Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2010, Article ID 159367, 13 pages
doi:10.1155/2010/159367

Research Article

Mechanism of Resource Virtualization in RCS for Multitask Stream Applications

L. Kirischian,[1] V. Dumitriu,[1] P. W. Chun,[2] and G. Okouneva[3]

[1] *Embedded Reconfigurable Systems Laboratory (ERSL), Department of Electrical and Computer Engineering, Ryerson University, Toronto, ON, Canada M5B2K3*
[2] *MDA Space Missions, Brampton, ON, Canada L6S4J3*
[3] *Department of Aerospace Engineering, Ryerson University, Toronto, ON, Canada M5B2K3*

Correspondence should be addressed to V. Dumitriu, vdumitri@ee.ryerson.ca

Received 8 March 2010; Accepted 14 September 2010

Academic Editor: Lionel Torres

Virtualization of logic, routing, and communication resources in recent FPGA devices can provide a dramatic improvement in cost-efficiency for reconfigurable computing systems (RCSs). The presented work is "proof-of-concept" research for the virtualization of the above resources in partially reconfigurable FPGA devices with a tile-based architecture. The following aspects have been investigated, prototyped, tested, and analyzed: (i) platform architecture for hardware support of the dynamic allocation of Application Specific Virtual Processors (ASVPs), (ii) mechanisms for run-time on-chip ASVP assembling using virtual hardware Components (VHCs) as building blocks, and (iii) mechanisms for dynamic on-chip relocation of VHCs to predetermined slots in the target FPGA. All the above mechanisms and procedures have been implemented and tested on a prototype platform—MARS (multitask adaptive reconfigurable system) using a Xilinx Virtex-4 FPGA. The on-chip communication infrastructure has been developed and investigated in detail, and its timing and hardware overhead were analyzed. It was determined that component relocation can be done without affecting the ASVP pipeline cycle time and throughput. The hardware overhead was estimated as relatively small compared to the gain of other performance parameters. Finally, industrial applications associated with next generation space-borne platforms are discussed, where the proposed approach can be beneficial.

1. Introduction and Related Works

During the last decade progress in CMOS technology has allowed an increase in the amount of FPGA resources by almost an order of magnitude. This technological progress has made possible the implementation of relatively complex systems-on-chip (SoC) in a single FPGA but on the other hand, caused several drawbacks. Firstly, increased complexity of the SoCs overcomplicates the routing structure in the FPGA. This, in turn, reduces system performance and dramatically increases design compilation time which often requires many hours or even days. From the application point of view, the increased amount of resources allows effective utilization of FPGA devices for parallel execution of different tasks associated with multimodal and multitask algorithms and streamed data. However, from the reliability point of view, reduction of CMOS gate dimensions increases the probability of hardware faults. These faults can be caused by radiation, hidden manufacturing defects, and many other factors.

The problems above are not new in the field, and one possible approach to solving some of them can be the virtualization of homogenous FPGA resources. First, the computing model which allowed virtualization of resources in a single FPGA has been proposed by Caspi et al. in [1] and is called SCORE. This model assumed segmentation of computation process into fixed-size "pages" and time-multiplexing the virtual pages on available physical hardware. The model, however, did not consider multimodal workloads and was not actually implemented in an FPGA platform. Instead, the hypothetical single-chip SCORE organization was proposed with associated hardware requirements. In [2] Wallner has proposed the model for multithread computation by virtualization of resources. There are also several works reporting

implementation of the concept of resource virtualization in RCS (e.g., [3, 4]). Most of the above implementations are oriented towards coarse-grained or hybrid RCS architectures. However, to the best of our knowledge, none of the above models provides aggregative solution for all the above problems. In contrast with the above, our approach of virtualization of computing resources is oriented towards fine-grained homogenous RCS architectures and multitask, multimodal workloads. There are several advantages which an architecture based on fine-grain homogenous resources could provide. First, it allows rapid and precise run time adaptation for workload variations. At the same time, fine-grain RCS architectures makes possible the restoration of on-chip computing circuits from almost all possible hardware faults. To make use of these advantages two concepts have been proposed. The first is the concept of application specific virtual processor (ASVP). An ASVP is a data stream processing pipelined data-path optimized for a certain algorithm (task segment) and predetermined data-stream structure [5]. Virtualization of resources in ASVP assumes that it can be on-chip (in the FPGA) assembled from virtual hardware components (VHCs). The VHC is the second concept proposed for effective utilization of homogenous logic and routing resources in FPGA-based RCS. VHCs in our consideration are macro-function specific modules [5] represented (and stored) in the form of a configuration bit streams. A VHC consists of two parts: (i) a functional part (similar to an IP-core) which performs function-specific data processing and (ii) an interface part which provides uniform input/output ports as well as synchronization circuits. The uniform interface allows plug-and-play allocation of the VHC in addressed slots of a partially reconfigurable FPGA. This approach makes possible run time on-chip ASVP assembling according to current task and task mode. ASVP assembling, therefore, requires an application-specific static infrastructure to be loaded into the FPGA at start-up time. This static part incorporates the set of slots-and-sockets for VHCs and associated communication routing for all modes of the task. Then, a set of VHCs can be loaded into predetermined slots and combined to create a specific ASVP. Each variant of the ASVP is associated with one of multiple modes of task operation. When the mode of operation is completed or interrupted, the associated ASVP should be modified in run time by replacing one or more VHCs in its architecture. This kind of ASVP reconfiguration dictates the necessity of run time component relocation in the target FPGA. In other words, it may be unknown which slot-and-socket will be available in the ASVP. Thus, the uniform VHC interface should allow allocation and then relocation of the virtual component to an available slot. In recent years a number of research efforts have been directed towards the problem of component relocation in dynamically reconfigurable embedded systems. A number of bit-stream manipulation methods have been proposed [6–8] which alter the bit-stream structure as it is downloaded to the FPGA. However, these works concentrate solely on the bit-stream manipulation process, avoiding consideration of all steps of the real-time relocation process. Becker et al. provide a method for relocation between portions of a device which are not identical in terms of resources [9]. Here, the authors present both the bit-stream manipulation method as well as some of the requirements which must be met for this method to work. However, the system infrastructure requirements are not discussed.

Hübner et al. propose a complete framework for virtual components and component relocation based on the Xilinx Virtex-II family of columnar devices [10]. The authors make use of the network-on-chip interconnect concept to accommodate communication between components. As well, the JBits software package is used to relocate components between slots. Finally, Bobda et al. present the Erlangen Slot Machine (ESM), a reconfigurable computing system based on the Virtex-II family of columnar devices [11]. The authors describe the architecture of the complete platform, including the interconnect infrastructure provided for module communication. Because of the columnar organization of the FPGA device used, the communication infrastructure is either incorporated inside the architecture of each virtual component, or deployed externally, inside a secondary infrastructure FPGA. This, in turn, reduces the effectiveness of the approach compared with a complete on-chip solution because of complicated system infrastructure (e.g., motherboard plus several baby boards, etc.).

In contrast with the above approaches, the component relocation process presented in this paper targets tile-based FPGA devices (i.e., the Xilinx Virtex-4 or Virtex-5 families of FPGA devices), which offer more flexibility in the implementation of ASVPs. For example, the communication infrastructure can now be separate from the components themselves. Thus, the research addresses the architectural requirements needed for various levels of component virtualization, including on-chip assembling and relocation of components. In the case of relocation, the proposed framework provides an example of how complete streaming systems can be constructed using component relocation. This includes both the on-chip infrastructure as well as the behavior of components themselves.

The above mechanism of dynamic VHC relocation allows the implementation of concepts of spatial and temporal locality similar to virtual memory organization in advanced sequential processor architectures. In other words, the sooner a VHC may be requested for operation, the closer to the configuration memory of the target FPGA it should be located. Thus, at the system level of architecture a memory hierarchy for configuration bit streams should be organized. This memory hierarchy needs to contain a cache for VHCs as well as Main Storage for VHC bit streams for all tasks and their modes of operation. This organization of the memory hierarchy allows one to take advantage of the following facts: (i) only one of the modes of each task is active at a time, (ii) only a few of all tasks are active at a time and (iii) the same VHCs can be used for different tasks in different FPGA slots.

Therefore, the proposed approach allows a dramatic increase in cost-effectiveness of RCS through the reduction of computation resources, because nonactive data-processing circuits are stored in the form of configuration bit streams but not in the form of actual circuits in the FPGA.

Furthermore, power consumption in this case will also be reduced accordingly. At the same time, restoration of transient hardware faults can be done much faster because scrubbing procedures should be applied only to the slot affected by single-event-upsets (SEUs) but not the entire FPGA. Permanent hardware faults, on the other hand, can be mitigated by rapid VHC relocation to anther available slot(s) [12]. Thus, virtualization of computing resources based on run time on-chip assembling of ASVP and VHC dynamic relocation may become a promising solution for all problems listed in the beginning of this section. However, there are several issues and uncertainties regarding the implementation of the above ideas and concepts in real RCS. Therefore, the focus of this research was on the following aspects: (i) the development of an RCS architecture with a memory hierarchy for virtual components which will be able to support dynamic allocation and on-chip assembly of application specific virtual processors (ASVP), (ii) investigation of the mechanisms for run time on-chip assembly of ASVP from virtual hardware components (VHCs) and (iii) mechanisms for run time VHC relocation in predetermined regions of the FPGA device and automated reassembly of the associated ASVP in run time.

All the above aspects of the research presented here have been targeted specifically towards streaming applications with high data-rate requirements. The proposed framework at the system level and on-chip was developed with autonomous mobile applications in mind, where minimization of power consumption and high reliability are the major concerns. It was clear that all components of the above framework should be prototyped and tested on hardware platform(s). Therefore, a prototype platform, the multitask adaptive reconfigurable system (MARS) was designed, manufactured and tested. In addition, high-frame rate (up to 200 fps) stereo vision sensors have been developed and incorporated with reconfigurable platforms as well as VHCs for 3D multistream video processing.

The rest of the paper presents the results of the above investigation and implementation. Section 2 provides a specification of multitask and multimode workloads, as well as the system architecture of the multitask adaptive reconfigurable system (MARS) platform. Section 3 presents the on-chip architecture organization and analyzes the effectiveness of this approach. Section 4 presents the mechanism of component relocation. In Section 5, a description of the industrial applications of the proposed mechanism is given. Finally, Section 6 concludes the paper.

2. RCS Architecture and Workload

The architecture of an RSC which is able to support virtualization of on-chip computing resources consists of two parts: (i) system level architecture and (ii) on-chip architecture. Both parts directly depend on the workload specification. Thus, a workload specification is discussed first.

2.1. A Multitask and Multimode Workload. For the purposes of this discussion it was assumed that every application is fundamentally comprised of operations that are arithmetic or logic. Associated operations are grouped together to create a *task* (**Ts**). Each task is considered an independent information object in regard to any data or control dependences with other tasks. Nevertheless, tasks may be linked by input data-streams or share the same resources for the output data. Once all interactions and communication between tasks are arranged, higher levels of computation (i.e., application −**AP**) can be formed. In other words, the application can be considered as a set of tasks that are connected (or disjointed) for the system at times. The connected set of tasks by output(s) of the system is referred to as a *thread* (**Th**). Different functionalities of a task are referred to as modes of the task (**Md**). This means that the task algorithm and/or data structure may have some variations. Each variant of the algorithm and/or data structure can be considered as a mode of the task.

The set of applications over the life time of the system is called the workload (**W**). As per definition made above, any change of the mode **Md***j* of a task **Ts***i* means that the set of VHCs—{VHC*ij* } has to be replaced by a new set—{VHC*ik*} associated with mode **Md***k* of the same **Ts***i*. In most real cases not all VHCs involved in a mode **Md***j* of a task ***Tsi*** should be replaced when the mode changes to **Md***k*. Usually, there are very few new VHCs which have to replace existing VHCs. This is true because in most real cases there is quite a limited difference in functionality between different modes of operation in a task. Furthermore, the request for mode switching is usually associated with a certain response time for switching to mode **Md***k*-**T***SW(ik)*. Therefore, reconfiguration of a number of VHC slots—**N***ij* associated with the new mode **Md***k* of a task **Ts***i* should be done within this period of time. Thus, the granularity of VHC slots directly depends on mode switching time, number and size of VHCs to be replaced and the bandwidth of the FPGA configuration port—**BW**conf.

In this case VHC slot size in a certain FPGA can be estimated by the size (volume) of the configuration bit-stream required for a slot—**BS**slot—and equal to

$$\mathbf{BS}\text{slot} \le \mathbf{T}\text{mode}(ij) * \frac{\mathbf{BW}\text{conf}}{\mathbf{N}ij}. \tag{1}$$

For example, if the mode switching time is *1ms* and the bandwidth of the configuration port is 800 Mb/*s* (e.g., for SelectMap-8 configuration port in Xilinx Virtex FPGA), the size of the VHC's configuration bit-stream will be equal to 780 Kbit for the VHC slot if only one VHC has to be replaced. It is possible to calculate the number of logic slices available for the VHC slot. In the case of the smallest Xilinx Virtex-4 FPGA (XC4VLX15), the slot size will contain 1030 slices and 6 VHC slots will be available in this FPGA. In contrast to this, the largest FPGA in the Xilinx Virtex-4 family may contain 64-VHC slots.

Keeping in mind the size of the configuration bit-stream for the VHC slot, volumes of VHC cache and main VHC memory can be estimated. In all cases the number of different VHCs to be stored in the Main VHC memory and VHC cache is expected to be quite large. Based on the above example, it is possible to estimate the number of VHCs to

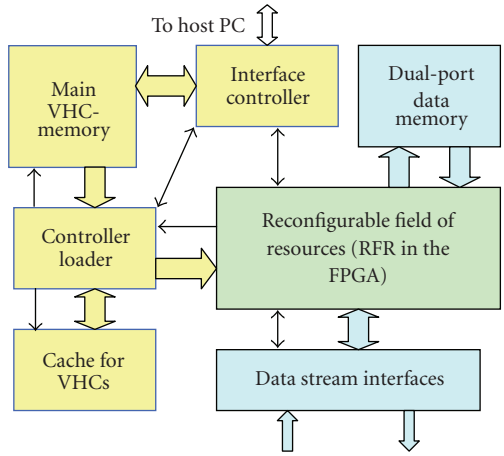


Figure 1: Block diagram of RCS architecture.

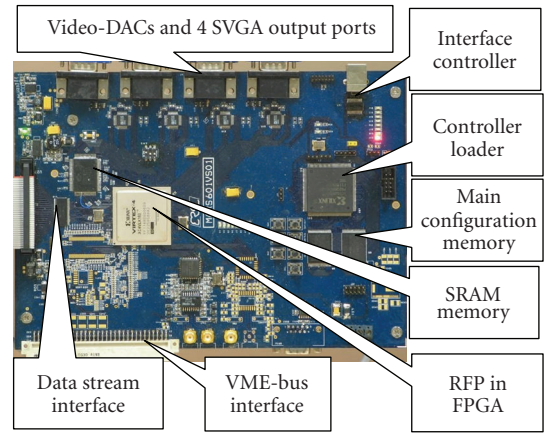


Figure 2: Component layout on MARS platform.

be stored in relatively cheap Flash memory. For example a 10 GB Flash memory module can store up to 107,546 VHCs. It is necessary to mention that this amount of VHCs is associated only with the current application. The bit streams associated with all other applications of the workload—**W** can be stored in remote locations (e.g., host PC). Therefore, the configuration memory hierarchy should consist of the following levels:

(1) *Operational level*: configuration memory of the target FPGA device(s) which contains bit streams of all active VHCs processing the data in current mode of operation,

(2) *Cache level:* VHC cache deployed as close as possible to the target FPGA device(s) which stores bit streams of VHCs that can be requested by other modes of the active tasks,

(3) *Main level:* main VHC memory deployed on the same board as target FPGA device(s) which stores bit streams of all VHCs required for the set of tasks in current application,

(4) *External level:* External (secondary) storage of the VHC bit streams required for all applications in the workload. This storage can be deployed in the hard disk drive of a host PC or any other carrier.

2.2. System Level Architecture of RCS. According to the above memory hierarchy for VHCs, the general architecture of an RCS able to support virtualization of FPGA resources is shown in Figure 1.

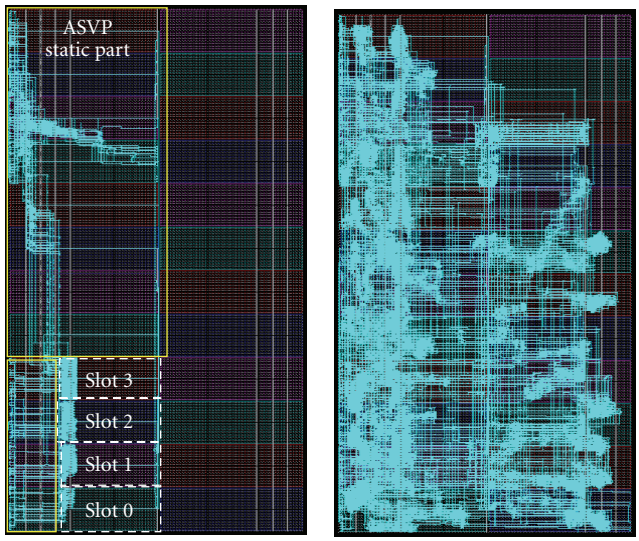
The major elements of this architecture are the following. (i) Reconfigurable field of homogenous resources—RFR (in the target FPGA), (ii) configuration memory hierarchy: consisting of the Main VHC memory, Cache for VHCs and configuration memory of the FPGA, (iii) configuration controller(s) and (iv) interface to the external storage of VHCs (e.g., host PC) with associated interface controller.

The actual system architecture of the RCS with the ability to support the virtualization of homogenous FPGA resources has been implemented in the form of a custom prototype board, the multitask adaptive reconfigurable system (MARS). This RCS has been designed around a Xilinx Virtex-4 FPGA device (XC4VLX160) with tile-based architecture. The platform and layout of major elements of the above architecture is shown in Figure 2. The MARS platform was designed for multistream high data rate video-processing applications. In the current configuration, MARS provides a wide variety of interfaces to video-sources and video-displays. It contains 4 SVGA output ports, one LVDS input port with bandwidth up to 2.128 Gb/s (16 bit × 133 MHz) and a universal VME-bus interface. The Main VHC memory was built on 4 NOR-Flash memory modules. The parallel read of 4 × 16 bit data words from these modules allows the controller-loader to provide the required timing for (a) accelerated parallel load of configuration bit streams for the entire FPGA configuration via SelectMap32 (3.2 Gb/s) and (b) parallel load of VHC bit streams for partial reconfiguration of selected FPGA slots via SelectMap-8 (800 Mb/s). The VHC-cache circuitry was not utilized at this stage of experiments.

3. On-Chip Architecture Organization

The initiation of task **Ts***i* consists of the configuration of the static part of the associated **ASVP***i*. The static part of the ASVP consists of the following elements: (a) Interfaces (IOBs) to external devices (e.g., video sensors, VGA-displays, etc.), (b) clocking and synchronization circuits, (c) data-memory units (BRAM based) and (d) communication infrastructure between VHC slots, I/O ports and data-memory units. Once the static portion of the ASVP is in place, the set of VHCs associated with the required mode of operation are loaded to predetermined VHC slots.

In Figure 3(a) a sample on-chip microarchitecture of a 4-channel parallel video-processing ASVP is shown. The upper and bottom-left boxes in the post-place and route layout show the *static part* of the ASVP design. The dotted boxes (bottom right) indicate the areas reserved for the VHC slots. In this picture VHCs are already loaded into the slots but do



(a) Reconfigurable design (b) Fixed design

Figure 3: (a) Sample of ASVP design for the current mode of a task, (b) Fixed design implementing all modes of a task.

Figure 4: Experimental setup for multistream/multimode Video-processing on the MARS platform.

not use the full amount of reserved resources. To analyze the effectiveness of the system a series of experiments for parallel video stream processing have been performed. One of the examples considered 2 parallel video streams (from a custom stereo vision sensor) to be processed on 4 video processors each of which could work in one of 7 modes of operation. Output streams have been directed to four SVGA monitors. The setup of this example is shown in Figure 4.

All modes and resource utilization for the associated VHCs for this example are listed in Table 1. In this case, each mode was associated with a certain VHC (function-specific video-processing circuits) to be configured in one of the above VHC slots (from Slot 0 to Slot 3).

To analyze the resource utilization, a comparison was made with a fixed (nonrecon19.9figurable) design. The fixed design reflected the traditional approach for static FPGA system-on-chip development. This design incorporates all video processors accommodating all modes of operation in one application-specific circuit. The on-chip layout of the fixed design is shown in Figure 3(b). In the presented example the ASVP based approach never exceeded 27% of the area utilized by a fixed design with the same function-

Table 1: Modes and resources for associated VHCs.

Mode	Required resource 4-input LUTs	Resource utilization 4-input LUTs
Camera display	172	33.6%
Normal edge	177	34.6%
Inverse edge	174	34.0%
PingPong edge	341	66.6%
Color intensity (Red)	102	19.9%
Color intensity (Green)	102	19.9%
Color intensity (Blue)	102	19.9%

alities. An increase in cost-effectiveness is gained because of virtualization of actual processing circuits. In other words, instead of keeping all processing circuits in real hardware (as for the fixed design approach) they can be stored in the form of configuration bit streams in Main VHC memory.

The same is true for power consumption minimization. A 29.8% reduction in power consumption was measured for the ASVP-based design (1.563 W) compared to the fixed system (e.g. 2.03 W). This reduction can also be attributed to lower resource utilization in the temporal domain. The power consumption was measured directly from the power supply (before the on-board DC-DC converters and power regulators). Initially power was measured with an "empty" system before configuration of the design. Then, power consumption was measured on the system with loaded fixed system design and consequently for each mode of the ASVP based design. The difference of power consumption between the "loaded" system and "empty" system was considered as the power consumption overhead associated with the loaded design

Obviously, the effectiveness of the proposed approach directly depends on workload specifics. However, it is necessary to mention that the effectiveness of the resource utilization increases proportionally to the number of modes in the task as well as the number of independent tasks running in parallel and using the same VHCs.

On the other hand, the cost-efficiency of the proposed approach directly depends on the cost of the hardware overhead. The additional cost of the memory hierarchy and associated controllers (Figure 1) increases the cost of the platform. However, the minimization of resource utilization by using the ASVP approach allows processing of the same workload on a smaller FPGA device. This, in turn, can bring a dramatic reduction of system cost. For example, the Xilinx XC4VLX160 (~16 M system gates) unit cost is around $5,600 USD (depending on number of I/O, package type, etc.) while a device from the same family which is four times smaller, the XC4VLX40, costs around $800 USD (http://avnetexpress.avnet.com/—list prices as of March 8th 2010). Comparing with the above cost reduction the hardware overhead cost required for configuration memory hierarchy can be considered as negligible. In our case, the hardware overhead cost did not exceed $200 USD including: 1GB Flash-based Main VHC memory, 4 MB SRAM-based VHC Cache, Xilinx XC95288 CPLD-based

Controller-Loader and PIC/USB interface to host computer. This is about 4.2% of the FPGA cost reduction considering Xilinx Virtex-4 or Virtex-5 FPGA families.

In addition to the cost of the hardware overhead in the system-level architecture, the effective implementation of the above concept of virtualization of FPGA resources directly depends on the mechanism for run time on-chip ASVP assembling. The foundation of this mechanism is a communication infrastructure which allows the allocation of VHCs to any VHC slots and further run time relocation of VHCs to new locations. This aspect of resource virtualization is discussed in the next section.

4. ASVP On-Chip Assembling and VHC Relocation Mechanisms

The major part of the research into the virtualization of hardware resources deals with the topic of run time on-chip ASVP assembly. This process is based on run time allocation and further relocation of VHCs. This research is important in its own right, as it increases the flexibility of a system, and adds additional fault-tolerance capabilities. However, some of the subjects encountered when implementing relocation-enabled systems are applicable to VHC-based systems in general. The remainder of this section will present a series of relocation experiments aimed at tile-based FPGA systems. In particular, the Xilinx Virtex 4 FPGA family is used in the experiments properly. This section will describe the experimental system used for the experiments, and will describe the obtained results. In addition, an on-chip support infrastructure which allows component relocation is introduced and described; as well, some design guidelines are discussed regarding the implementation of the VHCs themselves.

To perform relocation experiments, a dedicated system was designed and constructed from the ground up. The system incorporates many design decisions which were dictated by the requirement for relocation and dynamic partial reconfiguration, as dictated by the xilinx partial reconfiguration flow [13]. The system performs video processing operations on an acquired video stream and displays the results using a VGA interface. The video stream has a resolution of 640×480 pixels and a frame rate from 30 fps up to 200 fps. The stream can be displayed as it is, or a number of spatial color masks can be applied to it prior to display. The number and type of masks are variable, and the mask modules themselves are implemented as Virtual Hardware Components. For the first round of experiments, two types of mask VHCs were implemented: a smoothing mask and a y-dimensional Sobel filter.

4.1. Experimental System-on-Chip Architecture. As mentioned above, the implemented video processing system can display an unaltered VGA video stream, or it can apply two types of spatial masks to the stream, in various configurations. The experimental setup consists of a camera board, the MARS platform and a host computer. A block-diagram of the microarchitecture of a system with 4-VHC slots is shown in Figure 5.

The acquisition and display units are static components of the system, as well as the communication infrastructure. Together, the acquisition and display units act as the interface between the camera board, a VGA capable screen and the video processing components. All I/O signals (with the exception of reset and clock signals) are connected to these two units. The VHC slots in the system can accommodate one dynamic component each, and provide the actual video processing capabilities of the system. Finally, the communication infrastructure routes data between the various components in the system. All the above static elements are incorporated into the initial static architecture of the ASVP and are loaded into the target FPGA at start-up time. The host computer performs two tasks: configuration and control. At this stage of the experiments the FPGA JTAG configuration interface was used to perform all configuration activities, either full or partial. To control the communication interconnect, the host computer makes use of a serial interface and the microcontroller present on the MARS platform.

The complete system is conceived to operate as a deep data-processing pipeline. As virtual components are added or removed from the system, the depth of this pipeline varies. The components themselves are pipelined in nature, and the same basic interface is shared by all components in the system. This ensures that each component can be placed at any location in the system, and the pipeline structure will not change. By adopting this approach, the 30-frames-per-second frame rate of the acquired video is maintained regardless of the number of masks being applied. A deeper pipeline will lead to an increase in initial latency, but the data throughput will remain unaltered.

Two variations of the described system were implemented for experimentation. Both system types are identical in nature, with the only difference being the number of VHC slots present and the structure of the communication infrastructure. The first implemented system contains only four VHC slots; a second version of the system contains eight VHC slots. Two versions of the communication infrastructure were implemented, one for each system, and each will be described in detail in the following section. The post-place-and-route diagram of the four-slot system is shown in Figure 9, while that of the eight-slot system is shown in Figure 13.

4.2. Communication Infrastructure. The communication infrastructure plays a vital role in the operation of the complete system. Most obviously, it provides programmable communication paths between components. There are multiple methods available for providing communication paths between multiple on-chip components. These include the shared bus, the fully and partially connected crossbar, as well as various hybrid architectures.

For the proposed video system a fully connected crossbar was used. This architecture was selected as it provides complete connectivity inside the system, and minimizes latency due to resource sharing. The basic architecture of the cross bar is shown in Figure 6.

The core of the system is composed of a collection of multiplexers, each of which can route any cross bar input to any

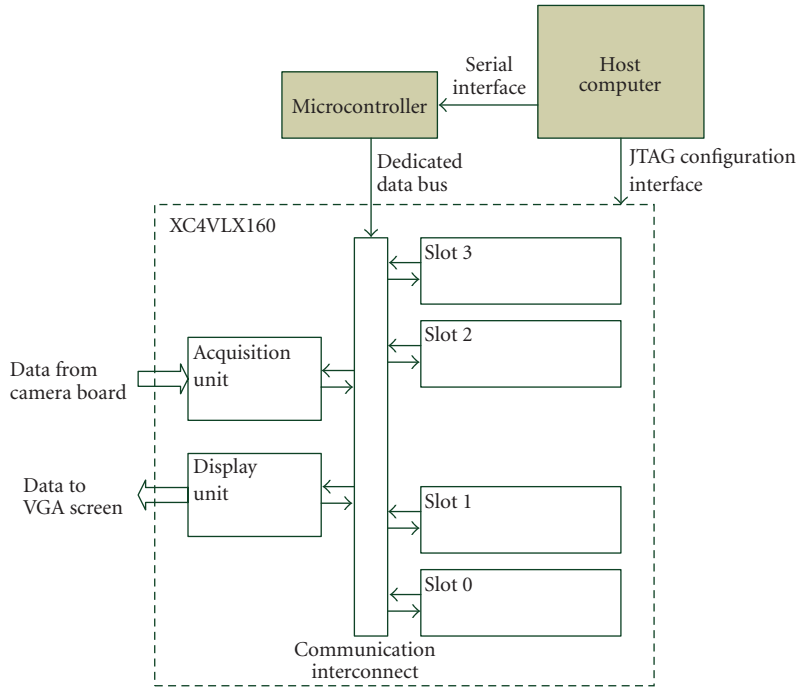


Figure 5: Four VHC slot microarchitecture.

output. As with many other communication protocols, separate ports are used for input and output. The cross bar is controlled by setting the multiplexer control signals to the appropriate value. The problem of communication delay variation is unique in on-chip systems; in a traditional design, the circuit remains static once implemented, and there can be no delay variation. However, in partially reconfigurable systems using VHCs, such delay variations can occur, when a component is relocated. The concept is illustrated in Figure 7, which shows the variation in delay between different slot positions; if a component is relocated from Slot 0 to Slot 3, the communication delay between it and the static part of the system will change. To address this issue, the concept of Latency Insensitive Design [14] has been applied to the architecture of the communication infrastructure. Essentially, the infrastructure is transformed into a pipeline; registers are added at the inputs and outputs of all combinational circuits (the multiplexers in this case), and all nets are segmented into smaller sections, connected to each other using registers. When a component is relocated, the result is that the number of registers (pipeline stages) between it and the rest of the system changes, as illustrated in Figure 8.

The conversion of the communication infrastructure into a pipeline works properly in situations where communications occur in one direction (such as stream situations with no hand-shaking protocols). For further experiments the four VHC slot communications infrastructure based on the architecture shown in Figure 6 was implemented in two versions: (a) a VHDL implementation with automated compilation to the configuration bit-stream, and (b) a mixed implementation combining both VHDL and manually placed and routed circuitry (Figure 9).

The reason for two implementations is that the infrastructure timing is critical to the correct operation of the system. The combinational or net delay between registers must be controlled for all connection paths inside the cross bar. The circuit diagram for the manually implemented interconnect is shown in Figure 10. In both cases, the control circuitry was implemented using VHDL, which consists of a finite-state machine and a collection of registers which store the control settings for every multiplexer in the cross bar.

The communication infrastructure of the eight-slot system was designed using a two-level hierarchical structure. Essentially, a nine-port cross bar was implemented using three five-port cross bars. The structure of this composite cross bar is shown in Figure 11. This design was adopted to determine what effect the increased complexity of the communication infrastructure would have on the performance of the system. The implementation of the eight-slot interconnect is a combination of manual and automatic synthesis. As the number of slots being connected rises, the complexity of the infrastructure also increases.

This leads to a very large increase in implementation time when manual placement and routing are performed.

To alleviate this problem, the interconnect was divided into two parts: the three cross bars used to implement the nine-port cross bar, and a dedicated connection bus, which is used to connect the nine-port cross bar to the ports of the VHC slots. The connection bus was implemented manually, while the hierarchical cross bar was implemented using automatic synthesis, placement and routing. The final structure of the interconnect structure is shown in Figure 12. The post-place-and-route circuit of the 8-VHC slot communication infrastructure is shown in Figure 13.

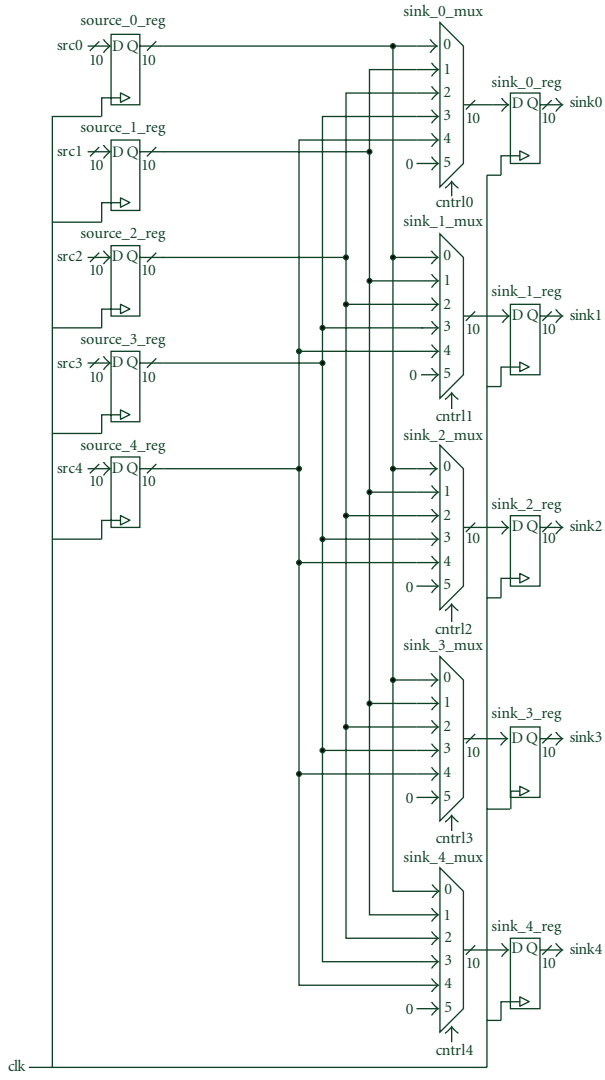


Figure 6: Five-port crossbar switch.

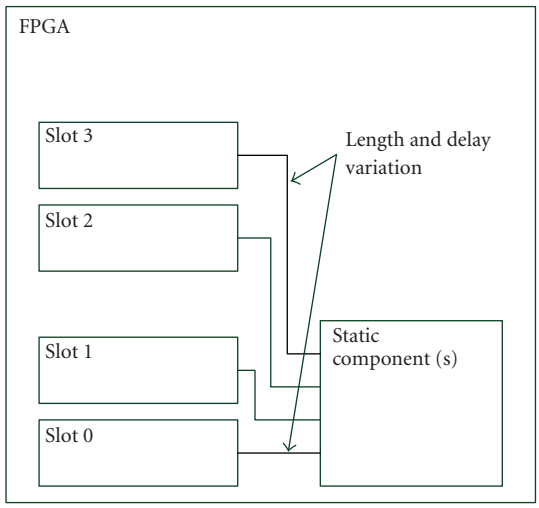


Figure 7: Delay variation in different slots.

4.3. Virtual Hardware Component Organization. This section will concentrate on the architecture organization of VHCs as dictated by the requirements for relocation. However, the internal circuitry of the particular VHCs will not be analyzed in detail here, as that would be outside the scope of this paper. In this consideration VHCs are determined as self-contained data-processing circuits which could meet the partial reconfiguration requirements of Xilinx Virtex FPGA devices [13]. The VHCs are composed of a pipelined data-path, a control unit, and local storage consisting of four RAMB16 blocks [15]. This structure implies that all VHC slots must contain the required memory resources, in addition to the homogenous resources used for general logic. By the same token, however, none of the components require external support ports, a fact which simplifies the design of the communication infrastructure (and the whole system). This approach, however, imposes certain limitation on the application.

This design constraint is adopted only temporarily and was accepted only insofar as it simplifies VHC design at the current stage of research. In further research stages this limitation is going to be removed.

The interfaces used by the components are reduced to the bare minimum needed for the system requirements. Each component has a sink interface and a source interface, the sink acting as the input to the component, and the source as the output from the component. Both the sinks and the ports have the same structure: each port consists of an 8-bit data bus and two synchronization signals. The data bus is used to transfer pixel information from one component to the next; one pixel color sample is transmitted every two clock cycles. The beginnings of each line and of each frame are signaled by the horizontal and vertical synchronization signals, respectively. The communication interconnect described above simply routes these signals as an aggregated 10-bit channel between various slots. This type of communication protocol allows self-synchronization of VHCs during run time relocation. In other words, if the data execution process stops between frames (as indicated by the vertical synchronization signal) VHC relocation can be initiated without data corruption. When the relocation process is complete the relocated VHC(s) can be activated by the same vertical synchronization signal. In video-processing systems the period of VHC relocation can be limited to the interframe period (seamless relocation) or to the period of one frame.

In the case of the video-processing applications considered in our experiments, the mask components are built to automatically synchronize themselves to the rest of the system using the vertical and horizontal synchronization signals. Once the vertical synchronization signal is received by a mask, it will start reading in data, one row at a time. After every row, the mask will stop and wait for the horizontal synchronization signal to arrive, before it will start reading in the next row. At the same time, once two rows have been buffered, the mask component will start generating output information, at the same rate that data is coming in; similar synchronization signals will be generated as outputs for the component downstream. Because of this, a component will stop operating if no vertical and horizontal signals are

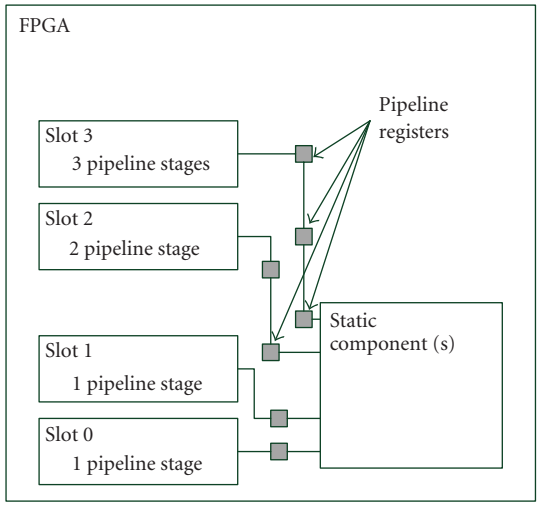


Figure 8: Latency variation in different slots.

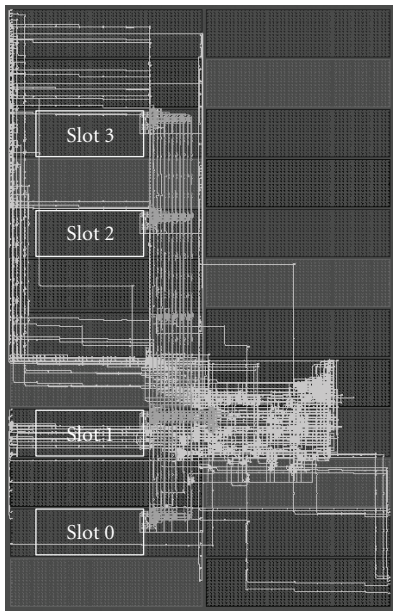


Figure 9: Four-VHC slot interconnect (VHDL and manual implementation).

received. In essence, by driving constant 0 values on the two synchronization signals, the components will become disconnected from the rest of the system, and stall. Similarly, once relocated and reconnected back to the system, the components will wait for the next frame before beginning operation.

It is necessary to mention that self-synchronization of VHCs is not limited to video stream applications. Generally speaking, any application dealing with streamed data can use this approach. This is possible because the data-stream usually has a specific structure which including synchronization periods, packet length, and so forth. However, since the data-stream structure depends on a given application, the self-synchronization mechanism should be customized to address this specific structure.

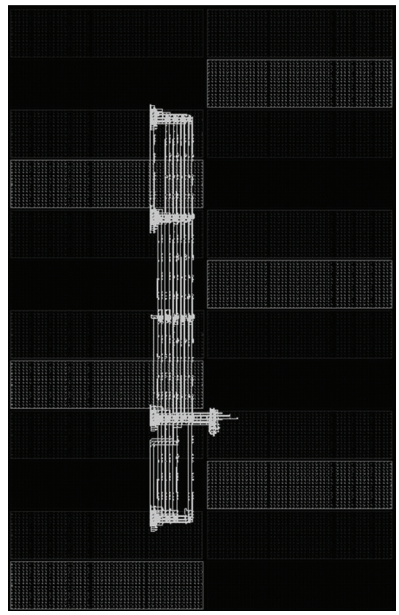


Figure 10: Four-VHC slot Interconnect (VHDL and manual implementation).

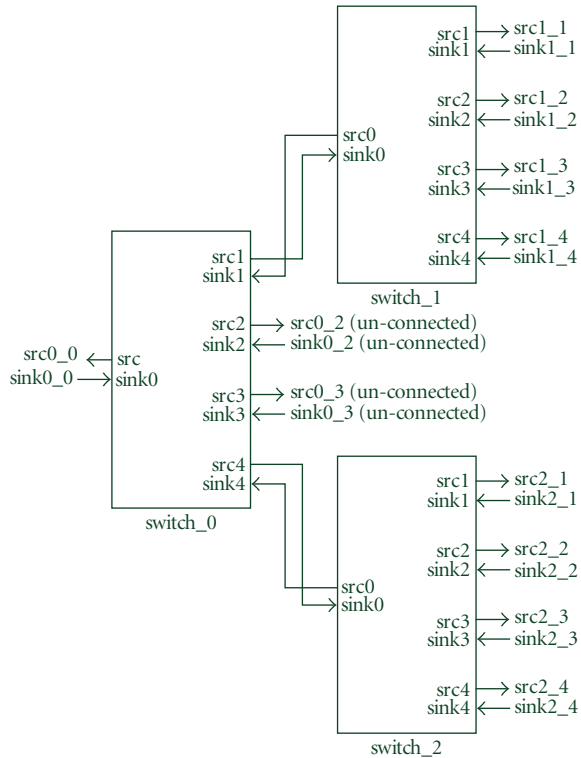


Figure 11: Eight-slot hierarchical switch.

4.4. Relocation Process Analysis. Once the on-chip communication infrastructure and VHC organization have been described, the relocation process can be analyzed in detail. Based on the systems described above, a series of relocation experiments were conducted. These experiments consisted of loading VHCs (masks) into all VHC slots to ensure

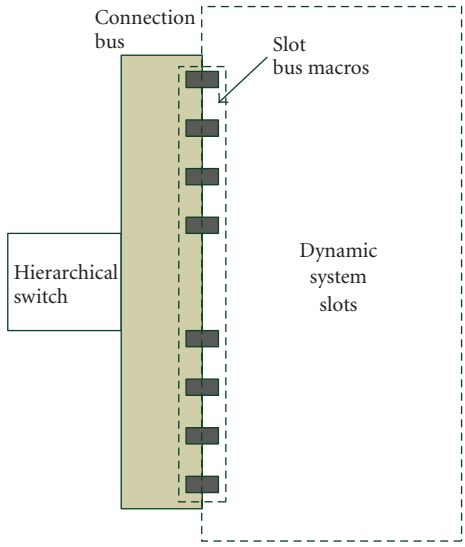


Figure 12: Eight-slot interconnect structure.

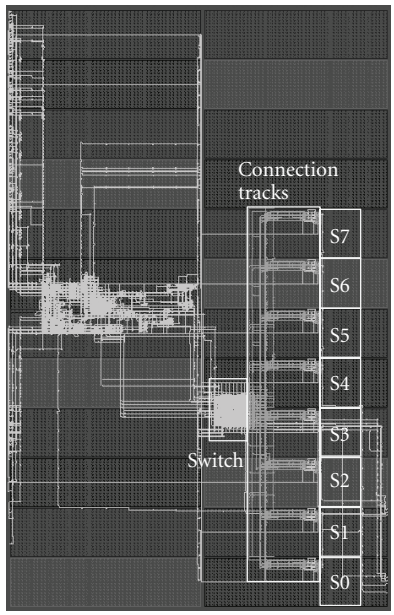


Figure 13: Post-place-and-route circuit of 8-VHC slot interconnects structure.

each mask/slot combination worked. As well, masks were relocated during system operation, to ensure that the process works correctly. For this round of experiments, the approach used was to generate multiple configuration bit streams for every VHC, one for each VHC slot in the system. The results of these experiments consist of timing information regarding the relocation process and the performance of the communication infrastructure, as well as resource utilization figures for the additional infrastructure needed to support VHCs and relocation.

4.4.1. Relocation Process Timing Overhead. The relocation process itself consists of a number of steps. To determine the total timing overhead needed to relocate a component, each step will be analyzed, and the timing information will be extracted. The steps needed for the relocation process are listed below.

(1) Disconnect (and stall) the downstream component from the source of the component being relocated.

(2) Configure the FPGA using the VHC bit-stream targeting the destination slot.

(3) Connect the relocated VHC sink to the source of the upstream component.

(4) Connect the downstream component source to the sink of the relocated component.

Of the above steps, 1, 3 and 4 are identical, essentially consisting of sending control information from the host computer to the reconfigurable platform, and waiting for the communication infrastructure itself to react. For the four-slot system the control information consists of two bytes of information, which are processed by the on-board controller and transmitted to the FPGA system. The time needed to transmit and process a control sequence is of 156 μs. The infrastructure itself requires less than 1 μs to react to the received control information. Steps 1, 3 and 4, require less than 160 μs to complete individually. All three steps together require less than 480 μs. It must be noted that this delay is due in very large part to the communication delay over the serial connection. If the control mechanism resided on-board, and the serial transmission times were eliminated, as much as 85% of this overhead could be eliminated. In the case of the eight-slot system, three bytes of information are transmitted from the host computer to the system for each port being set in the communication infrastructure. Since a hierarchical architecture is used, two ports must be set for each of the three steps described. In total, 422 μs are needed for each step, leading to a total time overhead of 1.27 ms. Once again, this overhead is due primarily to the use of a host computer and serial communication. If an on-board control mechanism were used, and serial communication eliminated, this overhead would be reduced by more than 95%.

The only remaining step to consider is step 2, the partial configuration of the FPGA itself. As mentioned previously, the JTAG configuration interface was initially used for this round of experiments. The sizes of both a complete bit-stream and a partial bit-stream are listed in Table 2. As well, the configuration times associated with each bit-stream are listed. Finally, to show the impact that configuration time has on systems using VHCs, the estimated configuration time for SelectMAP32 configuration is also listed, assuming a maximum configuration frequency of 100 MHz [16]. As can be seen, the configuration time currently dominates the relocation time. However, if SelectMAP32 were used, steps 1, 3 and 4 would require more time than the actual configuration process for a partial bit-stream. In such a situation, using an on-board controller would be beneficial.

Table 2: Bit-stream size and associated configuration time.

	Size (bytes)	JTAG (s)	SelectMAP 32 (ms)
System	5,043,464	6.73	12.6
VHC	90,578	0.12	0.23

Table 3: Communication interconnect hardware resource requirements.

Component	System gates	4-input LUTs	Slice Flip-Flops
INT2	660	30	60
INT4	2,173	154	154
INT4_M	—	394	362
INT8	6,032	569	337
INT8_M	—	569	817

4.4.2. Communication Interconnect Performance. A second set of experiments associated with the timing characteristics of the communication infrastructure were performed. In this case, the aim was to determine how much of a timing limitation the infrastructure would be to a system. This was done by determining the maximum operating frequency (the maximum clock frequency at which the communication infrastructure would still yield correct results). The experiment was conducted by building a test system consisting of the communication infrastructure under test, a simple pattern generator and an on-chip logic analyzer (ChipScope). This test system was implemented as a physical circuit on FPGA and the logic analyzer was used to observe data entering and leaving the communication infrastructure. The clock frequency of the system was increased by constant increments until synthesis could no longer be completed or until incorrect results were observed using the on-chip logic analyzer. It was found that both four-slot infrastructures (manual and automatic) as well as the eight-slot infrastructure were able to operate up to a frequency of 425 MHz. In the case of the eight-slot infrastructure, the two furthest slots in the system (slot 0 and slot 7 in Figure 13) were used to house the pattern generator and the logic analyzer. Despite this, the system offered the same performance as the four-slot system thanks to the use of additional registers in the longer communication paths.

4.4.3. Communication Interconnect Hardware Overhead. To support VHCs, a system must incorporate dedicated infrastructure circuitry, as was shown in the system above. This infrastructure fulfills an important role for the system, while at the same time not contributing anything to the actual processing performed by the system. Because of this, it is important to ensure that the infrastructure itself does not add too large an overhead to the rest of the system. In the case of the systems presented here, the communication infrastructure provides the support infrastructure needed by the system. Table 3 lists resource figures for various implementations of the communication infrastructure; 2, 4 and 8 slot versions are listed. In addition, for the 4 and 8 slot versions, data is provided for both the VHDL-only as well as VHDL and manual placement versions of the circuit. In the table, INT2 refers to a two-slot infrastructure (3 ports), INT4 refers to the four-slot infrastructure and INT8 refers to the eight-slot infrastructure; all these circuits are implemented in VHDL only, and use automatic synthesis. INT4_M refers to the four-slot infrastructure implementation which uses VHDL as well as manual place-and-route. INT8_M refers to the eight-slot infrastructure implementation using VHDL and manual place-and-route.

To determine how large the infrastructure variations are we can compare them with one of the system slots. One slot in the four-slot architecture contains 1,664 slices, or 3,328 LUTs and 3,328 1-bit flip-flops. In addition to this, each slot contains eight RMAB16 blocks and four DSP blocks; however, since the infrastructure variations do not use these resources, they will not be considered in this analysis. The most resource-heavy infrastructure in the above table is the eight-slot variation implemented using both VHDL and manual place-and-route. Nonetheless, when compared with the resources available in one system slot (which takes up less than one sixteenth of the device), we see that the communication infrastructure would account for only 21% of the total resources available in one VHC slot (counting all LUTs and slice flip-flops together) or less than 2.7% of the total resources reserved for all eight slots.

Summarizing the above, the proposed organization of on-chip communication infrastructure can provide run time adaptation of the FPGA-based stream processing system on the current set of tasks (current workload) as well as run time mitigation of hardware faults caused by various factors. On the other hand the cost-effectiveness of multimode/multistream processing systems can be increased by reducing power consumption compared with existing design methodologies. To make possible the above features the following aspects have been analyzed and tested: (i) flexible allocation and run time on-chip relocation of virtual hardware components, providing a mechanism for run time on-chip assembling of ASVPs; (ii) an operational frequency range up to 425 MHz for both 1-level (4-VHC slot) and 2-level (8 and 16 VHC slot) hierarchical organizations of the communication infrastructure in tile-based Xilinx Virtex FPGA devices; (iii) seamless relocation of VHCs using the mechanism of self-synchronization embedded in the VHC architecture; (iv) relatively small hardware overhead for the communication infrastructure (not exceeding 2.7% of resources allocated for 8-VHC slot structures).

All of the above gives us the strong belief that the proposed approach can be an effective solution for a wide variety of applications associated with high data rate processing of streamed data. Some of these applications will be discussed in the next section.

5. Industrial Applications

The presented research was oriented towards several industrial applications which required high data rate processing in many different modes. Furthermore, high reliability and low power consumption were important requirements for

these applications. In addition to that, minimal dimensions and weight (mass) were necessary. One of these applications is the next generation of space-born computing platforms for orbital and interplanetary missions. The initial focus was on high-frame rate machine-vision platforms for automated docking of satellites as well as automated object grasping using space manipulators (e.g., CanadArm-2 on ISS). This type of applications requires (i) multimode parallel video stream processing with high-frame-rate requirements (from 30 to 200 fps), (ii) automated switching from one mode to another within a relatively short time (milliseconds), (iii) parallel processing of several tasks allowing run time distance measurement, object tracking, image stabilization, video-compression and many other algorithms, (iv) minimum mass, dimensions, power consumption and power dissipation, (v) the ability to mitigate transient hardware faults caused by cosmic radiation effects (e.g., SEU—Single Event Upset) as well as restoration from permanent faults. Additionally, the platform architecture should allow remote upgrade of hardware and software components and rapid adaptation for new task algorithms and data structures.

The above requirements are usual for any autonomous embedded system oriented to work in a harsh environment (e.g., control and robotic systems for nuclear power stations, etc.).

Similar platform specifications can be found for satellite communication and broadcasting systems. Recently, these systems are oriented on broadcasting digital video streams on mobile and handheld devices. This requires utilization of complex COFDM modulators, which usually are developed around special chipsets and a number of statically configured large FPGA devices. The example here is DVB-S/H (Digital Video Broadcasting—Satellite-to-Handheld) systems. It is necessary to mention that the MARS platform was developed and manufactured as part of an RandD project oriented towards the creation of the next generation of platforms for DVB-S/H COFDM modulators.

Most recently, our implementation efforts of the above research are concentrated on the creation of the next generation of space-borne platforms for machine-vision. Several stereo vision sensors and associated VHCs have been developed and tested during a Collaborative RandD project associated with high-frame rate tracking of 3D objects. A special experimental setup has been recently designed and assembled on the basis of a 5-axis space platform where the high-frame-rate stereo vision sensor was installed for experiments on automated pose recognition and tracking. This setup is shown in Figure 14.

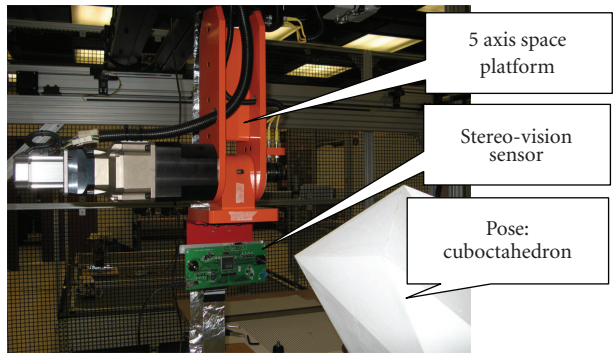


Figure 14: Experimental setup for autonomous docking of space structures.

6. Summary

A mechanism for the virtualization of computing resources for multitask and multimode stream applications has been developed and investigated in detail in this work. The major concept of the proposed approach is to assemble application specific virtual processors (ASVPs) at run time inside the FPGA rather than the traditional approach based on off-line compilation. The ASVP can be assembled using a field of homogenous computing resources in the FPGA and a configuration which reflects the current set of tasks and their modes. The building blocks for the ASVP are virtual hardware components (VHCs)—macro-function-specific stream processing pipelines. The VHCs are self-containing modules with embedded SRAM blocks, control FSM and synchronization circuits. A VHC becomes a real processing module only when configured in a specific slot of the target FPGA. Otherwise, the VHC is stored in one of multiple levels of a memory hierarchy: VHC cache, main VHC-memory or secondary storage. This organization of the system level architecture allows a dramatic minimization of actual hardware resources and power consumption as well as cost and dimensions of the RCS. The comparative analysis made on the multitask adaptive reconfigurable system (MARS) demonstrated negligible hardware overhead offset by sufficient gains in the main performance parameters. This experimental platform incorporates all required components needed to provide hardware support for virtualization of FPGA resources. At the same time, the on-chip communication infrastructure has been designed to provide the same hardware support at the FPGA microarchitecture level. Different schemes and cross bar switch configurations have been analyzed and a hierarchical organization (2 levels) of the communication infrastructure with pipeline transition registers was selected. It was experimentally determined that this system can support operating frequencies up to 425 MHz and eliminates propagation delay variations even between the farthest slots. At the same time, utilization of self-synchronization circuits in the VHCs allowed seamless run time component relocation which provides more flexibility for a system. This same relocation procedure allows rapid restoration of any transient faults as well as mitigation of permanent hardware faults. The above relocation procedures have been experimentally tested using different FPGA ports for configuration as well as on-board/external control mechanisms. It was determined that using on-board configuration controllers in conjunction with parallel configuration ports of the target FPGA, the relocation process could be done in a relatively short time (less than 1 ms). The on-chip hardware overhead was also analyzed in comparison with the amount of resources available for VHC slots. In all experiments the on-chip communication

infrastructure never exceeded an overhead of 2.7% compared with the resources dedicated to data-processing.

Finally, the proposed mechanism has been implemented for different industrial applications and a high potential for increasing system cost-efficiency has been demonstrated. The major applications for the above approach are mobile autonomous embedded systems where multiple tasks associated with streamed data should be performed, each with high data-rate requirements.

Acknowledgment

The authors wish to acknowledge the support of NSERC of Canada, Canadian Space Agency (CSA), Ontario Centers of Excellence (OCE-CITO), MDA Space Missions, UBS Ltd., CMC Microsystems, and Xilinx Inc.

References

[1] E. Caspi, M. Chu, R. Huang, J. Wawrzynek, J. Yeh, and A. DeHon, "Stream computations organized for reconfigurable execution (SCORE)," in *Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing*, pp. 605–614, Springer, Berlin, Germany, 2000.

[2] S. Wallner, "A reconfigurable multi-threaded architecture model," in *Advances in Computer Systems Architecture*, pp. 193–207, Springer, Berlin, Germany, 2003.

[3] M. Mishra and S. C. Goldstein, "Virtualization on the Tartan reconfigurable architecture," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '07)*, pp. 323–330, Amsterdam, The Netherlands, August 2007.

[4] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Matt, and R. R. Taylor, "PipeRench: a reconfigurable architecture and compiler," *Computer*, vol. 33, no. 4, pp. 70–77, 2000.

[5] L. Kirischian, V. Geurkov, V. Kirischian, and I. Terterian, "Multi-parametric optimisation of the modular computer architecture," *International Journal of Technology, Policy and Management*, vol. 6, no. 3, pp. 327–346, 2006.

[6] H. Kalte, G. Lee, M. Porrmann, and U. Rückert, "REPLICA: a bitstream manipulation filter for module relocation in partial reconfigurable systems," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, pp. 151–158, Denver, Colo, USA, 2005.

[7] F. Ferrandi, M. Novati, M. Morandi, M. D. Santambrogio, and D. Sciuto, "Dynamic reconfiguration: core relocation via partial bitstreams filtering with minimal overhead," in *Proceedings of the International Symposium on System-on-Chip (SOC '06)*, pp. 1–4, Tampere, Finland, 2006.

[8] S. Corbetta, F. Ferrandi, M. Morandi, M. Novati, M. D. Santambrogio, and D. Sciuto, "Two novel approaches to online partial bitstream relocation in a dynamically reconfigurable system," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI: Emerging VLSI Technologies and Architectures (VLSI '07)*, pp. 457–458, Porto Alegre, Brazil, March 2007.

[9] T. Becker, W. Luk, and P. Y.K. Cheung, "Enhancing relocatability of partial bitstreams for run-time reconfiguration," in *Proceedings of the IEEE Symposium on Field-Programme Custom Computing Machines (FCCM '07)*, pp. 35–44, Napa, Calif, USA, 2007.

[10] 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 Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*, vol. 2006, pp. 97–102, Karlsruhe, Germany, 2006.

[11] C. Bobda, M. Majer, A. Ahmadinia, T. Haller, A. Linarth, and J. Teich, "The Erlangen slot machine: increasing flexibility in FPGA-based reconfigurable platforms," in *Proceedings of the IEEE International Conference on Field Programmable Technology*, pp. 37–42, Singapore, December 2005.

[12] L. Kirischian, V. Geurkov, I. Terterian, V. Kirischian, and J. Kleiman, "Multilevel radiation protection of partially reconfigurable field programmable gate array devices," *Journal of Spacecraft and Rockets*, vol. 43, no. 3, pp. 523–529, 2006.

[13] Xilinx, "Early access partial reconfiguration user guide, 1.2 edn," 2008.

[14] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli, "Theory of latency-insensitive design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 9, pp. 1059–1076, 2001.

[15] Xilinx, "Virtex 4 FPGA user guide, 2.5 edn," 2008.

[16] Xilinx, "Virtex-4 FPGA configuration user guide, 1.1 edn," 2008.