Dynamic Reconfiguration of Modular I/O IP cores for Avionic Applications

Venkatasubramanian Viswanathan
Nolam Embedded Systems, France
vviswanathan@nolam.com

Rabie Ben Atitallah
University of Valenciennes, France
rabie.benatitallah@univ-valenciennes.fr

Jean-Luc Dekeyser
University of Lille1, France
jean-luc.dekeyser@lifl.fr

Benjamin Nakache
Nolam Embedded Systems, France
bnakache@nolam.com

Maurice Nakache
Nolam Embedded Systems, France
mnakache@nolam.com

***Abstract*—Dynamic reconfiguration using FPGAs has been demonstrated to be highly efficient in different application domains. However little has been explored in the avionic communication domain, where halting the system during runtime for changing the hardware is non-trivial. In this paper we present a runtime reconfigurable architecture using I/O Intellectual Property (IP) cores, used in avionic applications. The system provides a modular I/O interface for communication, using an FPGA Mezzanine Card (FMC). User application can dynamically install and execute the necessary hardware for communication with external avionic sub-systems using FMC. The system thus provides a highly modular and cost effective autonomous solution for an embedded avionic communication system using Dynamic Partial Reconfiguration (DPR). The above described solution has been tested using a Xilinx ML605 prototyping board providing a software interface with a Xilinx Microblaze processor core. The architecture has been evaluated with the JPEG application in terms of area utilization, reconfiguration latency and execution time. The reconfiguration latency can be hidden totally in many cases. While in certain others, the overhead of reconfiguration can be justified by the reduction in the resource utilization.**

***Keywords*-Dynamic Partial Reconfiguration, Avionic IP cores, Modular and Reconfigurable I/Os, FPGA Mezzanine Module, Intensive Signal Processing Applications.**

I. Introduction

Field Programmable Gate Arrays (FPGA) are being used for computational purposes in various fields in the industry. One of the main reasons amongst several others, for using FPGAs within an embedded system is performance. Military applications such as Precision Guided Munitions, 3D path planning and real-time system monitoring requires application processing rates in excess of 10 power 10 operations per second [1]. Although microprocessors execute sequential algorithms on medium volume of data efficiently, FPGAs can exploit the parallelism in the algorithm on huge amount of data. Thus they are highly suitable for time-critical applications. Hence FPGAs are being increasingly employed in safety and mission-critical applications within the aerospace and defense sectors.

Applications are becoming highly sophisticated and requires more amount of hardware resources for execution. This implies more space is required on-board to accommodate this additional hardware. However precious resources in a system such as Crewed Exploration Vehicle (CEV) should be reserved for the human habitants rather than be occupied by equipments [3]. Thus, future space applications will be severely limited in terms of resources [2]. Therefore, there is a demand that these systems manage these resources dynamically in order to maximize the functionality with minimum hardware as possible. Thus, many hardware applications in the avionic domain are directed towards reconfigurable computing. Given the well established fact that hardware execution outperform software, there is an ever growing need for reconfigurable hardware. FPGAs have a proven track record of bridging the gap between software and application specific hardware [4].

Finally the biggest challenge the avionic industry is facing today is the hardware obsolescence issue. The key factor being that, every time the application requirement changes, the I/O interface changes, and thus the underlying hardware protocol also changes. Changing the I/O interface and the underlying hardware meant redesigning the entire board, with lot of Non Recurring Engineering (NRE) costs and huge time to market. The FPGA Mezzanine Card (FMC) standard solves the I/O obsolescence issue with a single 400-pin connector and a standard for carriers and daughter cards. It provides high bandwidth low latency communication links with a potential overall bandwidth of 40Gb/s. With the introduction of FMC by the VMEbus International Trade Association (VITA) group [5], the use of FPGAs in avionic systems is becoming inevitable. Since FPGAs are highly flexible and programmable, the implementation of the underlying communication protocol for the FMC based I/O module can be done with great ease and flexibility. Thus totally eliminating the hardware obsolescence issue the avionic industry has been facing in the recent years.

In this paper we propose a modular, runtime reconfigurable on-chip hardware solution to swap between avionic bus communication protocols for an avionic applications. An FMC based I/O interface is capable of providing several I/O pins for different avionic communication bus standards such as ARINC429, CAN and MIL-STD-1553B, in the same daughter card. However, not all the underlying I/O protocols need to be active at the same time on the hardware. An unused module resident in an FPGA consumes power and precious hardware resources. Therefore, the underlying hardware should be installed and executed as and when requested by the system. Moreover, the time required to change an underlying hardware is very crucial in a mission-critical system that cannot be disrupted while subsystems are redefined. Therefore, our system also provides reconfiguration capability guaranteeing hardware change within a very negligible time frame. Thus the advantages of the proposed solution are as follows

- I/O protocols can be installed and executed on-demand which reduces device utilization, power consumption and hardware costs;
- Run-time adaptation satisfies mission specific requirement and reduces the overhead of post-mission maintenance;
- Dynamic reconfiguration capability on a standalone system with minimum user interaction;
- Reconfiguring from a reliable non-volatile memory;
- 100% uninterrupted system operation during runtime reconfiguration.

The rest of the paper is organized as follows. Section II presents the related works. section III presents our generic, modular and dynamically reconfigurable architecture. Section IV provides an overview of the implementation platform and the application scenario used to evaluate our architecture. Next, we evaluate our approach in terms of area overhead, reconfiguration latency and execution time as presented in Section V. Finally, Section VI concludes the paper summarizing our approach and opening some perspectives for future work.

II. Related Works

In this section we provide an overview of the already existing approaches specifically with respect to avionic applications, and highlight the non-availability of dynamic and partially reconfigurable hardware in avionic communication systems. In recent years the feasibility of using reconfigurable hardware is being explored in the field of avionics and defense applications [2], [3], [6]. However, using FPGAs in such applications has its own challenges since time, space, power, reliability and data integrity are highly crucial factors.

Historically, avionic communication buses were implemented using Commercial Off the Shelf Components (COTS). This approach was however not so efficient since the changing application requirements often rendered the system obsolete. Recently, several avionic communication protocols are designed to be implemented on the FPGA for the following reasons. First, more than one communication protocol can be integrated in a single chip. Second, the system can be programmed whenever required and finally it is also obsolescent proof. Extensive work has been done in developing hardware/software co-design for an avionic communication system based on ARINC communication protocol [7]. Another related work also proposes the configuration and deployment of infrastructure and related procedures on a distributed avionic communication system in an FPGA [8].

On the other hand, one of the main challenges of using reconfigurable hardware in space missions is that it has to be radiation and fault tolerant. Single Event Upsets (SEUs) are induced by radiation. The environment where the avionic systems operate, has unfavorable effects in these devices. Therefore it is important to provide a fault-tolerant computing platform for such applications which are prone to radiation effects. Work has been done to address and mitigate the effects of SEUs on FPGA and provide a reliable computing platform [10], [11].

Existing avionics systems typically employ static subsystems that ensure reliability through hardware redundancy. These systems would require multiple redundant copies of hardware or each type of subsystem. This is highly inefficient because it is usually only necessary to replace a small part of the system. In [2] the authors propose a methodology for applications to be a fault-tolerant system and sustain much longer than the traditional approach, using runtime reconfiguration capabilities. This is due to the fact that the reconfigurable hardware utilizes the on-board resources much more efficiently and effectively. Also, using FPGAs for accelerating applications has shown significant performance improvements in aerospace applications [6]. Therefore, based on existing works in the field of avionics, we aim to propose a new runtime reconfigurable avionic communication system.

III. Modular Architecture

In this section we describe the modular architecture that has been designed to demonstrate the DPR of an avionic communication system. The architecture is modular mainly because of two reason. First, hardware instances of communication protocols can be installed and executed on demand. Second, since the communication happens via the FMC, the module can be replaced with another, as long as it adheres to the FMC standard. Thus a single board can support any type of communication interface. Our generic architecture consists of a Xilinx Microblaze processor with some peripherals attached using a Processor Local Bus (PLB), as shown in Figure 1. We have used the XPS_HWICAP, which is the Xilinx ICAP controller, to perform runtime reconfiguration. Finally we also have the avionic communication IP cores (e.g., ARINC429 or CAN controller) and an application core (e.g., JPEG encoder). The ARINC429 and CAN Bus core are a part of the PLB subsystem, since they are serial communication protocols and only require the configuration of few registers for their operations. However, on the other hand, JPEG Encoder

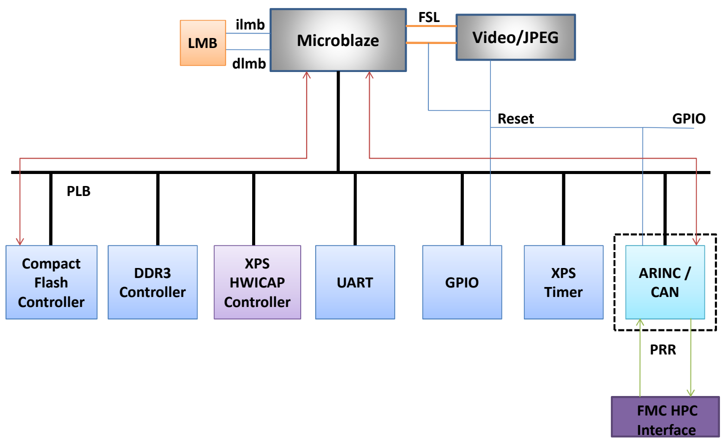

Figure 1: Generic hardware architecture of the dynamic reconfigurable system

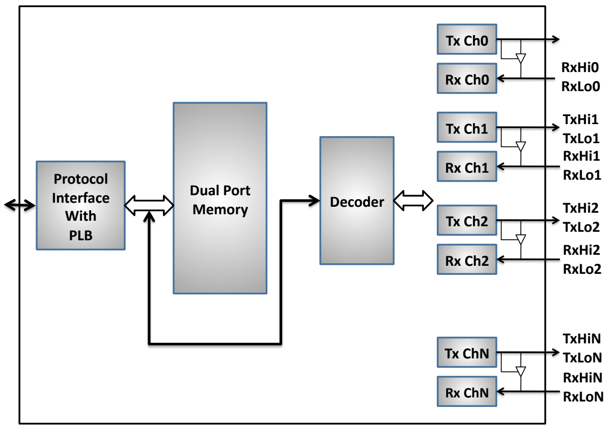

Figure 2: ARINC429 architecture

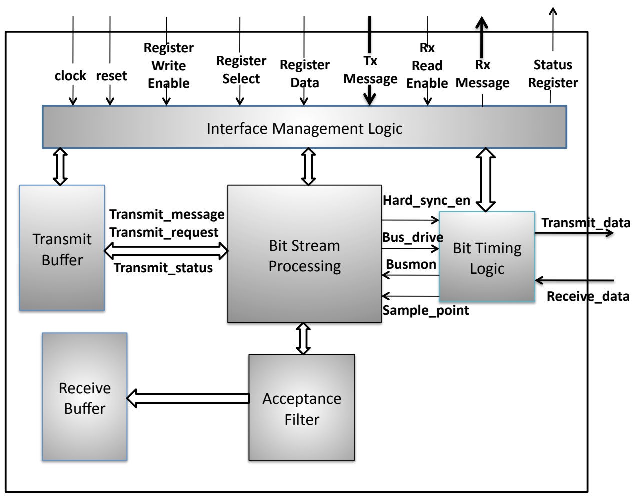

Figure 3: CAN bus controller architecture

[15] requires a high-bandwidth communication link with the processor, for processing and data transfers. Therefore, it is connected to the Processor using Fast Simplex Link (FSL) [14], which is a Xilinx IP developed especially for this purpose.

The architecture is divided into static part and the dynamic part. In our architecture we have chosen the the communication protocols to be the dynamic part. It is important to note that in Figure 1, the communication channels with the FMC are also reconfigured dynamically along with the protocol. The Microblaze processor, peripherals, and the JPEG encoder together forms the the static region. However, we can also fix the JPEG encoder as a dynamic region if need arises due to application requirements (e.g., switching between real-time video monitoring and encoding). In the following subsections we describe the IP Cores designed to be used in the our architecture.

A. *ARINC429 and CAN BUS*

We have developed two IP Cores ARINC429 and CAN Bus controller widely used in avionic communications. ARINC429 is an application-specific technical standard for the avionic data bus used on most higher-end commercial and transport aircrafts. It defines electrical characteristics, word structures and protocol necessary to establish an avionic bus communication. Messages are transmitted at a bit rate of either 12.5 or 100 kilobits per second to other subsystem, which are monitoring the bus messages. The design supports up to 16 Transmit and 16 Receive channels. Since the IP Core has been designed using generic IEEE 1076-1987 standard, it can be implemented on any FPGA device family and architecture. The architecture of the IP Core is shown in Figure 2. The CAN controller implements the Data Link Layer as defined in the document ”BOSCH CAN Specification 2.0”[1]. It implements a serial communication which efficiently supports distributed real-time control with a very high level of security. The design has two communication channels. The architecture of CAN controller is shown in Figure 3.

B. *JPEC Encoder*

In order to demonstrate Dynamic Partial reconfiguration in an industrial and avionic subsystem, we have used a JPEG Encoder [15] to encode the captured frames from an FMC camera module. Although the idea was to use an embedded FMC Low Pin Count (LPC) camera module [16], due to the non availability of the part, we have captured frames with a similar resolution and stored them in a Compact Flash (CF) and later used them for encoding. The encoder is intended to encode raw bitmap images into JPEG compliant coded bit stream. It encodes color images with RGB 24

[1]http://esd.cs.ucr.edu/webres/can20.pdf

bits and YUV input. It has two programmable quantization tables, in order to program the compression ratio of the encoded image. The design runs at 100MHz and is capable of providing a maximum throughput of 136 frames per second at a 50% quality setting. The camera FMC module itself can capture images at resolution of 640x400 with a maximum of 60 frames per second. Thefore the JPEG core offers more than required processing capability. As mentioned before, the encoder is attached to the processor via FSL link. After programming the quantization tables, the image is read from the CF and written into the FSL for encoding by the encoder. Although the design itself is capable of reading from the FSL at every clock cycle, most of the latency is due to the disk access latency of reading the stored frame from the Compact Flash. The encoded images are then converted into ARINC/CAN packets and then tranmitted using a corresponding protocol according to request.

IV. Implementation Platform

This section explains the implementation platform and the application scenario. The architecture described in the previous section was implemented on a Xilinx ML605 using a Xilinx Microblaze embedded processor. The board provides two FMC slots one with a high pin count (HPC) and the other with a low pin count (LPC). It can be used to host two different FMC cards simultaneously. However, due to the non availability of the LPC camera FMC module, we mount only one HPC FMC card to provide ARINC and CAN bus interface. The Xilinx PlanAhead tool was used to generate the partial bitstream. The operational frequency of the processor, buses, peripherals, and JPEG co-processor is 100 MHz. However, the XPS_HWICAP reconfiguration controller operates at 50 MHz to achieve design timing constraints for a reliable reconfiguration. A C program is used to initialize and to interact with the Microblaze and thus the underlying hardware. Initially the partial bitstreams are stored in the Compact Flash memory and they are read by the controller as requested by the application.

The application execution scenario is as follows. A communication protocol (ARINC249/ CAN) runs in parallel with the JPEG encoder. The captured image is encoded, and is transmitted to an avionic subsystem using an appropriate communication protocol (ARINC/CAN) using an FMC interface, as selected dynamically by the user application. The the FMC camera can record full length videos or frames according to the setting. In this setup, we have chosen to monitor individual image frames for testing purposes. Once the encoding starts and while the result are being written into the buffer, the Microblaze reads from the buffer simultaneously and sends the frames to the configured IP core for transmission. The IP core in turn, transmits the frame to another subsystem using the FMC based communication interface. However, if a next frame has to be transmitted to a different communication subsystem, then the application can request for a reconfiguration of the communication protocol. This reconfiguration can be done in parallel when the next frame is still being written to the buffer.

Although our design supports the maximum number of communication channels as dictated by the protocol, the FMC interface has only two communication channels for ARINC249 and CAN. The application can either choose to transmit a frame using only one channel or both. However, the ARINC and CAN protocols itself dictates that there can be up to twenty and sixteen channels respectively. Therefore transmission can be much faster by using an an ARINC/CAN FMC module which supports the maximum number of transmitter/receiver channels. For the above described scenario, measurement of execution time and the resources required are elaborated in the next section.

V. Results

In this section we evaluate our system in the Xilinx ML605 implementation platform. First, we measure the hardware resources required to implement our system and the area overhead of the reconfigurable hardware modules. We then evaluate the latency of dynamically reconfiguring the protocols with various bitstream sizes. Finally we measure the execution time for encoding and transmitting frames of different sizes with and without dynamic reconfiguration.

A. *FPGA Resource Utilization*

Table I shows the area requirements of different FPGA modules. The Microblaze core occupies 573 slices and 101 block RAMs. This is mostly due to the large embedded memory used to store the software executables. The JEPG encoder needs over 2400 slices 79 BRAM blocks and DSP blocks, since it buffers the input image frames and performs DSP algorithms to encode the image. About 4,900 FPGA slices and 3 BRAMs are occupied by the peripheral devices, and the PLB bus interface. The maximum size of the partial reconfigurable region is around 2300 slices and 24 BRAMs. However, the ARINC429 occupies only 1912 (5% of the total FPGA) slices and 2 BRAMs, while the CAN protocol uses even less which is about 689 (about 2% of the FPGA) slices and no block RAMs. The area improvements will be significant, when the I/O protocol provides more number of communication channels on smaller FPGA family. The minimal resouce utilization of our IP cores, makes it every compact and ideal to fit into an even smaller FPGA with a much lower device count.

B. *Reconfiguration Latency*

We measure the time taken to dynamically reconfigure the system with our communication protocols. All the timing measurements shown are measured from the software. Figure 4 shows the reconfiguration latency versus the bitstream size. From the graph, it is quite obvious that the bigger the

	Slices	FFs	LUTs	BRAMs	DSP	ICAP
Microblaze	573	1,737	1,474	101	3	0
JPEG Encoder	2,482	4,112	6,375	79	10	0
Rest of Static Region	4,937	4,543	4,051	3	0	1
Max PR Region	2,340	7,720	9,360	24	0	0
Used by ARINC429	1,912	3,198	7,644	2	0	0
Used by CAN Bus	689	1,016	2,754	0	0	0
Total	7,992 (21%)	18,112 (6%)	21,260 (14%)	207 (24%)	13 (1%)	1 (50%)

Table I: Area utilization of the FPGA design blocks

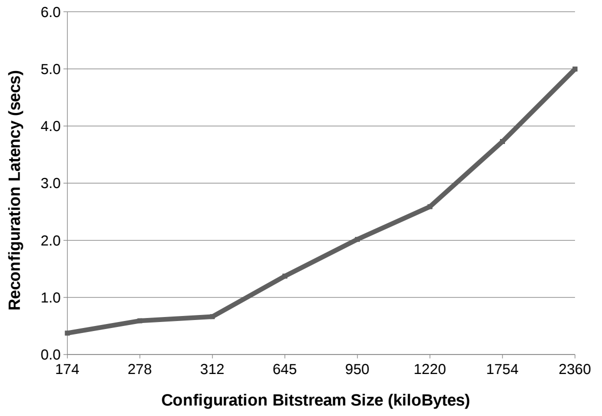

Figure 4: Reconfiguration latency versus configuration bitstream size

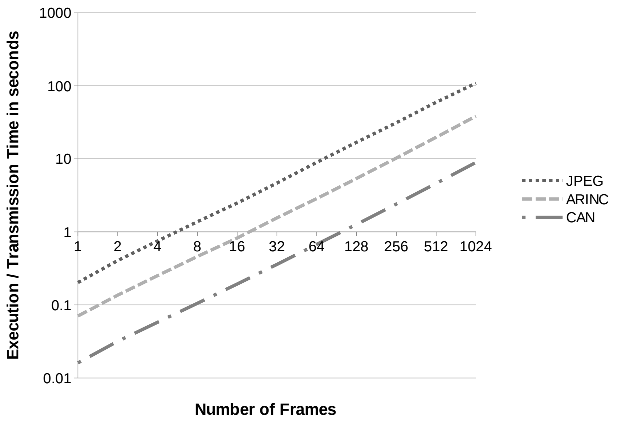

Figure 5: Execution / Transmission time versus number of frames

size of the bitstream the longer the reconfiguration latency. Bitstreams of size less than 500 Kbytes require less than a second to be configured. Configuration stream of size 645 Kbytes which is the size of our bistream, require roughly about 1.3 seconds. While bitstreams larger than a 2 MBytes have reconfiguration latency of few seconds. The read queue in the XPS_HWICAP controller buffers the configuration data before it is fed to the ICAP. However, we see that the throughput of the ICAP controller is less than the theoretical maximum because of the disk access overhead caused by the Compact Flash.

C. Processing and Transmission Times

Figure 5 shows the performance of our system in terms of time taken for encoding and transmitting a frame. The time taken for encoding and transmitting up to 1000 frames has been evaluated. It is worth mentioning that it is execution time in case of JPEG, whereas transmission time in case of ARINC/CAN. It is quite evident that the JPEG encoder takes more time to encode a frame, than the time taken by a protocol to transmit a frame. However both the protocols take different amounts of time for transmitting a frame. This is due to the fact that, ARINC429 can operate only at 12.5 or 100 kilobits per second (here we have chosen the latter as transmission rate). However, the transmission rate for CAN bus can be programed by the user. Therefore, it has been programmed to operate at a much higher transmission frequency. Furthermore, it is also worth noting that, ARINC transmits data using both the channels, while CAN uses only one channel for transmission. This is because, we use a loopback feature to test the correctness of transmission. ARINC can do a loopback on the same channel, however, CAN requires loopback on another channel. Hence this transmission choice. Finally, from Figure 6 it is also quite clear that the latency of reconfiguration can be hidden in certain cases i.e., in cases where the JPEG execution time is more than the reconfiguration latency and transmission time put together. Thus in such cases, the time taken to reconfigure the system is almost zero.

VI. Conclusion

In this paper we have presented the dynamic partial reconfiguration of an avionic communication system. By dynamically reconfiguring a required communication protocol, we obtain significant improvement in area utilization with no degradation in the performance of the communication protocols. This DPR approach also serves as a starting point for two things. First, many industrial applications

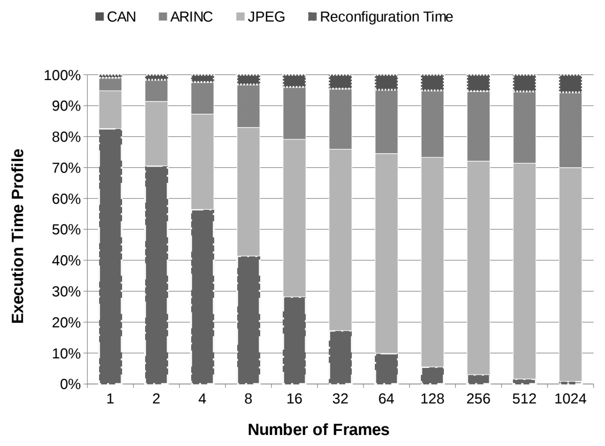

Figure 6: Application profile

have increasingly started using FPGAs due to the flexibility provided by FMC standard. However, they are less or not aware of the benefits of a dynamically reconfigurable system. Therefore, it is a good starting point for converging the industrial requirements of today, with the academic vision for the future. Next, this approach shall be a starting point to incorporate Fault Tolerance and several other features proposed in the literature, which are obligatory requirements for an aerospace application. Another aspect is to also consider improving the reconfiguration time. This involves the exploration of some possible alternative high-speed reconfiguration controllers such as UPaRC or Enhanced ICAP hard macro, to improve the reconfiguration time required for such a system.

Acknowledgment

The authors would like to thank ANRT. This work is being partially funded by ANRT in context of their CIFRE doctoral research grant: *CIFRE 2011/0953*. The Industrial Research Training Convention (French: **C**onvention **I**ndustrielle de **F**ormation par la **R**echerch**E**) is fully funded by the French Ministry of Higher Education and Research, which has entrusted the implementation to the National Association for Research and Technology (French: **A**ssociation **N**ationale de la **R**echerche **T**echnique).

References

[1] Pedersen, R.N. *FPGA-based military avionics computing circuits*. Aerospace and Electronic Systems Magazine, IEEE, 2004.

[2] Will Hua Zheng and Marzwell, N.I. and Chau, S.N. *In-system partial run-time reconfiguration for fault recovery applications on spacecrafts*. IEEE International Conference on Systems, Man and Cybernetics, 2005, Hawaii, USA.

[3] Osterloh, B. and Michalik, H. and Habinc, S.A. and Fiethe, B. *Dynamic Partial Reconfiguration in Space Applications*. NASA/ESA Conference on adaptive Hardware and Systems, 2009, California, USA.

[4] S. Hauck and A. DeHon *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*. A.Morgan Kaufman Publishers, 2007.

[5] VITA57[Online]. *FPGA Mezzanine Card*. http://www.vita.com/fmc.html.

[6] L.A. Cardona., J. Agrawal., Y. Guo., J. Oliver. and C. Ferrer *Performance-Area Improvement by Partial Reconfiguration for an Aerospace Remote Sensing Application*. International Conference on Reconfigurable Computing and FPGAs, 2011, Cancun, Mexico.

[7] Cloute, F. and Contensou, J.-N. and Esteve, D. and Pampagnin, P. and Pons, P. and Favard, Y. *Hardware/Software Co-Design of an Avionics Communication Protocol Interface System: an Industrial Case Study*. Proceedings of the Seventh International Workshop on Hardware/Software Codesign, 1999. (CODES '99), Rome, Italy.

[8] Ivancic, W.D. *Modular avionics for seamless reconfigurable UAS missions*. IEEE Aerospace Conference, 2006, BigSky, Montana, USA.

[9] Lopez, J. and Royo, P. and Barrado, C. and Pastor, E. *Modular, cost-effective, extensible avionics architecture for secure, mobile communications*. Digital Avionics Systems Conference, 2008, St. Paul, Minnesota, USA.

[10] Sterpone, L. and Margaglia, F. and Koester, M. and Hagemeyer, J. and Porrmann, M. *Analysis of SEU effects in partially reconfigurable SoPCs*. NASA/ESA Conference on Adaptive Hardware and Systems (AHS), 2011, California, USA.

[11] Lanuzza, M. and Zicari, P. and Frustaci, F. and Perri, S. and Corsonello, P. *Exploiting Self-Reconfiguration Capability to Improve SRAM-based FPGA Robustness in Space and Avionics Applications*. ACM Trans. Reconfigurable Technology Systems, 2010, NY, USA.

[12] Bonamy, R. and Hung-Manh Pham and Pillement, S. and Chillet, D. *Ultra-fast power-aware reconfiguration controller*. Design, Automation Test in Europe Conference Exhibition (DATE), 2012, Dresden, Germany.

[13] Hansen, S.G.; Koch, D.; Torresen, J. *High Speed Partial Run-Time Reconfiguration Using Enhanced ICAP Hard Macro*. IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011, Alaska, USA.

[14] Xilinx[Online] *Fast Simplex Link (FSL)*. http://www.xilinx.com/support/documentation/xapp529.pdf.

[15] Opencores[Online] *JPEG Encoder*. http://www.opencores.org.

[16] Avnet[Online] *Industrial Camera FMC Module*. http://www.em.avnet.com/.../Xilinx-Spartan-6-FPGA-Industrial-Video-Processing-Kit.aspx.