FPGA Remote Update for Nuclear Environments

Ana Fernandes, Rita C. Pereira, Jorge Sousa, Paulo F. Carvalho, Miguel Correia, António P. Rodrigues, Bernardo B. Carvalho, Carlos M. B. A. Correia, and Bruno Gonçalves

***Abstract*— In recent years there has been a growing interest to use reconfigurable modules, based on field-programmable gate array (FPGA) devices, in nuclear environments. One of the requirements for these types of modules, when operating in complex future nuclear power experiments, is their remote update capability. The operational needs of pulsed fusion reactors will lead to a large production of very high energy neutrons (MeV range). The current procurement policies for nuclear installations do not allow exposure of electronics to radiation, except following very strict rules. However, considering the "as low as (is) reasonably achievable" (ALARA) principle with respect to human exposure to radiation, the access to cubicles might be restricted, requiring the remote update of FPGA codes. FPGAs are volatile devices, and their programming code is usually stored in dedicated flash memories for proper configuration during module power-on. This paper presents an alternative method for FPGA remote update, capable to store new FPGA codes in inboard Serial Peripheral Interface (SPI) flash memories. The new method, based on the Xilinx Quick Boot application note and adapted to PCIe protocol, was developed with the KC705 Evaluation Kit from Xilinx and successfully tested in the in-house Advanced Mezzanine Card (AMC) prototype, installed on the ATCA-PTSW-AMC4 carrier module from the ITER Fast Plant System Controller catalogue.**

***Index Terms*— Firmware upgrade, field-programmable gate array (FPGA), nuclear fusion diagnostics, Peripheral Component Interconnect express (PCI-e).**

I. Introduction

RECONFIGURABLE hardware modules based on field-programmable gate array (FPGA) devices are currently used by the most demanding applications [1]. In recent years there has been a growing interest to use FPGA-based modules in nuclear power plants, demonstrating that full digital platforms can efficiently monitor and control such environments [1]. FPGA-based modules present a combination of software and hardware features, providing flexibility and capability similar to software however with lower complexity, simpler system structure and improved performance characteristic of hardware [1]. Moreover, FPGAs are able to provide truly parallel data processing, synchronism, flexibility in its configuration and unique performance at high processing frequencies. Considering the steady state operations with high energy content foreseen for future generation of fusion devices, such as ITER and DEMO, real-time tools and mechanisms will be mandatory for data handling and machine safety control. Therefore, FPGA-based modules will be essential for real-time data transfer and processing in those environments.

FPGAs are volatile devices, configured during system power-on through dedicated inboard flash memories, where the code is stored. The most common method for code transfer to local memories requires: i) a JTAG module (IEEE 1532-2002 standard); and ii) dedicated software tools (e.g., Impact from XILINX) running in a nearby personal computer, capable to transfer FPGA codes to dedicated storage devices through the JTAG cable [2], [3]. However, this method may be unfeasible especially in complex machines with very restricted access. According to the ALARA principle with respect to human exposure to radiation, future fusion devices might not allow free access to the instrumentation cubicles, demanding the remote update of FPGA codes. This work describes an alternative method capable to store new FPGA codes in Serial Peripheral Interface (SPI) flash memories, through the Peripheral Component Interconnect express (PCIe) network established on Advanced Telecommunication Computing Architecture (ATCA) backplanes, linking data acquisition endpoints and the data switch blades. Moreover the method can be extended to other XILINX FPGA-based modules with PCIe communication protocol established with the host PC. The implementation details and test results are described in the next sections.

II. PCIE Communication Protocol in ATCA-Based Systems

IPFN has undertaken a research program, which envisages the specification phase and development of high-availability ATCA-based systems for ITER [4]. The high throughput allowed by the ATCA backplane architecture, based on point-to-point protocol agnostic connections, enables high performance communications between modules over serial multi-Gbit/s links. Different types of communication protocols are supported by ATCA backplane (e.g., PCIe, Gigabit Ethernet (GbE), Serial RapidIO (SRIO), Infini-Band). However, the PCIe characteristics such as: i) very high throughput with very low latency between modules and the central controller; ii) multivendor support both on electronics and development tools; iii) large code legacy, including PCI compatibility; and iv) hot-plug functionality, were the main reasons to select the PCIe protocol for ATCA backplane communication [5]. A typical PCIe system tree is composed by a Root

Manuscript received June 18, 2015; revised March 24, 2016; accepted April 24, 2016. Date of publication April 27, 2016; date of current version June 21, 2016. This work was supported by the Fundação para a Ciência e Tecnologia (FCT) through Project UID/FIS/50010/2013. The views and opinions expressed herein are the sole responsibility of the authors.

A. Fernandes, R. C. Pereira, J. Sousa, P. F. Carvalho, M. Correia, A. P. Rodrigues, B. B. Carvalho, and B. Gonçalves are with the Instituto de Plasmas e Fusão Nuclear, Instituto Superior Técnico, Universidade de Lisboa, 1049-001 Lisboa, Portugal (e-mail:anaf@ipfn.ist.utl.pt).

C. M. B. A. Correia is with LIBFIS-UC, Departamento de Física, Universidade de Coimbra, 3004-516 Coimbra, Portugal.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNS.2016.2559478

Complex (RC) which connects the host processor (ATCA controller) and memory subsystems (ATCA modules) to the PCIe hierarchy. The FPGA code of each ATCA module includes an endpoint core from XILINX, compliant and electrically compatible with the PCIe Base Specification [6]. The PCIe endpoint core supports the exchange of Transaction Layer Packets (TLPs) such as memory read/write, input/output (I/O), and configuration message transactions, between endpoints (ATCA modules) and PCIe switch upstream ports or host (e.g., the ATCA controller) at speeds up to 5 Gb/s (Gen2) [6]. The endpoint physical links are implemented through the Multi-Gigabit Transceivers (MGT) available in FPGA. Data transfer and synchronization tasks between ATCA modules and the ATCA host are established through dedicated shared memory space available on host. Configuration registers available in that shared memory space must be properly defined between host and the FPGA code, guaranteeing the correct system operation. The memory address mapping starts in a Base Address Register (BAR) set by the BIOS or by the Operating System (OS) in the PCIe configuration space during power up [7].

III. Remote FPGA Code Update

Different programming methods can be used to upload new FPGA codes to dedicated inboard flash memories [2]. One of the most common method is the JTAG IEEE 1532–2002 platform, requiring proprietary hardware and software on site.

Another option is using the Shelf Manager, the Intelligent Platform Management Controller (IPMC) and the I2C Intelligent Platform Management Bus (IPMB) available in ATCA shelfs. The main drawback of this solution is the low throughput of the IPMB (100 kbit/s), requiring long programming times [2].

Considering the IPFN modules, the best solution for code transfer is using the PCIe communication protocol between ATCA modules and its host. Therefore, it was implemented an FPGA code update method for flash programming through PCIe protocol, allowing to avoid any proprietary hardware (e.g., JTAG) or software (e.g., Xilinx software) tool in the remote site.

The method is based on the Quick Boot application from Xilinx [8], detailed in the following section.

A. Quick Boot Method

The Quick Boot method for FPGA remote update describes a safety solution for updating the FPGA code to local storage devices, though based on the JTAG programing tool [8]. This solution enables a reliable field update of memory flash through a complementary combination of a fast, robust configuration method and efficient Hardware Description Language (HDL)-based, in-system programming. The method can be applied to SPI and Byte-wide Peripheral Interface (BPI) flash memories, widely used by recent FPGA-based modules. It proposes three reserved areas within the main flash memory array to store: i) a special header with a critical switch word and a warm boot jump sequence; ii) the initial FPGA code; and iii) the update FPGA code. As depicted in Fig. 1, the critical switch word is the key to the configuration method. When the critical switch is ON, the FPGA loads the updated FPGA code, otherwise the FPGA loads the initial code. During an update process the method allows to reprogram the reserved update image area. The initial image area must never be modified ensuring the proper operation of the module.

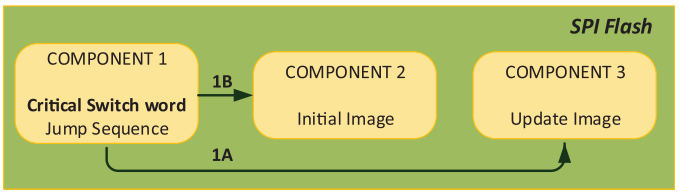

Fig. 1. Flash memory array based on XILINX Quick Boot application note [8]. It is composed by 3 reserved areas: 1) special header with the critical switch word and a warm boot jump sequence; ii) the initial FPGA code image area; iii) the update FPGA code image area. If the critical switch word is ON the configuration sequence jump to the update image area and load the update code (1A), otherwise the initial FPGA code is loaded (1B).

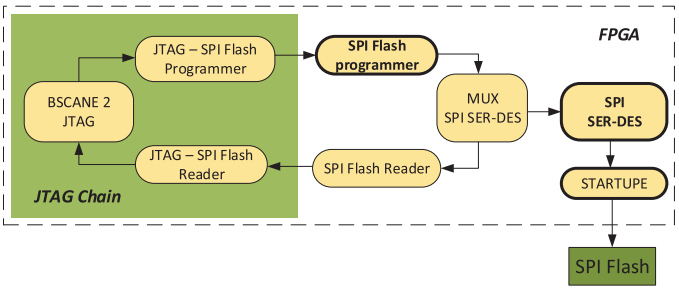

Fig. 2. QuickBoot reference design for FPGA code update to SPI flashes through JTAG cable. The main components are the SPI Flash Programmer with the programming algorithm and the SPI SER-DES for interfacing with the SPI flash bus. The JTAG-SPI Flash programmer and flash reader are used for data transfer through JTAG standard.

The Quick Boot application note includes: i) a Perl language [9] Script (PERLS) for generating flash memory image files; ii) HDL flash programming codes capable to in-system update image files to SPI or BPI flash memories; and iii) additional HDL codes able to bridge the FPGA JTAG port with the flash programmer codes, as depicted in Fig. 2.

The SPI flash programming algorithm is able to perform the following tasks.

1) Erase the subsector or block containing the critical switch word (turns OFF the switch word).
2) Erase the update image area.
3) Upload the new code into the update image area.
4) Verify that the update image area was correctly reprogrammed.
5) Reprogram the critical switch word if the previous task succeeds (turns ON the switch word).

B. Remote Update Through PCIE

The work developed was based on the Quick Boot application method, adapted for programming SPI Flash memories through PCIe communication protocol, instead of using JTAG tools. The updated FPGA code is sent to the remote host through any available communication protocol (e.g., Ethernet).

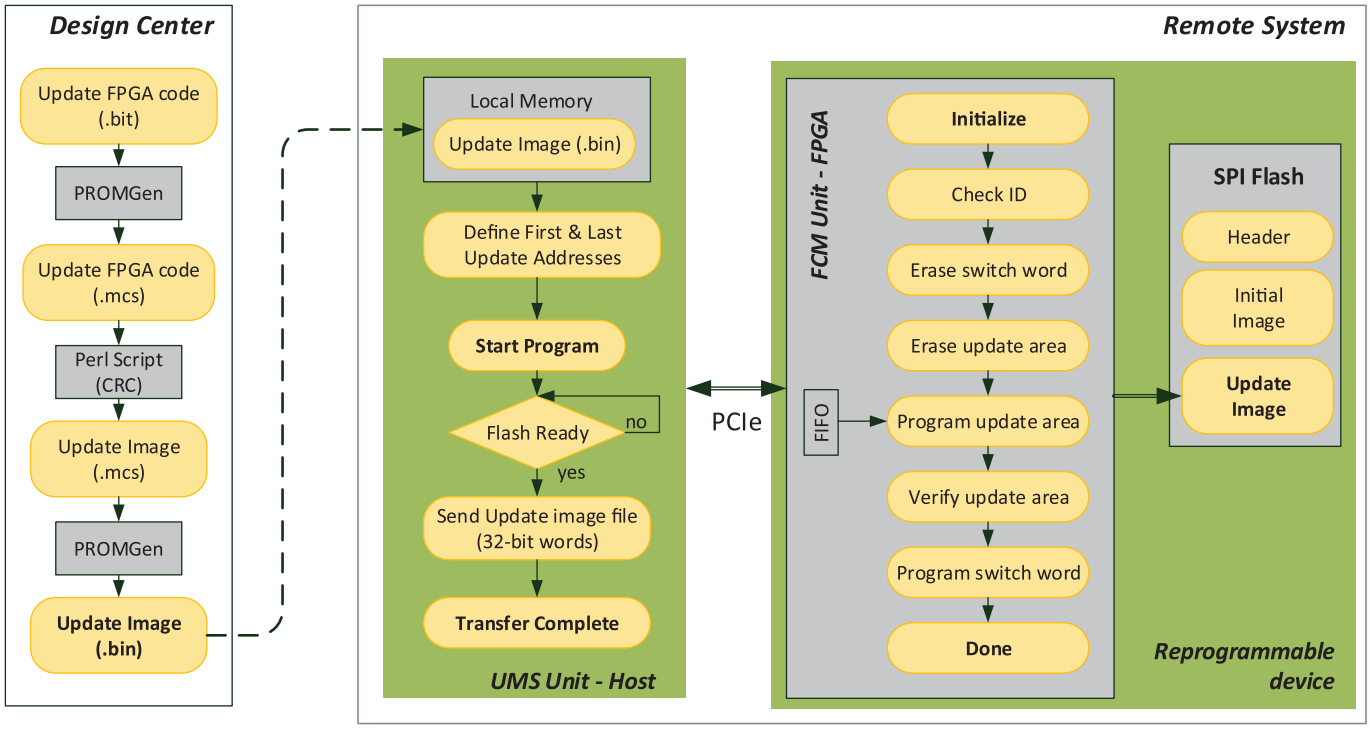

Fig. 3. SPI flash update flowchart. When a new FPGA code is available, it will be converted to an update image file and sent to the remote system. In the remote site, the host UMS unit and the FCM unit at FPGA are responsible for the updating process to the SPI flash.

In the remote site, a dedicated Update Management Software (UMS) unit, running on the host PC, communicates with the FPGA-based module through PCIe links. The UMS is responsible for indirect programing the SPI flash through the FPGA Code Management (FCM) unit running at FPGA, as depicted in Fig. 3, detailed during the following sections. Therefore, the UMS unit main tasks are: i) enabling the update; ii) sending the update address values, given by PERLS; and iii) sending bursts of data with the new FPGA code. To include the two images in the same SPI flash, the FPGA code size may need to be reduced, by turning ON the compression toll in the FPGA code compiler. As result, different FPGA code sizes are produced for each new compilation, leading to different update address values.

To ensure the proper operation of the method each new FPGA code must include the FCM unit. It is composed by HDL modules available from Quick Boot HDL code design (SPI flash programmer and SPI SER-DES, Fig. 2), adapted to the PCIe communication protocol.

As depicted in Fig. 4, the FCM main components are: i) the PCIe engine for protocol communication between FPGA and host PC; ii) the SPI state machine with the programing algorithm, capable to send commands/data to the SPI flash and receive data from SPI; and iii) the SPI SER-DES for interface with the SPI flash bus.

Therefore, when a new update starts the FPGA receives the SPI update addresses, followed by the start command and bursts of data containing the new FPGA code, to be temporarily stored in a buffer (a first-in first-out (FIFO) core) while the SPI flash is not ready. When the data transfer finishes the flag "Done"' turns ON.

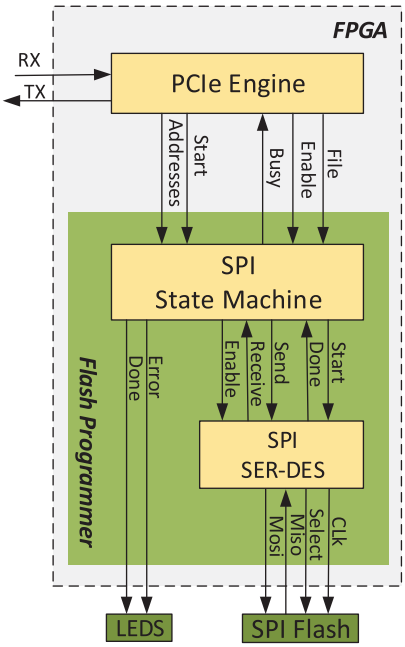

Fig. 4. FCM unit flowchart. The main components are: i) the PCIe engine with the PCIe endpoint core and its interface for protocol communication between FPGA and host PC; ii) the SPI flash state machine; and iii) the SPI flash serializer/deserializer for interfacing with the SPI flash bus.

C. Implementation Details

The first step required to prepare a module for the remote update capability is to program the SPI flash in-site, with an initial .mcs file produced with the PERLS, through the JTAG platform. This initial file is composed by three reserved areas

as depicted in Fig. 1, where the initial and the update image areas are filled with the same FPGA code. The PERLS is also responsible for delivering the initial (A0) and final (B0) addresses of the update image area in this initial file. After this task, the module is ready to be remotely updated. The new FPGA codes are first converted to an update.mcs file, produced with the same PERLS, and then to a .bin file to be sent to the remote controller, as depicted in Fig. 3. From the addresses given by PERLS for each new update image file (A1 and B1) it is possible to determine the update image area addresses ($A = A0$; $B = A0 + (B1 - A1)$), requested by the UMS unit.

Finally the UMS unit in the remote host is able to start the update procedures, as described in Section III-B and detailed in Fig. 3 flowchart.

When the update finishes, the FPGA needs to reset for new configuration during startup. This may be done through an hardware reset, or through the FPGA software reset followed by a PCIe hot-plug mechanism, performed by the ATCA hardware management [10].

IV. Results

The FPGA code update method was developed and tested in laboratory with two different FPGA-based setups. Initially it was tested in a KC705 developing kit from XILINX [11] connected to the host through the PCIe finger. The KC705 module includes a Kintex-7 (XC7K325T-2FFG900C) FPGA and a 128 Mb Quad-SPI Flash memory from Numonyx (N25Q128A13BSF40F). The host PC OS is the Linux Fedora Core 21, kernel version 3.18.7.200.

For testing purposes were produced two FPGA codes with different version number, the initial and the update FPGA code, using the ISE 14.7 compiler from XILINX. These two FPGA codes (.bit files) were converted to an initial (.mcs) and an updated (.bin) image files by PERLS. The SPI flash was first programmed with the initial image file through the JTAG cable, connecting the KC705 module to a near laptop where the XILINX Impact tool is installed. After this first step, the JTAG cable can be avoided as well as the Impact tool, and the KC705 module is ready for remote programing through PCIe. Therefore, new update image files are sent to the remote host, and the UMS unit is allowed to start the updating procedure, as described in Section III-C. A dedicated UMS application is used to report the updating status, as depicted in Fig. 5.

If the update succeeds, the firmware version number corresponding to the update code should appear in the associated register (Fig. 5) on the next power-on. When the update does not succeed (e.g., a power-off occurs during the update process) the FPGA is configured with the initial FPGA code, as reported by the firmware version number register.

One of the main purposes to develop the FPGA code remote update is to implement this capability in IPFN ATCA-based modules targeting nuclear environments. As so, the remote update method was also tested in the Advanced Mezzanine Card (AMC-MKX1) prototype module [12], installed on the ATCA-PTSW-AMC4 carrier module from ITER Fast Plant System Controller catalogue [13], [14]. The AMC-MKX1

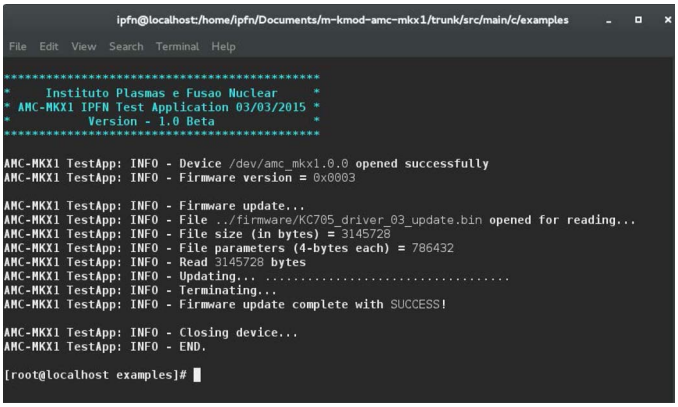

Fig. 5. UMS application report during the update process.

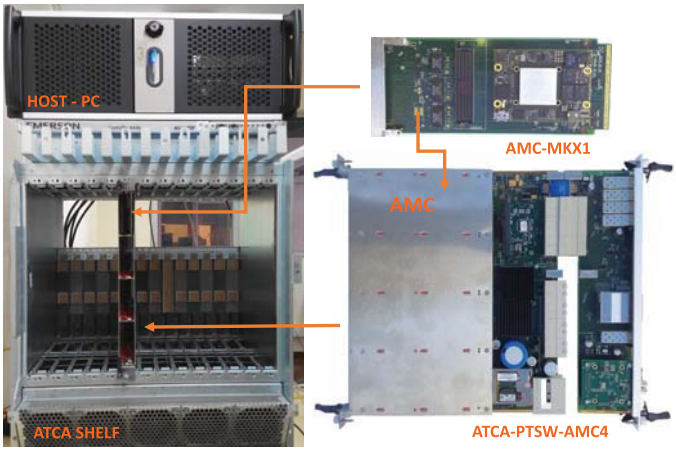

Fig. 6. Fourteen-slot ATCA shelf with an AMC-MKX1 prototype module installed on the ATCA-PTSW-AMC4 carrier module from ITER Fast Plant System Controller catalogue. The ATCA-PTSW-AMC4 is connected to a RTM module with PCIe external cable interface for connection to the host PC. The AMC-MKX1 module includes a Kintex-7 (XC7K325T-2FFG676I) FPGA and a 256 Mb Quad-SPI Flash memory from Numonyx (N25Q256A13EF840F).

module was specially conceived for real-time spectroscopy applications expecting to deliver and process data at a rate of up to 100 GB/s. It includes a Kintex-7 (XC7K325T-2FFG676I) FPGA and a 256 Mb Quad-SPI Flash memory from Numonyx (N25Q256A13EF840F). The ATCA-PTSW-AMC4, mounted in a 14-slot ATCA shelf subrack, is connected to a Rear Transition Module (RTM) module with PCIe external cable interface. The host personal computer is an independent controller unit, housing a $\times$ 16 PCIe 3.0 switch-based cable adapter from One Stop Systems (OSS) vendor for connection through PCIe cable to the RTM module. The DAQ system is depicted in Fig. 6.

The same tests were successfully performed with the AMC-MKX1 remote FPGA code update. Moreover, the total programming time for the FPGA code update is about 100 s, which is a significant decrease in time consumption when compared with JTAG programing times [3].

V. Conclusion

This paper describes an alternative method for FPGA reprograming, capable to cope with the remote update requirement for reconfigurable modules operating in nuclear environments. The method is based on the XILINX Quick Boot application

note, however adapted to the PCIe communication protocol. The new method allows to reprogram SPI flashes remotely, without any dedicated hardware (e.g., JTAG) and software (e.g., XILINX Impact) tool in the remote site. Moreover, it can be considered a safety solution once the initial FPGA code is never deleted during the update process. FPGA is preferably configured with the update code however, if the remote update procedure fails the FPGA is configured with the initial FPGA code. Furthermore, even if the method was tailored for ATCA-based modules, it can be adapted to any system with PCIe network established between reconfigurable modules and the host. The method was developed with the Xilinx KC705 Evaluation Kit and successfully tested in the AMC-MKX1 prototype, installed on the ATCA-PTSW-AMC4 carrier module from the ITER fast plant Instrumentation & Control products catalogue. Considering other modules from the ITER catalogue, the next step will be the method extension to series-6 FPGAs.

REFERENCES

[1] J. Ranta, The Current State of FPGA Technology in the Nuclear DomainVTT Tech. Res. Center, 2012, [Online]. Available: http://www.vtt.fi/inf/pdf/technology/2012/T10.pdf

[2] D. Makowski, G. Jablonski, P. Perek, A. Mielczarek, P. Predki, H. Schlarb, and A. Napieralski, "Firmware upgrade in XTCA systems," *IEEE Trans. Nucl. Sci.*, vol. 60, no. 5, pp. 3639–3646, Oct. 2013.

[3] A. Yang, Using SPI Flash with 7 Series FPGAsXILINX, Tech. Rep., 2041.

[4] B. Gonçalves, J. Sousa, A. J. N. Batista, R. Pereira, M. Correia, A. Neto, B. Carvalho, H. Fernandes, and C. A. F. Varandas, "Atca advanced control and data acquisition systems for fusion experiments," *IEEE Trans. Nucl. Sci.*, vol. 57, no. 4, pp. 2147–2154, Aug. 2010.

[5] A. J. N. Batista, J. Sousa, and C. A. F. Varandas, "ATCA digital controller hardware for vertical stabilization of plasmas in tokamaks," *Rev. Sci. Instrum.*, vol. 77, no. 10F527, 2006.

[6] *7 Series FPGAs Integrated Block for PCI Express v3.0*, XILINX, 2014, [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/pcie_7x/v3_0/pg054–7series-pcie.pdf

[7] R. Budruk, D. Anderson, and T. Shanley, *PCI Express System Architecture*. Reading, MA, USA: MindShare, Addison Wesley, 2003.

[8] R. Kuramoto, QuickBoot Method for FPGA Design Remote UpdateXILINX, xAPP1081, 2014.

[9] *The Perl Programming Language*, 2016, [Online]. Available: https://www.perl.org/

[10] P. Carvalho *et al.*, "PCI express hotplug implementation for ATCA based instrumentation," *Fusion Engineer. Design*, vol. 96–97, pp. 738–741, Oct. 2015.

[11] *KC705 Evaluation Board for the Kintex-7 FPGA User Guide*, XILINX, 2013, [Online]. Available: http://www.xilinx.com/support/documentation/boards_and_kits/kc705/ug810_KC705_Eval_Bd.pdf

[12] R. C. Pereira, N. Cruz, A. Fernandes, J. Sousa, and B. Gonçalves, "Real-time data acquisition and processing system design for the ITER radial neutron camera," *Proc. Sci. 1st EPS Conf. Plasma Diagnostics*, 2015, [Online]. Available: http://pos.sissa.it/archive/conferences/240/158/ECPD2015_158.pdf

[13] P. Makijarvi, ITER Catalog of I&C products – Fast ControllersITER, 2014, [Online]. Available: http://static.iter.org/codac/pcdh7/Folder%202/9-ITER_Catalog_of_I%26C_products_-_Fast_Cont_345X28_v2_1.pdf

[14] M. Correia, J. Sousa, A. P. Rodrigues, A. J. N. Batista, B. Gonçalves, C. A. F. Varandas, and C. M. B. A. Correia, "Atca based hardware for control and data acquisition on nuclear fusion fast control plant systems," *IEEE Trans. Nucl. Sci.*, vol. 58, no. 4, pp. 1701–1705, Aug. 2011.