A Scalable Architecture to Support Networked Reconfiguration

Y. Ha[1*], P. Schaumont[1], L. Rijnders[1], S. Vernalde[1], M. Engels[1], and H. De Man[1*]

[1]IMEC vzw, Kapeldreef 75, B-3001, Leuven, Belgium
[*]Department of Electrical Engineering,
K. U. Leuven, B-3001, Heverlee, Belgium
Phone: +32 (0)16 281 566 Fax: +32 (0)16 281 515

E-mail: { yjha | schaum | rijnders | vernalde | engelsm | deman } @imec.be

Abstract— **This paper presents a proposed scalable architecture to enable networked reconfiguration for the next generation communication terminals. A new system architecture that supports networked reconfiguration is defined. It contains two new blocks - the virtual reconfigurable architecture (VRA) and the application specific resource manager (ASRM). VRA can be considered as a hardware virtual machine, and it separates the terminal architecture into application independent and application specific parts. ASRM is used to automatically manage FPGA resources similar to the way a conventional OS manages memory or CPU resources. By providing both a hardware and a software virtual machine, the networked reconfiguration users only need to develop a design description targeted on the virtual platform that exploits VRA and ASRM.**

Keywords— **FPGA; computer architecture; reconfiguration, communication terminals.**

I. Introduction

Communication protocol stacks are composed by different layers, from low to high (see Fig.1). Fixed hardware is generally used to implement the lower protocol layers since importance is placed on high-speed data transport processing rather than flexibility. However, today's obvious trend for the internetworking forces network systems to support a wide variety of communication protocols, including some parameters and work modes in their lower layers. On the other hand, up to now, higher layers of communication protocols have almost all realized as software components because they are complex and require frequent updating. But in the near future, some of the operations in higher protocol layers might need to be implemented as hardware to achieve adequate data manipulation rates. Thus, lower layer telecommunication protocols require more flexible implementation and higher ones will be forced to achieve better performance [1]. Therefore, an efficient way to fuse both hardware and software is becoming indispensable for building the next generation communication systems (see the right part of Fig.1). Field programmable gate array (FPGA) [2] is considered to be one good solution for this problem. It can be dynamically reconfigured to realize different functions on the fly. FPGAs provides system designer a good choice when compromising system performance in a level between hardware and software.

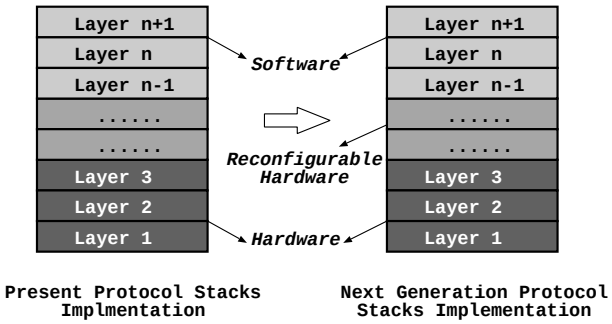

Fig. 1. Implementation of communication protocol stacks

But present FPGA system cannot support *networked reconfiguration*, which is desired in the new service deployment of next generation communication systems. In the networked reconfiguration, as shown in Fig2, the communication terminal provides customized data processing by first dynamically reconfiguring itself. Its reconfiguration information (agent) is coming from the service provider through network. The reconfiguration information is prepared by the servicer provider in a hardware agent format, which is abstract enough to be implemented on any FPGA platform, just like what Java software bytecode does. The hardware reconfiguration agent does not need to know that it will be implemented on which kinds of

ISBN: 90-73461-18-9

reconfigurable resources (e.g., which series of FPGAs from which vendors). Transported hardware reconfiguration information is corresponding to the requirements of a specific service. Therefore, after the terminal hardware is reconfigured, the service data are then processed in a customized way.

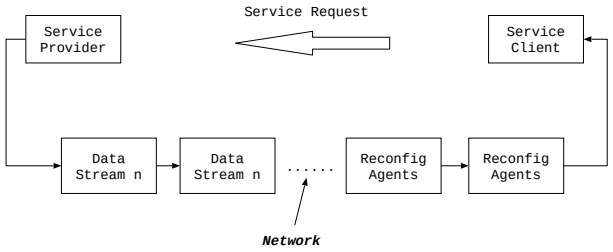

Fig. 2. Networked Reconfiguration

This paper tends to discuss our present ideas to implement the networked reconfiguration. In the next section, a proposed scalable terminal architecture is first given out. Based on the implementation requirements to the various blocks in the proposed scalable architecture, research challenges in realizing its blocks like VRA and ASRM are examined. In section 3, CAD design and implementation flow for the networked reconfiguration design is first described. The specification of an abstract FPGA model is then introduced. While in section 4, resource management mechanisms like in real-time operating system for CPU and memory are expanded in application-specific resource manager (ASRM) for reconfigurable hardware resources (RHW). Finally, two demonstrators are designed to prove the concepts developed in the previous sections.

II. System Architecture

As introduced above, the next generation communication terminals with the capability of networked reconfiguration is desired to receive the hardware reconfiguration information from the service provider through network, and configure itself on the fly according to the performance requirements.

An novel scalable architecture for the networked integrated service terminal is proposed as in Fig.3. Compared to the traditional terminal architecture, the new scalable architecture adds two new parts. They are virtual reconfigurable architecture (VRA), application specific resource manager (ASRM). The VRA and ASRM are specially added to support the networked reconfiguration.

VRA is a hardware virtual machine corresponding to the virtual machine in the software part. It works together with the software virtual machine to separate the whole architecture into application independent part and application specific part. Hardware and software bytecodes running on the hardware and software virtual machines respectively. By providing a virtual world, the service provider only need to provide service description targeted on the virtual platform comprised by VRA and VM.

In the traditional architecture, the ASRM is real-time operating system. But in this scalable architecture, we have both fixed application-specific hardware (FHW) and reconfigurable hardware (RHW). Therefore, a new resource manager that can manage this two parts is needed. This new resource manager is the ASRM.

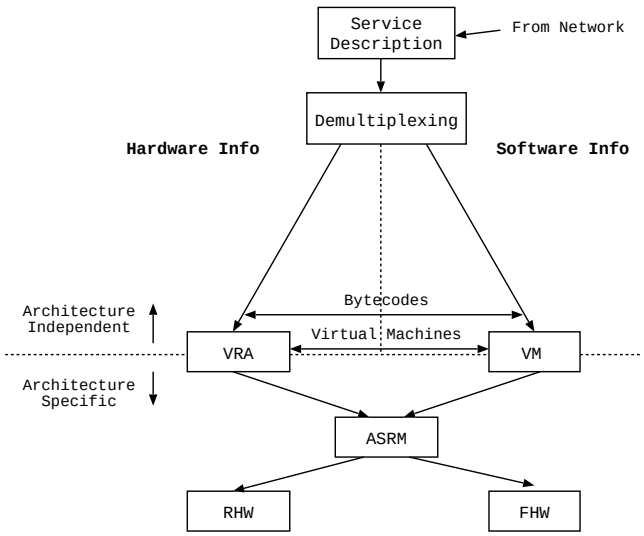

Fig. 3. Scalable architecture for networked integrated service terminal

III. Virtual Reconfigurable Architecture

Virtual reconfigurable architecture is one of the most important block in the scalable architecture Fig.3. In this section, a special design flow of FPGA applications in the networked reconfiguration is first described. Then an abstract FPGA model is introduced to support such a design flow.

A. Design Flow for FPGA applications in Networked Reconfiguration

To implement networked reconfiguration, a special design flow is modified as in Fig.4. This design flow is different and modified from the traditional CAD design flows which are done locally in one site [**?**][4]. The new whole implementation flow is divided into two parts. Most of the work is done in service

provider's side. Although that may influence the efficiency of the whole implementation, it greatly eases the jobs needed to be done in the client's side, and lowers its requirements for the computing resource.

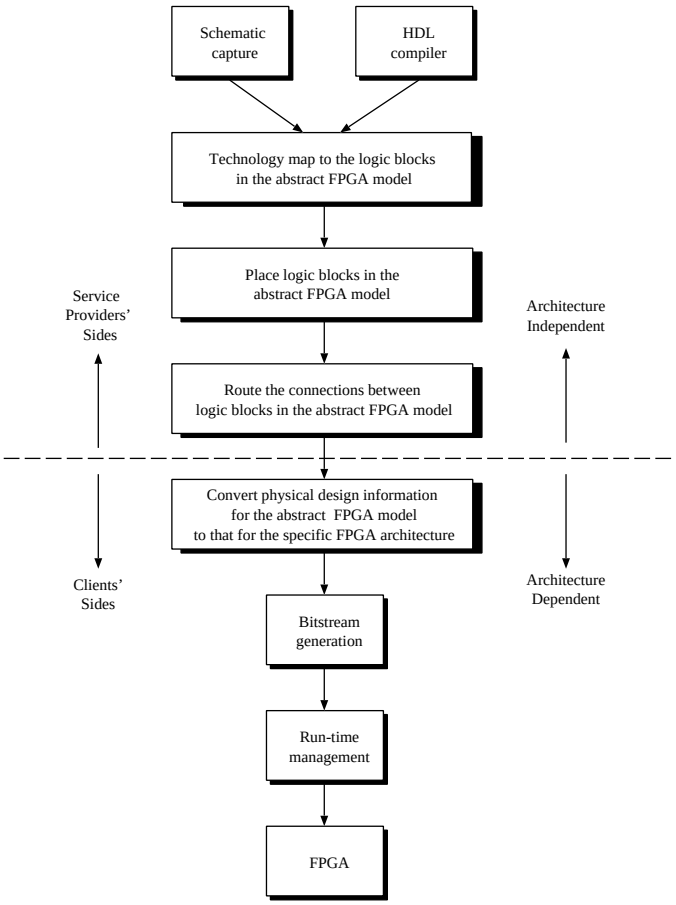

Fig. 4. An implementation flow for the FPGA applications in networked reconfiguration

As shown Fig.4, in In the service provider's side, an application design is first compiled and technology mapped to the abstract FPGA architecture model which will be discussed later. Then it will be placed and routed on the same model. The resulted physical design information will be written into a bitstream file, which then will be transported through the network to the client's side.

In the client's side, the received bitstream file is converted to the physical design information for that of the local FPGA architecture. After the bitstream has been generated, the networked reconfigurable hardware design will appear as one library of the system. This hardware library behaves like a software library, its implementation can be called through a software routine exactly the same as a software procedural call. The only difference is that the hardware procedural call is implemented in reconfigurable hardware resources (FPGAs). The usage of this reconfigurable hardware resources is managed by the ASRM.

B. Abstract FPGA model

As shown in the Fig.4, an abstract FPGA model is used as the core of VRA. Before going into the details of abstract FPGA model, it is interesting to highlight the following two facts:

- If an application wants to be implemented efficiently in the client's side, placement and routing should be completely, if not possible, mostly finished in the service providers' sides. Because the most time-consuming phase of the compilation is for placement and routing.
- Requiring placement and routing to be done in service providers' sides will bring new problems. Since placing and routing are generally a technology dependent step, it is almost impossible to realize that with the present design methodologies.

In the software virtual machine, people meet similar problems as in hardware virtual machine. On the one hand, to get the platform independent feature, the software codes running on virtual machines should be abstract enough. On the other hand, these abstract codes should be more efficient than the interpreter scripts. To meet this, people cleverly invented the abstract Java virtual machine.

Inspired by the software virtual machine, it is natural for us to think that, we can have an abstract FPGA architecture model. This model is the core of our VRA block in the scalable architecture. All the original circuit designs are first implemented in this abstract FPGA architecture model in the service provider's side. This half finished implementation is then transported through the network to the client. In the client's side, interpreter inside the VRA will convert this abstract design information to the detailed programming file that configures the specific local FPGAs.

The proposed abstract FPGA model is comprised by three parts, namely, the technology mapping model, FPGA architecture model [5], and routing model. The technology mapping model is used to map design logic into abstract look-up tables. FPGA architecture model contains physical information description of a FPGA architecture. Routing model describes the routing resources of a FPGA device, and is used in the routing phase. We will discuss the technology mapping model and routing model next.

B.1 Technology Mapping Model

Several steps should be done in the service provider's side for the technology mapping in the networked reconfiguration (see Fig.5). A logic design can always be represented by a logic network as shown in Fig.5(a) [6]. But computation complexity of operation algorithms on such kind of logic network is very high and generally NP-hard. To reduce the computing complexity, this logic network will be first converted from logic network to a forest of logic trees Fig.5(b). After that, many available efficient algorithms can be used efficiently and the resulting implementations are assembled together according to the interconnection patterns of the forest [7].

In one given tree, every subtree that has at most k leaf nodes can be implemented by a single LUT. Therefore, a node with more than k fan-in nodes are considered as *infeasible node* for a k-input LUT. For example, node O in Fig.5(b) is an infeasible node to a LUT with $k = 2$. As a result, before the LUT binding, all the infeasible nodes in trees will be first decomposed into several logic equivalent nodes as shown in Fig.5(c).

Because the abstract FPGA LUT should be general enough, its k is fixed to be 2. Each node in the preprocessed logic tree is then bound into a $k = 2$ LUT Fig.5(d)(e).

Placement and routing are done next to get the physical connection information. Those information is then back annotated into the same graph of Fig.5(e), and get the Fig.5(f).

B.2 Routing Model

Architectures of FPGAs can be described by *routing-resource graphs* (RRGs) [8], which are the internal representation for a router. Therefore, the translation from abstract FPGA routing information to local FPGA routing information becames a problem of translating the routing-resource graph of abstract FPGA to a routing-resource graph for the local FPGA(see Fig.6).

In the routing-resource graph $G_r(V, E)$, each wire and each logic pin becomes a *vertex* v_i, and each switch becomes an *directed edge* (v_j, v_i) (for unidirectional switches, such as buffers) or a pair of directed edges (for bidirectional switches, such as pass transistors) between the two appropriate nodes. Once the FPGA architecture has been represented in this form of general directed graph, algorithms in the graph theory field can be used or referenced to route the design.

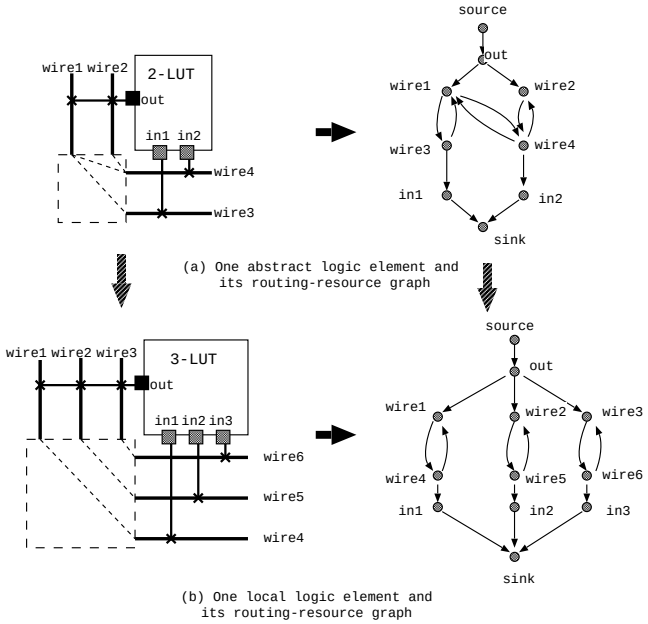

Fig. 6. Converting routing-resource graphs from abstract to local FPGA routing architecture

Fig.6a shows the routing-resource graph corresponding to a portion of an FPGA whose logic block is with 2 input pins, and 1 output pin. Suppose the logic block of Fig.6a is part of abstract FPGA model, while the logic block in Fig.6b is part of local FPGA model. The interpretation of the physical design information on abstract FPGA model to a local FPGA architecture means the translation of information on one routing-resource graph Fig.6a (abstract FPGA) to another routing-resource graph Fig.6b (local FPGA). But please note that, once the circuit has been totally routed onto the abstract FPGA routing architecture, its routing-resource graph can be cleaned by removing all the unused nodes (logic block and I/O pins) and edges (switches).

The abstract FPGA routing model can be defined as follows:

- A good abstract FPGA routing model should be able to be described by a routing-resource graph with very regular and concise topology. This topology of RRG will be important when translating physical design information from the abstract FPGA routing architecture to that of the local FPGA routing architecture.
- A interpretation of the physical design information from abstract FPGA routing architecture to that of the local FPGA routing architecture is equivalent to a mapping $G_a(V_a, E_a) \longrightarrow G_l(V_l, E_l)$, where $V_a \subseteq V_l$, and $E_a \subseteq E_l$ or there is a mapping $E_a \longrightarrow E_l$.

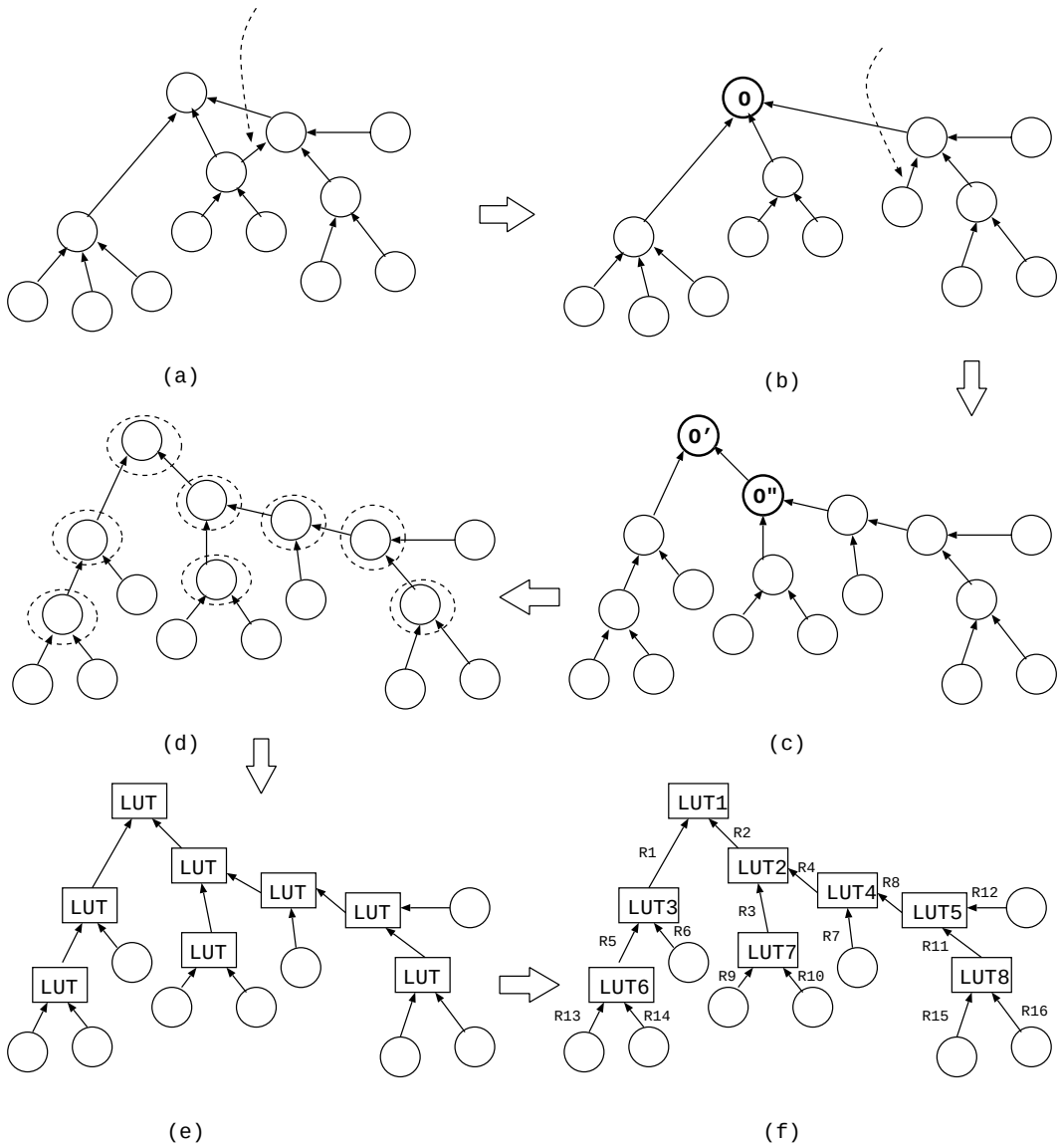

Fig. 5. Technology mapping in the networked reconfiguration

IV. Application Specific Resource Management

In the traditional architecture, the application specific resource manager is equivalent to the real-time operating system. But in the scalable architecture of Fig.3, two kinds of resources (as shown in Fig.7) needs to be managed. (The two resources are fixed application-specific hardware (FHW) and reconfigurable hardware resources (RHW) respectively). Therefore, a new resource manager should be able to fulfill not only the original functions of real-time operating system, but also some extra management functions for the added reconfigurable hardware resource.

This new resource manager is the ASRM in our scalable terminal architecture Fig.3. This section discuss in detail what functions are expected for the ASRM.

Please note that, in Fig.7, "hardware execution engine" does not mean that it is really made by hardware. The hardware execution engine is implemented by pieces of codes just like the software execution engine inside the Java virtual machine. "Hardware" here actually refers to the information running on the engine is hardware information. The core of this hardware execution machine is our abstract FPGA model.

In Fig.8, the layer structure of ASRM is given. Since ASRM is an adapted extension to the operating systems for the traditional computing architecture, we only focus on the new blocks that are created specially for the new hardware virtual machine block (VRA) and reconfigurable hardware resources (RHW).

The expanded new blocks of *FPGA management*, *virtual FPGA management*, *FPGA I/O* will be dis-

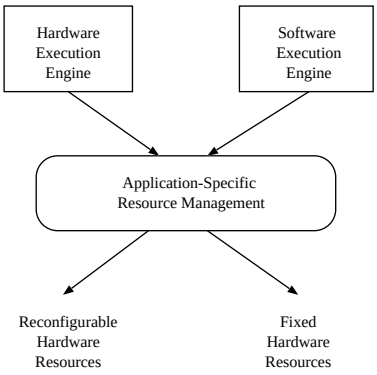

Fig. 7. A general look at the application-specific resource manager

cussed below. As expected, there is no block corresponding to CPU scheduling. Since in the hardware machine, all the resource have been scheduled inside the design, only the implementation positions on reconfigurable hardware resources should be scheduled by the ASRM. Therefore, most of the management in ASRM for RHW are extremely like memory management in the OS.

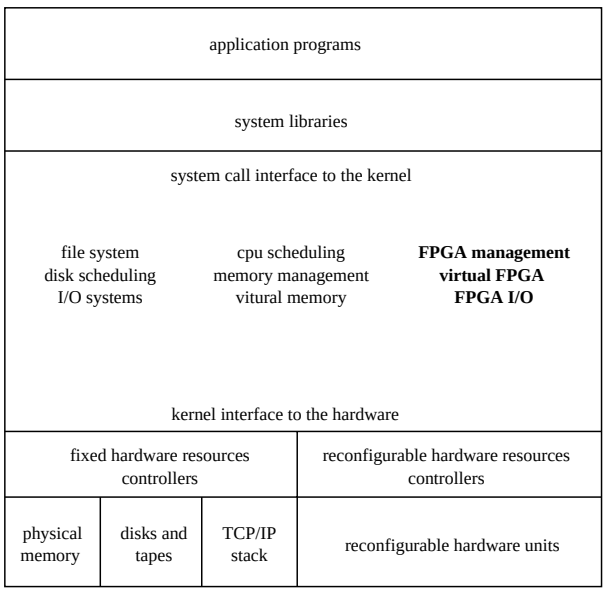

Fig. 8. The layer structure for the application-specific resource manager

A. FPGA management

FPGA management in the ASRM allocates the FPGA resources for several designs (hardware processes) which are running in the same reconfigurable hardware. Its functions are corresponding to the memory mangement in the operating systems.

B. Virtual FPGA management

Virtual FPGA management in the ASRM helps the realization of one big design without needing it to mapped to the FPGA resources completely at one time. Its functions are corresponding to the virtual memory management in the operating systems.

C. FPGA I/O

One of the purposes of the ASRM is to hide the peculiarities of specific hardware details from the user. For example, the FPGA pads will be assigned for multiple applications by the ASRM. Since reconfigurable hardware resources can be considered as a new kind of I/O devices, device drivers for the reconfigurable hardware resources should be added into the I/O systems of ASRM.

V. Demonstrators

To prove the concepts developed in the previous sections, two demonstrators are designed.

In the first demonstrator (see Fig.9), only one FPGA application will be running on the virtual reconfigurable architecture. The application desired to do networked is first written in the format *.vhd* or *.blif*, and then entered as input to the server part of VRA software VRA_S1. The VRA_S1 will process the design into hardware bytecode and sent through network to the client part of VRA software VRA_C1. VRA_C1 will be in charge of converting this abstract bytecode into routing results for the local FPGA.

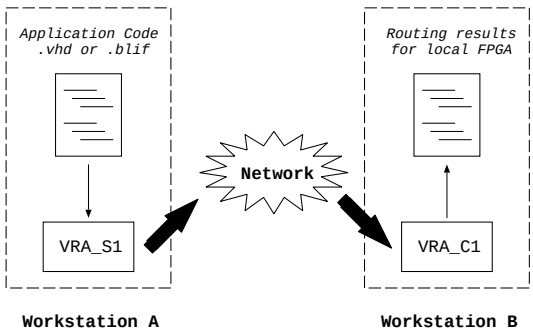

Fig. 9. Demonstrator of networked reconfiguration for a single application

In the second demonstrator (see Fig.10), multiple FPGA applications will be running on the same virtual reconfigurable architecture. But this time, the FPGA resources need to be scheduled and allocated by the ASRM.

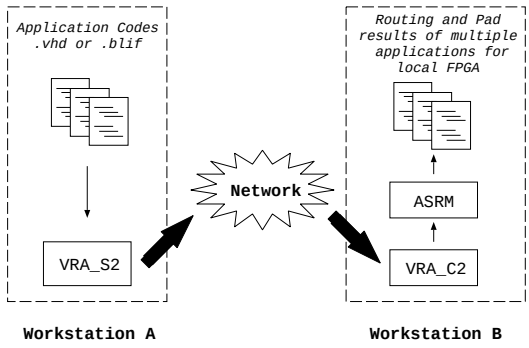

Fig. 10. Demonstrator of networked reconfiguration for multiple applications

VI. Future Work

An scalable terminal architecture to support networked reconfiguration had been proposed in this paper. We are going to further improve and implement it. Besides, a remote dynamical reconfiguration protocol will be developed and experiment together with the networked reconfiguration for the terminal architecture. More algorithms that consider FPGA architectures with heterogeneous logic and memory blocks are also planned.

Acknowledgments

The authors would like to acknowledge the discussions with many people in the DESICS division of IMEC.

References

[1] A. Tsutsui, T. Miyazaki, *ANT-on-YARDS: FPGA/MPU Hybrid Architecture for Telecommunication Data Processing*, IEEE Trans. on VLSI systems, pp.199-211, June 1998.

[2] S. Brown, R. Francis, J. Rose, and Z. Vranesic, *Field-Programmable Gate Arrays*, Kluwer Academic Publishers, 1992.

[3] S. H. Ludwig, *Hades: Fast Hardware Synthesis Tools and a Reconfigurable Coprocessor*, Ph.D. thesis, ETH Zurich, 1997.

[4] Xilinx Inc., *The Programmable Logic Data Book*, 1994.

[5] V. Betz and J. Rose, *VPR: A New Packing, Placement and Routing Tool for FPGA Research*, Int. Workshop on Field-Programmable Logic and Applications, pp. 213-222, 1997.

[6] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, Inc 1994.

[7] A. Sangiovanni-Vincentelli, A. El Gamal, and J. Rose, *Synthesis Methods for Field Programmable Gate Arrays*, Proc. of the IEEE, pp.1057 - 1083, July 1993.

[8] C. Ebeling, L. McMurchie, S. A. Hauck and S. Burns, *Placement and Routing Tools for the Triptych FPGA*, IEEE Trans. on VLSI systems, pp.473 - 482, Dec 1995.

This page was intentionally left blank