Remote SoC/FPGA Platform Configuration for Cloud Applications

Octavian Machidon[1], Florin Sandu[1], Corneliu Zaharia[2], Petru Cotfas[1], Daniel Cotfas[1]
[1]Department of Electronics and Computers, “Transilvania” University, Brasov, Romania
[2]Digital Optics Corporation, Brasov, Romania
E-mail: octavian.machidon@unitbv.ro, sandu@unitbv.ro, czaharia@doc.com, pcotfas@unitbv.ro, dtcotfas@unitbv.ro

Abstract- **The growing development of Cloud Computing raised the need of making hardware available “as a Service”. Reconfigurable hardware – like FPGA (Field Programmable Gate Arrays) – makes the ideal candidate for being integrated into Cloud systems due to their high flexibility and scalability. This work describes the design and implementation of a remote reconfigurable FPGA-based SoC (System on Chip) with a Service-oriented configuration interface over the Internet, and underlines several solutions in order to meet the specific challenges raised by this type of integration. The FPGA board (that can be at a remote location) is configured by a Web Service linked to a JSP (JavaServer Pages) Web-page where the user can provide a bitstream configuration file for the Programmable Logic (PL). On the SoC/FPGA board, dedicated software has been developed to run on the embedded Processing System (PS), managing the downloaded bitstreams and configuring the PL.**

I. Introduction

Reconfigurable hardware has become very popular in the past decade, being integrated today into a variety of applications. These devices offer adaptability and scalability, providing flexible solutions for developing reusable systems due to their potential in minimizing the requirements for dedicated hardware and optimizing power consumption.

As Cloud Computing is developing rapidly, the need for making available the “Hardware as a Service” (HaaS) [1] is growing, and the high adaptability and scalability of the reconfigurable computing implementations favor such an approach, due to the fact that it can meet the cloud requirements for continuous scaling to variations in load and demand. The services become more configurable, less dependent on physical resources, as intermediation, middleware, is more flexible itself – towards an optimal deployment in terms of performance / cost.

By using Web Services to access reconfigurable resources, the interaction between reconfigurable computing and applications distributed in the cyber-space could bring important benefits like reducing complexity of the redesign and a higher flexibility in IP (Intellectual Property) Core extensions support, thus improving programmability [2].

One of the key features required for reconfigurable FPGA-based systems in order to be integrated on the Internet is the remote reconfiguration or field upgradeability. Remote access allows configuring, upgrading or modifying an embedded system from a remote location [3]; in the case of FPGA systems this is done by sending a bitstream via the Internet, without physical access to the hardware.

This work describes a remote reconfigurable FPGA-based SoC with a Service-oriented configuration interface over the Internet. The hardware design was deployed on a Xilinx Zynq™-7000 AP (All Programmable) SoC XC7Z020 [4] on an Avnet ZedBoard 7020 [5]. We “published” online this reconfigurable resource by connecting the development board(s) – via the Ethernet port(s) – in the Intranet of a “Console” PC where we implemented a Glassfish Web-Server and a resident Web Service offering to remote clients in the Internet the capability of bitstream upload for board configuring. For this purpose, we created a Web-page running on the Server and available on the Internet, that provides to the user, at the client terminal, a friendly graphical user interface (GUI) for uploading bitstreams and remote-configuring the FPGA on the ZedBoard.

The paper is organized as follows: section II describes the design and implementation of the system, both the hardware design on the ZedBoard (including the Ethernet communication and the internal configuration routines needed for PL configuration) and also the software design – the Web Server and the FPGA configuration Web-Service running on the PC Console. The functionality of the entire system is presented and explained in section III in a test-oriented and use-case perspective. Finally, section IV states the conclusions and the potential future extensions on the subject.

II. Design and Implementation

A. *System overview* – Fig.1

The system’s core is the XC7Z020 AP SoC belonging to the Xilinx Zynq™-7000 family.

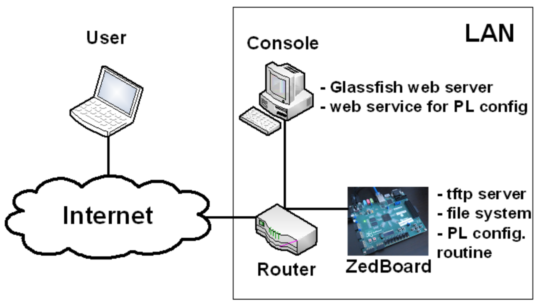


Fig. 1. Overview of the system and communication network.

It integrates a dual-core ARM® Cortex™-A9 MPCore™ based processing system (PS) and Xilinx programmable logic (PL) in a single device. This unique integration provides a high level of performance, enhancing the possibilities for complete or partial re-configuring, with PS not only monitoring the PL but also having the potential of local control and even local decision.

This design takes advantage of these embedded computing capabilities that the board offers, together with the on-board peripherals: bitstreams storage DDR3 memory, UART (for debug and communication with the Console) and Gigabit Ethernet (downloading bitstreams and receiving configuration commands).

The board is connected on a LAN, in the Intranet of the PC Console that is hosting the above-mentioned Glassfish Web Server that runs the Web-page enabling remote clients to upload bitstreams and to send configuration commands via the Internet, as well as a Web Service that does the actual configuration of the board. Besides the advantage of unified ports, this Web Service – based control enables the API-based programmatic control not only by human remote users but also by remote machines. Our approach is a modern one, considering that also reputed companies like National Instruments went to a unified Application Deploy as a Web Service, performing as many as possible data exchanges as Web Services and orienting implementation to generic Representational State Transfers (REST) [6].

B. *Hardware architecture*

The hardware architecture relies on the XC7Z020 SoC processor, running at 100MHz. The PS is running on one of the ARM Cortex-A9 cores, and it executes the configuration software. A DDR controller is used to interface the on-board 512MB DDR3 memory chip. The PS is configured with the following I/O peripherals enabled: ENET0, SD0 (for a Secure Digital card), UART1, USB0 and GPIO (General-Purpose I/O). The PS internal clock generator also provides a 100MHz clock to the PL. The PS architecture is based on the AXI (Advanced eXtensible Interface) protocol for interconnecting IP Cores and system components.

Fig. 2 shows the diagram of the PS with the connected peripherals.

A 32MB file system (FS) residing in the DDR3 memory has been implemented for storing the configuration bitstreams downloaded via the network. This was accomplished using the LibXil MFS (Memory FS) component, which provides the capability of managing program memory in the form of file handlers. Read/write access to files stored in this FS was accomplished through C language function calls specific to this file system. Using this FS brings the potential asset of having a local bitstream version-management and regression. For our development, this is a capability that could prove useful in organizing a bitstream "cache", especially in the light of future extensions like dynamic partial re-configuration (PR), where a fast access to the partial configuration files is needed.

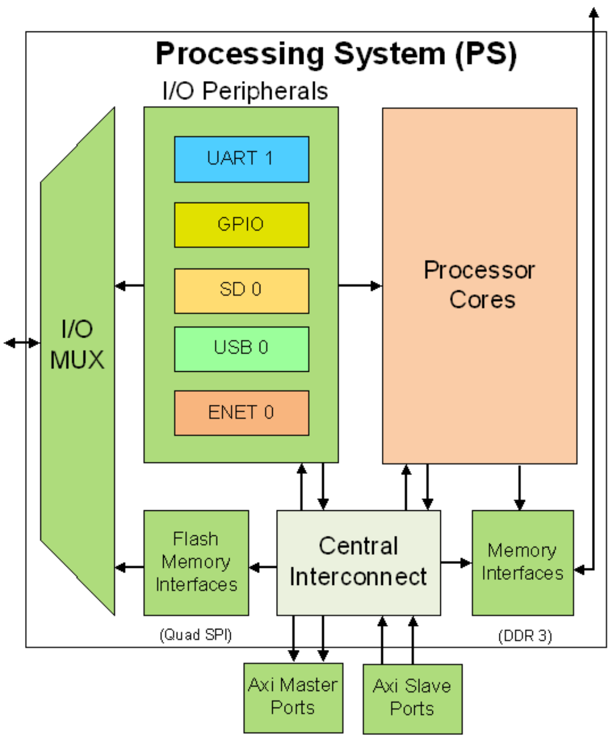


Fig. 2. Processing system architecture.

Only "re-validation" of recently-used (and stored in the "cache") bitstreams would be enough, instead of repeating the full configuration procedures.

We implemented the network communication, based on the on-board Gigabit Ethernet controller, by using the TCP/IP LwIP (Lightweight IP) Stack, an open source networking stack designed for embedded systems. It is shipped together with Xilinx SDK for adding network capability to an embedded system and it is provided under a BSD style license. For this implementation, we used LwIP version 1.4.0. This way, we implemented a TFTP (Trivial File Transfer Protocol) Server and a TCP Echo Server, based on the Xilinx LwIP Application Note (XAPP1026) [7].

TFTP is a UDP-based protocol used for file transfer. The TFTP Server uses callback functions to handle connections and I/O data. The TFTP Server stores uploaded files in the local MFS. The Echo Server is TCP based, and it is used to communicate configuration and status commands with the Web Service running on the Console.

We chose the solution of having a MFS and TFTP Server because it allows several bitstreams to be stored on the board, validated and ready at any time to be (re-)used for configuring the PL. This also supports future work regarding PR, when locally stored partial bitstreams are used to configure a remote module (RM) of the design during runtime, without affecting the functionality of the rest of the system.

The ARM embedded microprocessor has the capability of reading and writing the FPGA configuration memory through the device configuration (DevC) / Processor Configuration Access Port (PCAP) interface.

This design uses this configuration interface in order to modify FPGA circuit functionality and structure by loading a new bitstream during the operation of the circuit.

The bitstream stored in the on-board DDR3 memory is transferred to the PL via the AXI-PCAP Bridge, using a FIFO buffer for data synchronization between AXI and PCAP interfaces. A driver function sets the correct PCAP mode and initiates the DMA transfer. At this point, the PCAP can be configured for either full or partial reconfiguration, by setting the control register value for PCAP mode accordingly [8].

C. Software architecture

1. Embedded software

The PS embedded software implemented is a standalone C application running on one of the ARM Cortex-A9 cores. This application is composed of three main parts: TFTP Server, Echo Server, and PL configuration methods. It writes status messages on the UART to a terminal running on the local Console for status check and debug purpose.

The application initializes at startup the IP address, network mask, and gateway so that the board can communicate on the local network.

Both the Echo Server and the TFTP Server have been implemented in an event-driven RAW API mode, being single-threaded and having a callback style interface. Callback functions were set on startup (using specific LwIP commands [7]) to be called each time a certain event takes place (connection accept, read or write).

The TFTP Server has a *tftp_receive* function that is set to be called each time a packet is received on the Server port. It processes a new connection request, creating a new port for serving the response, and it calls the corresponding response function (*tftp_read* or *tftp_write*) based on received opcode.

For the Echo Server there are two main callback functions: *echo_receive_data* and *echo_accept_connection*. The function *echo_accept_connection* is called asynchronously for every new connection that is accepted. It sets up the receive callback for the connection, specifying the *echo_receive_data* function, which is called when a new packet is received. *Echo_receive_data* reads and saves the packet content, so that the *echo_application_core* method may act based on it. This method calls specific routines when a new bitstream is about to be loaded to the PL: *write_file_to_ddr* writes the corresponding file from the MFS to a special area in the DDR memory from which it is transferred to the PL by the *config_PL* method.

When *config_PL* is called, the following sequence of operations takes place:

1. The Xilinx Device Configuration Interface (XDcfg) is initialized by enabling the PCAP interface and setting the control register for the corresponding PCAP mode (full PL configuration):
2. DMA and PCAP Done interrupts are cleared
3. The bitstream is transferred from DDR to PL fabric
4. PCAP and AXI Done interrupts are polled, so the function call returns only after both transfers are complete.

PL configuration status, and also other messages related to the embedded application status and functionality, are displayed via the UART and can be checked on a serial terminal running on the Console (Fig. 3).

We embedded these communication protocols for a Web-based exposure of Controls and Capabilities susceptible to become accessible by programs and machines and not only by humans – this is one of our work objectives. This is an important step ahead compared to simple Remote-Desktop control (a pseudo-"introduction of distance"). Another notable feature beyond the simple Remote-Desktop is this monitoring and debugging capability bringing the Console closer to the OAM (Operation, Administration and Maintenance) terminal functionality.

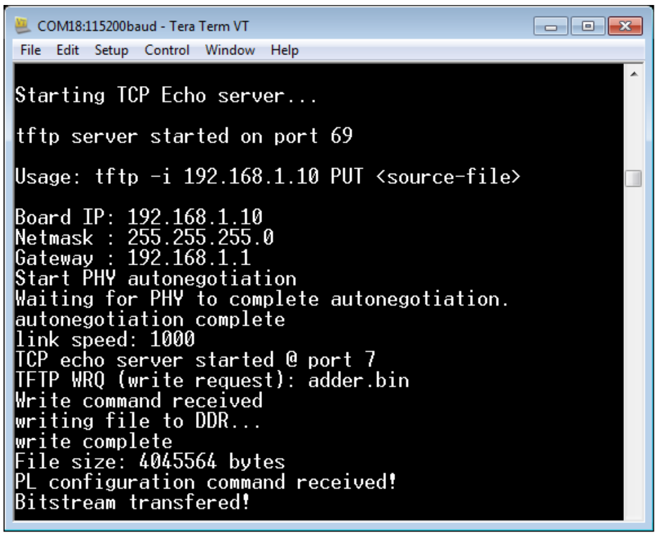
2. Web applications

Service-oriented architectures and Web Services are the latest step in our development of the middleware, solving the problem of inter-operability (e.g. via general purpose ports, compared to specific ports that aren't open by default, because of security issues) [9].

Heterogeneous technologies – in terms of lower OSI (Open System Interconnect) layers – can be easier integrated; the actual trends towards "Everything as a Service" have an important application domain in the „Clouds" of distributed resources.

We aimed to making programmable logic available to the user by means of the services' standard interface, without the user – "service integrator", not mandatory a FPGA expert! – having to bother using specific tools for bitstream download. The Service-oriented approach (Fig.4) that we propose offers high scalability and adaptability – it can accommodate a network of different boards, all being configurable using the same Service and Web interface.

We implemented the *ConfigWS* Web Service using Java technology and NetBeans IDE (Integrated Development Environment), running on a Glassfish 3.0.1 Server instance.


COM18:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
Starting TCP Echo server...

tftp server started on port 69

Usage: tftp -i 192.168.1.10 PUT <source-file>

Board IP: 192.168.1.10
Netmask : 255.255.255.0
Gateway : 192.168.1.1
Start PHY autonegotiation
Waiting for PHY to complete autonegotiation.
autonegotiation complete
link speed: 1000
TCP echo server started @ port 7
TFTP WRQ (write request): adder.bin
Write command received
writing file to DDR...
write complete
File size: 4045564 bytes
PL configuration command received!
Bitstream transfered!
```

Fig. 3 Serial terminal running on the Console and displaying the embedded application status.

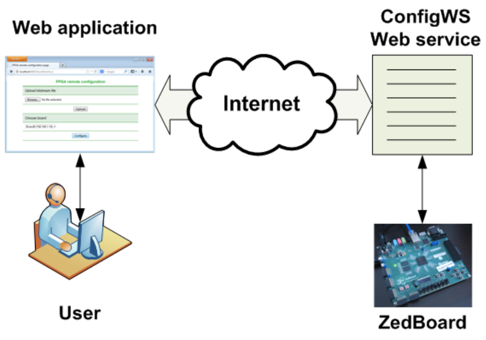


Fig. 4. Web applications communication diagram.

Three methods were implemented for communicating with the board:

uploadBitstream – receives the filename and board IP as parameters and uploads the specified configuration file to the TFTP Server running on the board's PS.

configPL – sends a TCP message containing a configuration command to the board specified by an IP address transmitted to the method as a parameter.

statusCheck – interrogates the board for its status by sending a specific TCP message, the message containing the board's reply is then displayed on the Web-page.

The above-mentioned Web Services are specifically managing data and processes of the application. Web Services do not provide, by themselves, a graphic interface to the user [10]. This Java-based exposure of Controls and Capabilities enables the replacement of the Man-Machine interface with a M2M (Machine-to-Machine) interface – giving a Cloud Management "handler".

We designed the Web application as a Java servlet that receives and processes information from a JSP page using Javascript. The JSP page offers the user the possibility of uploading a locally stored bitstream file that will be used for PL configuration. Also, on the Web-page, a list of connected configurable boards is displayed, allowing the user to choose which one to be configured using the selected bitstream. This allows for a network of boards to co-exist and be configured by the same Web Service, being uniquely identifiable by IP address, which is passed alongside the configuration file as a parameter to the Web Service.

D. System security

The security of the system is an important subject, especially since it is designated to be part of a cloud computing architecture. This issue needs to be split in two: securing the interface and data between the client and the server, and the security of the remote ZedBoard.

For the first part, security can be applied to the web services, either at the transport-level and/or the message level. Message security uses XML Encryption and XML Digital Signatures to secure the SOAP messages. Transport level security authenticates the identity of the user by implementing a user authentication method for the web application (either basic authentication or client-certificate authentication over HTTP/SSL). Our implementation supports implementing any of the two levels of security for protecting the bitstream configuration files and other information while in transfer to the web server.

On the other hand, securing the ZedBoard located "in the cloud" is a sensitive subject, given the fact that today cloud computing clients are requesting for strong security guarantees to be mentioned in the SLA (Service Level Agreement). In the case of this implementation, a high level of security can be achieved by using the built-in features of the Zynq SoC: bitstream encryption (AES – Advanced Encryption Standard) and the unique Xilinx Anti-Tamper (AT) technology that protects the chip against physical tampering (side channel, fault insertion or readback attacks). The latter is of special interest since physical attacks are one of the main concerns in the case of cloud computing, where the system that handles the client's sensitive data is located remotely, exposed to physical interference (for example, by a malicious system administrator).

III. System Functionality and Applications

By accessing the FPGA remote configuration Web-page running on the Web Server over the internet (Fig. 5), the user can choose a locally stored bitstream file to be uploaded and used for PL configuration. Also, the user can choose which board to be configured using the selected bitstream.

The upload action is executed by the *uploadBitstream* method implemented in the *ConfigWS* Web Service. After a bitstream has been uploaded, by clicking the Configure button, the *configPL* method sends the configuration command to the board and the PS configures the PL using the uploaded configuration file.

The user can also check the status of the board by clicking the "Check Status" button. This calls the *statusCheck* Service method, that sends a status check request message to the board and displays on the page the reply received containing information regarding board's configuration status (Fig. 6).

In case of a communication error during any of the three operations, the user is informed by a specific message displayed on the page.

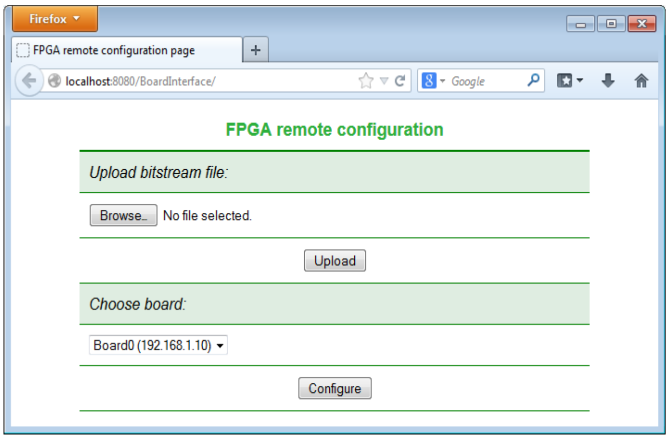
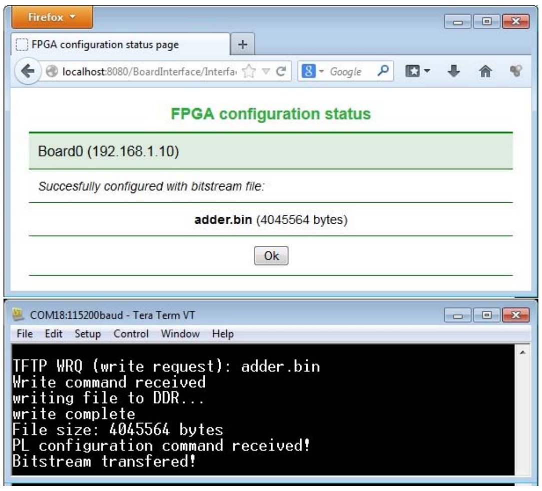


Fig. 5. FPGA remote configuration Web-page running on Glassfish Server.


Firefox
FPGA configuration status page
localhost:8080/BoardInterface/Interfa
Google
COM18:115200baud - Tera Term VT
File Edit Setup Control Window Help

FPGA configuration status

Board0 (192.168.1.10)

Succesfully configured with bitstream file:

adder.bin (4045564 bytes)

Ok

```
TFTP WRQ (write request): adder.bin
Write command received
writing file to DDR...
write complete
File size: 4045564 bytes
PL configuration command received!
Bitstream transfered!
```

Fig. 6. FPGA status after a configuration command has been executed. Successful configuration message displayed on the webpage (up) and on the serial terminal running on the console PC (down).

The PL is configured using a PROM file (.bin) containing configuration data for the FPGA device. The user has the possibility of uploading configuration files in both .bit (BitGen-generated configuration bitstream) and .bin formats. In case of a .bit file, the Web Service converts it to a PROM format using the Xilinx PROMgen tool, before uploading it to the board. When a configuration message is received via the Ethernet, the PS embedded application calls the *write_file_to_ddr* and *config_PL* methods in order to configure the PL with the configuration file. After the configuration process is complete, a TCP Echo reply message is sent containing the outcome of the configuration attempt.

The remote-reconfigurable FPGA/SoC system has a great potential in Cloud Computing systems where such a platform could be integrated in order to solve many practical problems related to the scaling of hardware resources [11]. These potential applications imply an efficient management of the board's configuration and application specific data flows. While the on-board Gigabit Ethernet connection used for transferring configuration files and commands can be shared with PL application specific I/O data, a recommended solution would be adding another Ethernet interface to the board by using a FPGA Mezzanine Card (FMC) (e.g. a 1000 Base-T Ethernet Card).

Such a board configuration would have the advantage of separating the Ethernet port used for PL configuration and PS communication to the Console on the LAN, and the other Ethernet port(s) dedicated to the PL application, and configured by the user. This also improves the security of the entire system, in the perspective its integration in Cloud IaaS (Infrastructure-as-a-Service) a higher level compared to HaaS.

By having reconfigurable hardware resources available through Web Services these resources are ready to be easily integrated into Cloud systems, with potential applications in Cloud security due to the FPGA's computation closed environment (built-in bitstream protection and isolated memory spaces) [12].

Also, the reconfigurable potential of the SoC can be enhanced by implementing the PL design with a Network-on-Chip (NoC) architecture, a flexible communication structure that can be used for implementing application-specific reconfigurable systems, being capable of connecting tens of IP cores in a flexible and easily scalable manner. Due to its internal architecture, a NoC can integrate easily reconfigurable modules (RM), since the network adapter acts like an interface between the RM and the on-chip network, ensuring the de-coupling of computational and communication domains. Thus, all the reconfigurable IP cores sharing the same reconfigurable region on the chip will have the same standardized interface with regard to I/O signals and placement pins; this integration is made easier by the NoC network adapter [13].

A major area where this platform could have many applications is in distance learning – remote virtual laboratories. Being credited as a way to offer a hands-on experience in locations lacking possibilities for a real-life expertise, or as a cost-effective approach making available technologies and devices for multi-user practice while integrating them at a single location, distance learning has gained a lot of popularity in recent years, especially in the field of engineering, where the role of laboratory work is very important [14]. Thus, this system constitutes a key building block for a remote Hardware Design/Embedded Systems Laboratory that could offer the students a practical and complementary experience for their respective courses, enhancing their understanding of digital systems operation.

Researchers have highlighted a series of benefits of such laboratories including reduction of costs, improved learning, higher flexibility and inter-institutional research sharing [15].

There are also possibilities to "publish" high-end experimental resources located in post-graduate educational facilities for other institutions (e.g. under-graduate schools or colleges, SME – Small or Medium Enterprises etc) that lack funds and/or expertise for such laboratory integration.

Enhancing usability and accelerating the return of investment is also possible in an international cooperation of academic institutions (e.g. more efficient usage rate in complementary time intervals) – "multiplexing the educational resources". In the perspective of such a distributed environment, the possibility of adding extra communication between the webpage and the board has been considered at the design stage of the system.

For educational purposes, but not only, the web interface could allow extra data to be sent to the PS and from here into the PL using specific registers. In a remote laboratory scenario, students could thus send data to the board that would be processed by the design that they programmed the FPGA with, while the results could be displayed on the board LCD or LEDs, and made visible by a webcam online stream for a full laboratory experience. Such an approach has the potential of bridging the gap between the textbook simulation-only class environment for learning Hardware Description Languages (HDL) and real-life applications.

The implemented remote-reconfigurable FPGA/SoC system was validated in post-graduate education at “Transilvania” University of Brasov-Romania, taking advantage of our implementation using Web Services that is aiming to a direct interfacing of the boards with the students on the Web, without the need of remote-desktop connections to PCs. Thus, our solution has a great potential in cost-reduction, as a single PC Console running the Web applications can manage an entire network of such FPGA-based boards. A Web-Cam live stream can easily be sent via the Internet to the Board configuration page for a complete remote learning experience. Also, the above-mentioned inter-institutional access sharing of such resources is easier to manage and implement with this Service-oriented approach.

IV. Conclusions and Future work

This paper describes a FPGA-based SoC Service-oriented integration as a remote reconfigurable platform ready for deployment in Cloud Computing systems.

The design takes advantage of the embedded computing capability that the XC7Z020 AP SoC offers by using the PS and PL integration into a single device to exploit possibilities of complete and partial PL re-configuration and monitoring by embedded software running on the PS.

The system was especially designed in order to easily integrate future developments like partial re-configuration; a file system was implemented on the board to be used for bitstream storage, but also for managing partial bitstreams that are dynamically used for configuring certain modules of the design during runtime, without affecting the functionality of the rest of the system.

The user has access to the board configuration options by accessing a Web-page linked to a Web Service.

This Service-oriented architecture was chosen for easing inter-operability and providing scalability and flexibility to the system: the Web Service and configuration interface can be used to configure a large number of boards connected via a LAN and identifiable by their IP address.

The high scalability makes this design ready to be easily integrated into Cloud Computing systems, with many possible applications that take advantage of the numerous assets that the synergy between reconfigurable hardware and distributed computing brings.

The Cloud applications are enabled by the complete range of communication solutions that we developed for the integration of our platform in a distributed environment.

Considering the generic concept of Reconfigurable Input-Output (RIO) developed by National Instruments, we will integrate our SoC/FPGA Platform in LabVIEW-driven configurations via the specific NI middleware and, last but not least, via the new NI LabVIEW FPGA Compile Cloud Services [6]. Compile Cloud Services open the broad spectrum of distributed logic with central (re-)configuration. (without the need of resource-intensive on-site compilation).

To endorse our future development, we considered that the newest National Instruments “myRIO” boards are using the same Xilinx Zynq SoC (the 7010 family) [16].

In the educational domain, we also plan to integrate our FPGA remote laboratories for e-Learning hardware courses in the area of computer engineering in the existing Virtual Electro-Lab http://vlab.unitbv.ro/velab developed by “Transilvania” University since 2004.

Acknowledgment

The authors would like to thank Digital Optics Company – Brasov, Romania, that offered the infrastructure used for conducting the research described in this paper.

This paper is supported by the Sectoral Operational Programme Human Resources Development (SOP HRD), ID134378 financed from the European Social Fund and by the Romanian Government.

References

[1] A. Stanik, M. Hovestadt, O. Kao - Hardware as a Service (HaaS): Physical and virtual hardware on demand, Proceedings of the IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom), Taipei, 3-6 Dec.2012, pp.830-836

[2] C. Wang, X. Li, P. Chen, J. Zhang, X. Feng, X. Zhou “Regarding Processors & Reconfigurable IP Cores as Services”. Proceedings of IEEE 9th International Conference on Services Computing, 2012, pp.668-669

[3] P. Ogrutan, L. E. Aciu, Microcontroller Based System for Accelerated Reliability Tests for Electronic Equipment, International Conference AFASES 2013 Brasov, 23-25 May 2013, pp. 387-392

[4] www.xilinx.com/zynq Zynq-7000 All Programmable SoC (accessed 28 Nov 2013)

[5] http://www.zedboard.org ZedBoard Development System (accessed 10 Dec 2013)

[6] www.ni.com – LabVIEW Manual, National Instruments (accessed 30 Dec 2013)

[7] A.Sarangi, S.MacMahon - Xilinx LightWeight IP - XAPP1026 Application Note - v3.2 - October 28, 2012, www.xilinx.com/ (accessed 3 Jan 2014)

[8] C. Kohn - Partial Reconfiguration of a Hardware Accelerator on Zynq-7000 All Programmable SoC Devices- XAPP1159 Application Note – v1.0 – January 21, 2013, www.xilinx.com/ (accessed 20 Dec 2013)

[9] G. Alonso, C. Fabio, K. Harumi, M. Vijay, Web services. Springer Berlin Heidelberg, 2004. ISBN 3-540-44008-9

[10] D. Chappell, T. Jewell, Java Web Services, O’Reilly, 2002, ISBN 978-0-596-00269-5

[11] A. Madhavapeddy, S. Singh, “Reconfigurable Data Processing for Clouds”. Proceedings of the IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2011, pp.141-145

[12] K. Eguro, R. Venkatesan, “FPGAs for Trusted Cloud Computing”. Proceedings of 22nd International Conference on Field Programmable Logic and Applications, 2012, pp.63-70

[13] L. Moller, G. Ismael, N. Calazans, F. Moraes. "Reconfigurable systems enabled by a Network-on-Chip." Proceedings of FPL'06 IEEE International Conference on Field Programmable Logic and Applications – Madrid, 28-30 Aug. 2006, pp. 857-860

[14] El Medany, Wael M. "FPGA remote laboratory for hardware e-learning courses." Computational Technologies in Electrical and Electronics Engineering, Proceedings of the 8th IEEE International Conference on-, SIBIRCON 2008, pp.106-109

[15] F. Morgan,S. Cawley, “Enhancing Learning of Digital Systems using a Remote FPGA Lab”, Proceedings of the 6th International Workshop on Reconfigurable Communication-centric Systems-on-Chip, 2011, pp.1-8

[16] www.ni.com/white-paper/14604/en/ – myRIO Hardware at a Glance, National Instruments, (accessed 10 Jan 2014)