the amount of bandwidth needed. In Fig. 8, the data throughput for PLB transactions over the burst rate is shown. Obviously, the datarate is barely sufficient for the above example. In contrast, the implemented NPI module increases the bandwidth near the theoretical maximum of $400 \frac{MB}{s}$ given by a clock of 100 Mhz and a word length of 32 Bit.
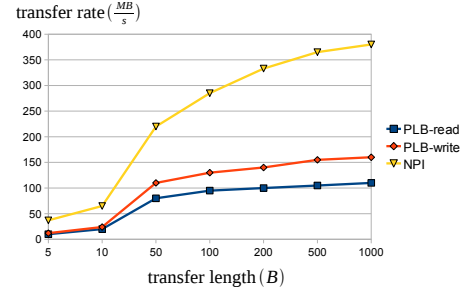

Fig. 8. Speed comparison between memory transactions via PLB, and by using the NPI interface directly.
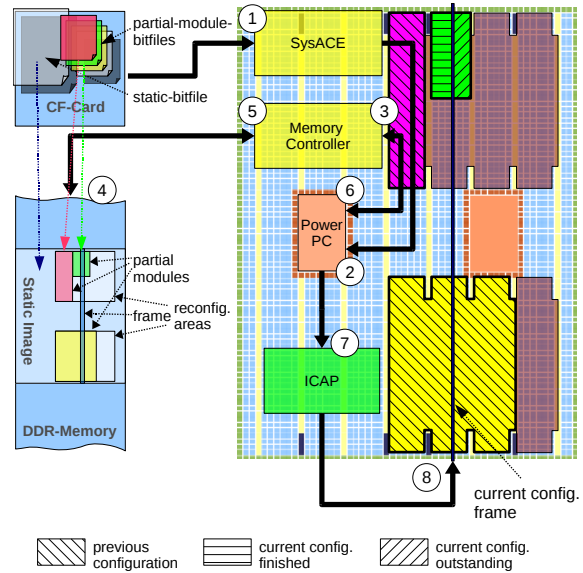

Fig. 9. Reconfiguration procedure.

B. Run-time Reconfiguration

A scenario for run-time reconfiguration is illustrated in Fig. 9. It is triggered by the PowerPC. The image of the static part is loaded from a CF card into the memory (see steps 1 to 4 in Fig. 9). This is necessary, since reconfiguration is done in *frames* which cover one complete vertical column of the FPGA as illustrated in the figure. This means that for partial reconfiguration also logic from the static part has to be re-written. When configuring a partial module, its image is loaded into the memory (via steps 1, 2, 3, 4). Then, it is combined with the corresponding parts of the static image and other partial modules which are already configured, to generate the frames for reconfiguration. These are then loaded via steps 5 to 8 illustrated in Fig. 9).

Some of the partial modules of our case study are shown in Table I. It lists the modules' widths and heights in numbers of CLB slots, the size of the modules' images in Bytes and whether interrupt and bus requests are required. The reconfiguration periods of these modules are displayed in Table II. It shows the durations of loading the modules from the CF card, writing them onto the FPGA, and setting the alignment and request parameters, separately.

module	slots (w/h)	irq/req	size(B)
marker	4/1	n/n	14.120
skin	7/1	n/n	24.683
accel.	7/2	y/y	49.323

TABLE I

MODULEDATA FROM THE SMART CAMERA CASE STUDY.

module	reconfiguration time (ms)			
	load	write	connection	all
marker	15,9	7,7	0,3	23,9
skin	27,9	13,2	0,3	41,4
accel.	56	13,3	4,5	73,8

TABLE II

RECONFIGURATION TIMES FOR MODULES FROM THE SMART CAMERA CASE STUDY.

part.	calculation time (ms)						comm. overhead
	E		S	P	overall		
	sw	hw			sw	hw	
100	0,9-1,4	0,4	0,4	0,2	1,5-2,0	1,0	30 %
200	2,1-2,9	0,8	1,0	0,3	3,4-4,2	2,1	31 %
500	4,8-6,2	2,1	3,5	1,0	9,3-10,7	6,6	34 %
1000	9,0-12,2	4,2	7,1	1,9	18,0-21,2	13,2	38 %

TABLE III

HARDWARE ACCELERATION OF PARTICLE FILTER WITH TIME REQUIREMENTS FOR PARTICLE EVALUATION (E), SAMPLING (S), AND PROPAGATION (P).

(a) The particle filter tracking three objects.

(b) The object tracker used to play a pong game.

Fig. 10. The particle filter in action. The framework tracks three image regions (a person's head and hands). The tracked hand positions are directly used to control the paddles of the video game.

As can be seen, most time is spent on reading the modules from the CF card. This period can be bypassed by prefetching the modules into the DDR memory. The time for loading a module onto the FPGA is proportional to the size of the module. Note that this is independent of the module's height, since reconfiguration is done by loading a complete frame. Connecting the module is performed by setting the alignment parameters. This requires the least amount of time. Here, most of the time is spent on connecting the interrupts and bus requests.
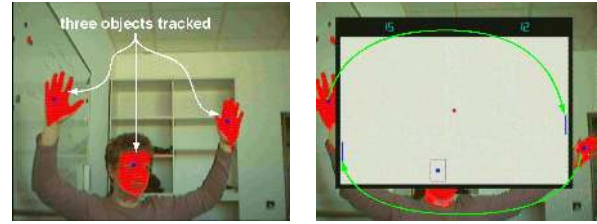
C. Case Study Results

Table III displays the speedup of the particle filtering supported by hardware. The calculation of the observation module is much faster. However, the memory bandwidth is the limiting factor. Due to the small size of the image regions, only small burst transmissions are possible. This leads to a poor performance because of the latency of the external memory. As can be seen, the communication between the PowerPC and the hardware accelerator imposes 30% to 38% overhead.

Still, the proposed system makes it possible to track human motion in real time with 50 FPS and a resolution of 720×576 pixels. By loading the video game hardware module, the proposed SoC enables a person to control the paddles of the game in real time, using the results of the tracked hand positions. All processing is performed by the system and displayed via the VGA output as illustrated in Fig. 10.

VII. CONCLUSION AND FUTURE WORK

This paper presents a flexible architecture for building systems-on-chips on FPGA-based architectures. The presented architecture offers a reconfigurable area which can be used to dynamically load hardware modules. One benefit of such an approach is that customizable and self-adaptive flexibility can easily be implemented. The flexibility of the proposed architecture is discussed by presenting an object tracking application for smart cameras. The results are encouraging and prove the flexibility of the architecture.

REFERENCES

[1] M. Tusch, "High-performance image processing on FPGAs," *Xcell Journal*, no. 57, pp. 42–44, April 2006.
[2] C. T. Johnston, K. T. Gribbon, and D. G. Bailey, "FPGA based remote object tracking for real-time control," in *International Conference on Sensing Technology*, 2005, pp. 66–72.
[3] J. Schlessman, C.-Y. Chen, W. Wolf, B. Ozer, K. Fujino, and K. Itoh, "Hardware/software co-design of an FPGA-based embedded tracking system," in *Proceedings of CVPRW*, 2006, p. 123.
[4] S. Fleck, F. Busch, and W. Straßer, "Adaptive probabilistic tracking embedded in smart cameras for distributed surveillance in a 3d model," *EURASIP J. Embedded Syst.*, vol. 2007, no. 1, pp. 24–24, 2007.
[5] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," in *Proc. of FPGA*, 2006, pp. 21–30.
[6] S. Wildermann, G. Walla, T. Ziermann, and J. Teich, "Self-organizing multi-cue fusion for FPGA-based embedded imaging," in *Proc. of FPL*, 2009, pp. 132–137.
[7] C. Claus, W. Stechele, and A. Herkersdorf, "Autovision - a run-time reconfigurable MPSoC architecture for future driver assistance systems," *it - Information Technology*, vol. 49, no. 3, pp. 181–, 2007.
[8] N. Alt, C. Claus, and W. Stechele, "Hardware/software architecture of an algorithm for vision-based real-time vehicle detection in dark environments," in *Proceedings of DATE*, 2008, pp. 176–181.
[9] C. Claus, W. Stechele, M. Kovatsch, J. Angermeier, and J. Teich, "A comparison of embedded reconfigurable video-processing architectures," in *Proc. of FPL*, Sept. 2008, pp. 587–590.
[10] Xilinx, *Early Access Partial Reconfiguration User Guide For ISE 8.1.01i (UG208)*, 03 2006.
[11] P. Sedcole, "Reconfigurable Platform-Based Design in FPGAs for Video Image Processing," Ph.D. dissertation, Department of Electrical and Electronic Engineering, Imperial College of Science, London, GB, Jan. 2006.
[12] IBM, *IBM CoreConnect bus cores*, 07 2006.
[13] D. Koch, C. Beckhoff, and J. Teich, "A Communication Architecture for Complex Runtime Reconfigurable Systems and its Implementation on Spartan-3 FPGAs," in *Proc. of FPGA 2009*. Monterey, California, USA: ACM, Feb. 2009, pp. 233–236.
[14] "Recobus-homepage," http://www.recobus.de/.
[15] D. Koch, C. Haubelt, and J. Teich, "Efficient reconfigurable on-chip buses for fpgas," in *Proc. of FCCM*, April 2008, pp. 287–290.
[16] D. Koch, C. Beckhoff, and J. Teich, "Recobus-builder a novel tool and technique to build statically and dynamically reconfigurable systems for fpgas," in *Proc. of FPL 2008*, Sept. 2008, pp. 119–124.
[17] Xilinx, *Multi-Port Memory Controller (MPMC) (v4.03.a)*, 07 2008.
[18] M. Isard and A. Blake, "Condensation—conditional density propagation forvisual tracking," *Int. J. Comput. Vision*, vol. 29, no. 1, pp. 5–28, 1998.
[19] V. Vezhnevets, V. Sazonov, and A. Andreeva, "A survey on pixel-based skin color detection techniques," in *Proc. Graphicon-2003*, 2003, pp. 85–92.