Toward Generic and Adaptive Avionic Test Systems

George Afonso
EADS Innovation Works
george.afonso@eads.net

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

Nicolas Belanger
Eurocopter Group
nicolas.belanger@eurocopter.com

Martial Rubio
Eurocopter Group
martial.rubio@eurocopter.com

Stephan Stilkerich
EADS Innovation Works
Stephan.Stilkerich@eads.net

Jean-Luc Dekeyser
LIFL, USTL, INRIA Lille-Nord Europe
jean-luc.dekeyser@lifl.fr

Abstract

In the manufacturing process, the test phase is considered the most important challenge for designers of safety-critical avionic systems found in modern helicopters. Indeed, these systems often operate in uncertain conditions and they must provide safety, fault tolerance, and deterministic timing guarantees. For large range of helicopters, the functioning of the different system units is checked using several software and hardware environments. Unfortunately, this test methodology increases the time-to-market and the cost of the final product. Focusing this issue, we propose a generic test environment that can adapt easily to the helicopter range and the Unit-Under-Test (UUT).Within this environment, we conceived a hybrid CPU/FPGA architecture in order to design innovative avionic test systems that meet performance and flexibility goals. Furthermore, we defined an efficient test methodology that favors the reuse of hardware and software models, the adaptability of the system according to the scenario, and the interoperability of heterogeneous units. The presented case study shows the strong impact of our environment to reduce the complexity of the test phase. The Eurocopter corporation intends to adopt this environment as a part of the next generation test benches.

1 . Introduction

Test Systems have always been considered as an essential part in the avionic development cycle. Due to the ever-changing face of technology, the Eurocopter[1] research department leads to the development of *Pro-Active Test Sys-tems*. The objective of this project is to bring reliability and competitiveness to the avionic industry. In the first quarter of 2010, the development of new test system has been started.

In present industrial practice, different test benches are used for the verification of various helicopter ranges (EC175, EC135, etc.) and Unit(s)-Under-Test (UUTs) (automatic pilot, navigation, etc.). Each test bench relies on a specific hardware architecture and software tools. This is due to the heterogeneity of the helicopter parts (which are under test) in terms of computing requirements and handled data structures. In general, several specialised CPU boards are needed to satisfy real time constraints which leads to sophisticated synchronization and communication schemes. In addition to this, dedicated avionic I/O boards (Arinc 429, 1553, etc.) are required depending on the UUTs. This test methodology calls for separate teams with different domain experts in order to achieve the test of each part. The overall avionic system verification is done through the first prototype of the helicopter. Today, this test process is very complex and expensive to perform.

This paper addresses the above test methodology limitations and calls for an innovative avionic test environment. Our main objective is to build up a generic test environment by the means of offering more flexibility regarding the selection of the suitable target avionic system. An efficient test methodology favors the reuse and the interoperability of hardware and software models while switching between different scenarios. Furthermore, automation in manufacturing process is a key for increasing the productivity and reducing the cost.

Our contributions in the avionic test environment domain can be summarized as follows:

- First, we proposed a hybrid CPU/FPGA architecture

---

[1]Eurocopter is the leader in civil and military helicopter manufacturing. http://www.eurocopter.com/

for the test system. Indeed, today multicore CPUs come with high computation rates while the FPGA offers flexibility and adaptability to the system. Within our environment, a great care has been devoted to the real time aspect in order to satisfy tight computing and communication deadlines. Our proposal relies on industrial and certified technologies that can be embedded easily on the final avionic product.

- Second, we defined an efficient methodology that makes profit from the hybrid architecture to adapt the test system according to the target realization. The reuse and the interoperability advantages are ensured in this methodology with the help of the reconfigurable technology.

This paper is organized as follows. After Section 2 which presents the related works, Section 3 exposes an overview of the avionic test loop. Our generic and adaptive test environment is decribed in Section 4. In order to evaluate our approach, Section 5 presents the experimental results for a typical avionic test system.

2 . Related works

For the past twenty years, the avionic test systems were based on real time specific hardware architectures such as the well-spread VME CPU boards [5]. The VMEBus is particularly efficient to allow Input/Output (I/O) event management, multi-processing synchronization, and a transparent access to the different hardware resources. As a most of aeronautic firms, Eurocopter has integrated the VMEBus as a standard backbone for the test benches of embedded helicopter systems. The proprietary test system named ARTIST is based on VME technology and the real time Operating System (OS) VxWorks. These technologies have been used for all helicopter benches in order to validate the avionic equipments.

Due to the present performance requirement, an increase in the computation rates is needed, but it cannot be delivered by the VME CPU boards. Furthermore, this solution is considered as an expensive maintainable technology. To overcome these drawbacks, Eurocopter recently decided to move to a "half generation" test system based on high performance PC or workstation solution. Upcoming architectures are based on multi-core computer plugged with I/O boards to communicate with the equipments under test. Eurocopter has selected the PEV1100 VME Bridge solution [3, 4] from the Swiss company IOxOS [2]. The PEV1100 allows a local host to interface with a VME64x bus using a PCI Express (PCIe) external cable which offers transparent access to I/O boards. To achieve higher communication performances, IOxOS Technology had developed a dedicated

interface between the PCIe and the VME64x bus. This interface is built with the latest FPGA (Field Programmable Gate Array) technology in order to implement PCIe endpoint hardware cores.

The usage of multicore hosts allows an immediate increase in the capacity of computation. An important outcome of this transition is the refusal of the obsolete CPU boards. However, this solution cannot guarantee the real-time criteria while the execution of concurrent tasks due to the lack of an appropriate OS environment. Furthermore, this solution brings new communication latencies between the CPUs and I/O boards plugged in the VME backplane.

Among existing avionic test systems provided by cutting-edge firms, we quote Aidass family [12] used in particular for Eurofighter EADS Military Air Systems, U-Test [4] developed by EADS TestServices, and ADS2 from Techsat GmbH [3]. The proposed solutions are fully based on CPUs resources (PC or VME boards) and are close to Eurocopters solutions. These test systems can both deal with I/O management and simulation environments. Today, the management of increasing required power relies on additional CPU installation. In this work, our proposal is to make profit from the new available hardware computing resource (FPGA) and to build adaptive avionic test systems. Indeed, FPGA technology could offer a higher computation rates comparing to CPUs up to 10x [2]. It could implement heavy models in a hardware fashion and hide communication latencies with I/O devices.

3 . The avionic test loop

In order to perform a complete simulation, we need to modelize each part of the helicopter and the environmental parameters (weather conditions, geographical factors, etc.). Fig. 1 presents a simplified test loop system that simulates the helicopter behavior including three models: the *flight mechanic model*, the *navigation model* and the *automatic pilot model*. In the initialization phase, the flight mechanic model takes several parameters such as the initial position relative to the ground and the aircraft configuration file. As a main result, this model gives back an *equilibrium position*. In addition, it sends the *common data area* structure containing the position and the speed of the aircraft to the navigation model. This later computes the helicopter destination and sends it to the automatic pilot model via the *ordered roll* structure. Finally, the flight control is managed by the automatic pilot.

In the first test step, software models (simulating each part of the aircraft) are used in order to perform a functional verification of the overall system. The above described test system is really simplified in terms of number of models and complexity. Thanks to the current multicore architectures,
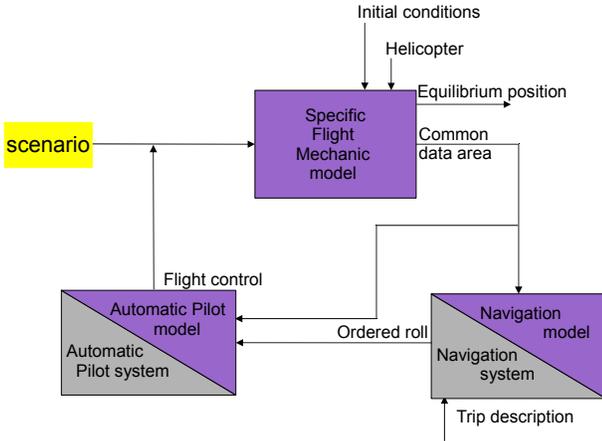
Figure 1: The simplified test loop system

the computing capacity is becoming important, nevertheless they cannot guarantee the execution of tasks in a real-time. This is especially important because a task overflow could yield to the test cycle failure. For this reason, several CPU boards are used in the current test benches. In the next test steps, each software model can be replaced by the corresponding real hardware (the grey parts in Fig. 1) which calls for additional I/O boards. Hence, a strong effort is needed to set up the interoperability between heterogeneous software and hardware models in the same environment which increases the complexity of the test phase.

4 . Adaptive avionic test system

4.1 A hybrid hardware architecture

The main objective of the hybrid architecture is to exploit the new available hardware computing resource (FPGA) in order to build up adaptive avionic test systems. Fig. 2 presents the proposed hybrid architecture for the next avionic test systems composed of Multi-core CPUs and FPGAs. Our expectation of the above described architecture is to prototype some models which can be eligible and relocated in the FPGA. The objective is to increase the performances of these models and to reduce the communication latencies by the means of embedding the different parts in the same chip. To do so, we need first to profile our avionic test loop in order to extract the complex models that will be implemented in the FPGA. Second, different hardware model configurations will be explored to reach an optimal well-balanced global system. Indeed, FPGA technology could implement heavy models in a hardware fashion with the management of the parallelism degree to address the real-time constraints of the application.

Actually, most of the current model are mainly based on software IP's (Intellectual Property). Many tools such as

Vector Fabrics vfAnalyst [16], Intel Parallel Studio [8] or CriticalBlue Prism [6] can help to extract the parallelism degree by analyzing data dependencies and thus to decide whether a hardware transcription of the concerned model is interesting or not. Moreover, other existing tools such as ROCCC [15] or GAUT [7] can provide a translation from C to VHDL.

In addition to the avionic models, FPGAs will implement I/O IPs such as Arinc 429 in order to perform the communication with the UUT. Each node of the architecture presented in Fig. 2 needs to communicate with other computing elements using a fast bus, such as Ethernet or PCIe. To make our choice, communication latency with respect to the real time constraints is considered the most important metric [13].
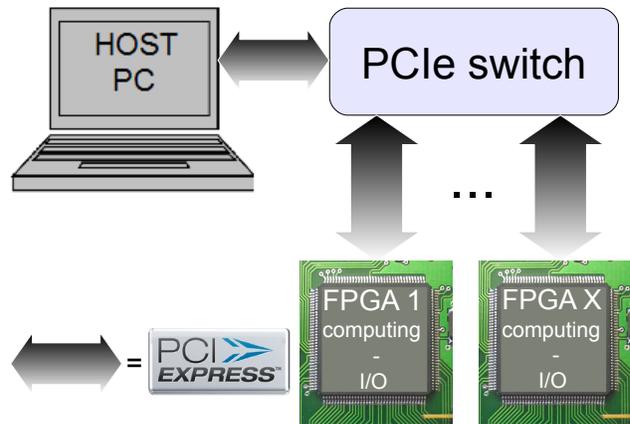

Figure 2: Hybrid hardware architecture

4.2 Design methodology

Nowadays, each aircraft part has a dedicated test bench that must be maintained as long as the related range still functional. To reduce the cost of maintainability, our new adaptive architecture including FPGAs for the computing and I/O parts is considered a generic test bench. Indeed, there are specific I/Os and model configurations for each aircraft. Fig. 3 presents the design methodology for adaptive avionic test system. In order to test all Eurocopter aircraft, a specific library containing software and hardware models such as the *mechanical flight* or the *navigation*. The selection of the appropriate models will be adapted according to the defined system constraints. Actually, the hardware/software mapping is performed manually by our designers. However, we are planning to implement a heuristic method that can lead to a better optimized implementation.

In order to obtain a fast test system prototyping, we need to deal with the heterogeneity of both hardware and software parts. In current industrial practice, manual coding is
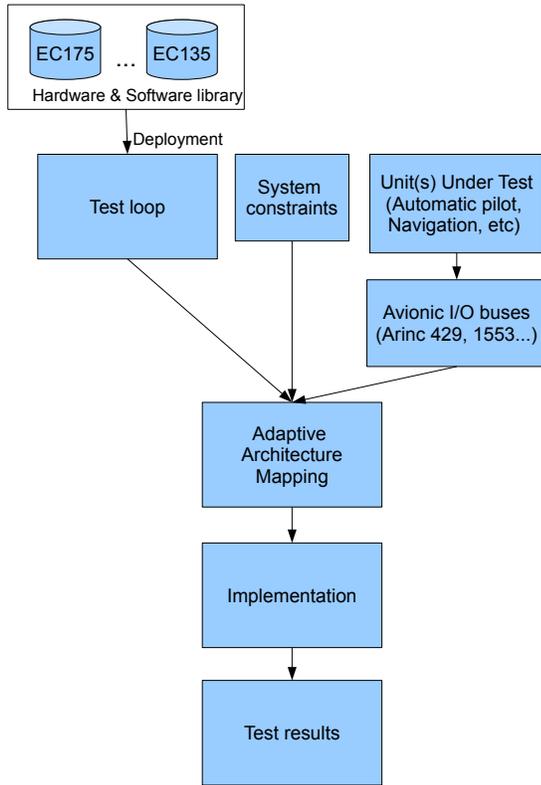
Figure 3: Design methodology for adaptive avionic test system

still widely adopted in the development of hybrid architectures, which is clearly not suited to manage the complexity intrinsic in these systems. For designers, this approach is very tedious, error-prone and expensive. To overcome this challenge, we propose the usage of a *Model-Driven Engineering* [11] (MDE) approach in the specific context of CPU/FPGA hybrid system design. The objective is to build a tool that turns automatically a high level specification of the test system into an executable implementation.

Hybrid system modeling: We propose the usage of the MARTE [4] standard UML profile to model the hybrid avionic test system. Later, a compilation chain will be defined to turn automatically the high level specification into an executable implementation. In the MARTE specification, an application is a set of tasks connected through ports. Tasks are considered as mathematical functions reading data from their input ports and writing data on their output ports. Fig. 4 illustrates the modeling of the avionic test loop described in section 3. Its specification is a simple

directed cyclic graph. In addition, MARTE allows to describe the hardware architecture in a structural way. Typical components such as HwProcessor, HwFPGA and HwRAM can be specified with their non-functional properties. Fig. 4 shows an example of an hybrid multiprocessor architecture. The main component of this architecture is composed of the Xeon-X3370 processor (multicore CPU) and the Virtex-6 Xilinx FPGA. Furthermore, MARTE provides the Allocate concept as well as the concept specially crafted for repetitive structures Distribute. This latter concept gives a way to express regular distribution of tasks onto a set of processors or FPGA resources. Fig. 4 illustrates two types of distribution (*timeScheduling* and *spatialDistibution*) depending on the target hardware platform. The different models of our avionic test loop can be mapped onto the host multicore processor, the embedded processor (Microblaze) or the hardware resources in the FPGA.

Deployment: To transform the high abstraction level models into an implementation code, very detailed deployment information must be provided. In particular, each elementary component must be linked to an existing code. For this purpose, a *deployment* profile is introduced. A key point in our methodology is to favor the reuse of IP block. In this profile, we introduce the concept of *virtualIP* which expresses one functionality, independently of the target implementation (CPU or FPGA). It contains one or several *softwareIP* or *hardwareIP*, each one could be used to define a specific implementation at a given programming language (VHDL or C). Fig. 5 shows the *virtualIP* of the `virtualAutomaticPilot` which contains two *softwareIP* written first in C language for a software execution on CPU and secondly described in VHDL language for an FPGA implementation. Using the *ImplementedBy* dependency, the designer can select the adequate IP for each hardware and software component. The implementation which fits best the target will be used to reify the component during the compilation phase. For automatic code generation, we use the concept of *CodeFile* to specify the required code and compilation options.

5 . Case study

In the case study, the proposed test methodology will be applied to the example presented in Section 3. For the system requirement, a time constraint of 20 ms is defined. The first step consists to profile the application using a software tool on the host machine in order to estimate the execution time consumed by each model. As a main result, the flight mechanic model takes up to 90% of the processor utilization while other models occupies only 10% of the total execution time. After profiling and identifying the performance bottlenecks of the application, we have explored different hardware configurations to implement the flight mechanic
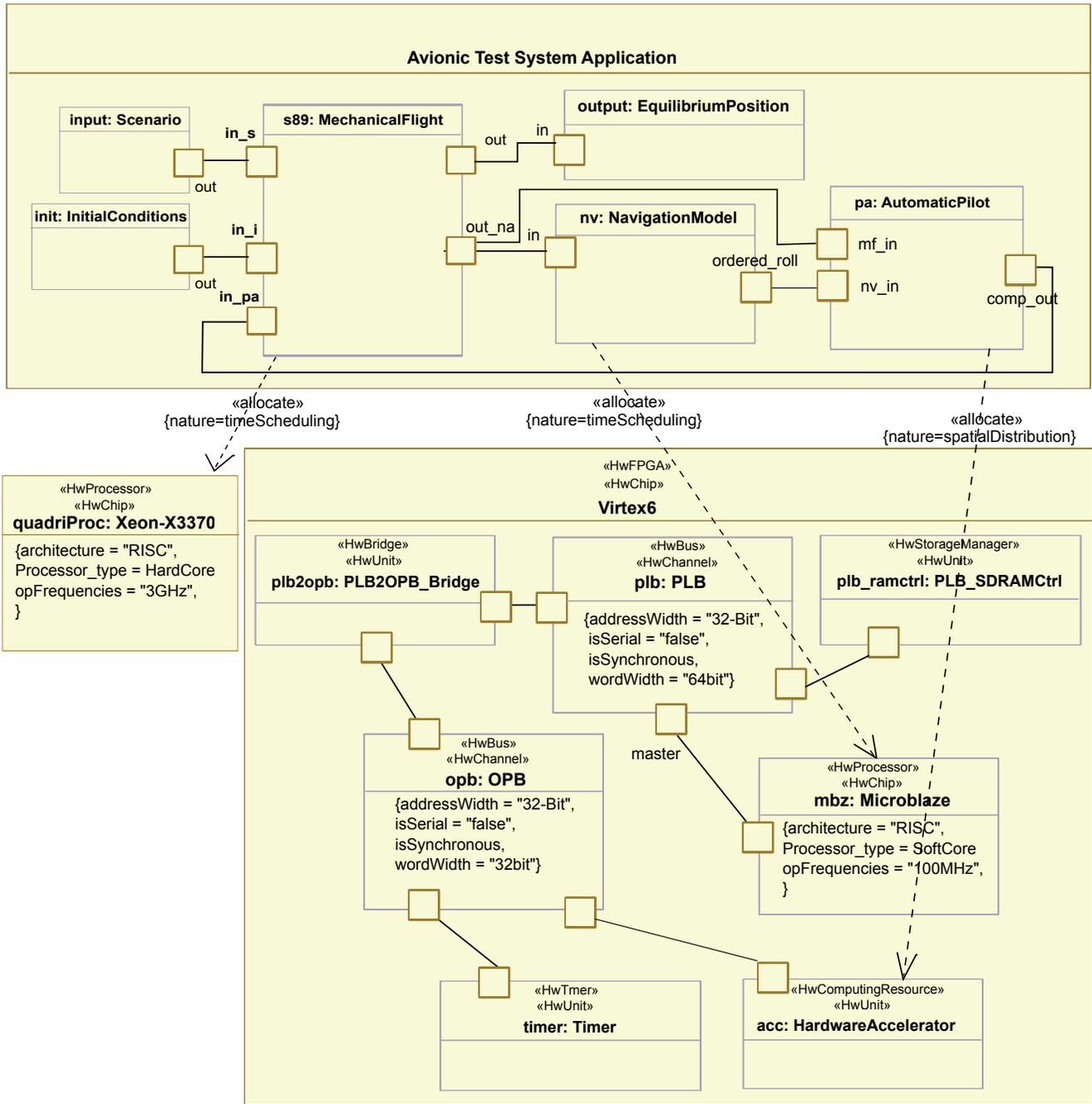
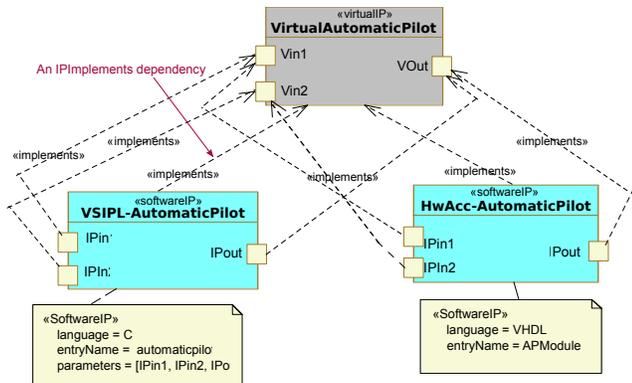Figure 4: Hybrid system modeling with MARTE

Figure 5: Example of the automatic pilot task deployment

Figure 6: Different hardware implementations for the flight mechanic model

model in the FPGA. Afterward, the communication cost is analyzed between the different nodes of our hybrid architecture. Indeed, the computation is not located only in the FPGAs, but also in the Multi-Core processor. So we need to profile the communication part of our architecture to guarantee that it doesn't provide latency which could make the test cycle fail. Indeed, in this case study, the defined time constraint is 20ms, so, it becomes inconceivable that the communication time is higher than this period. The two most common buses were chosen for our study are the Ethernet and the PCIe x8.

5.1 Implementation results

We considered the ML605 Virtex-6 Xilinx board as the hardware platform in which the model is implemented. In our baseline system (configuration A), all the functional blocks of the model are executed on a single Micoblaze processor running at 100 MHz. The processor communicates with the local BRAM memory (Block RAM), where the applications local data and instructions are stored on-chip. Both instruction and data caches are also used. In the second configuration B, CORDIC (Coordinate Rotation Digital Computer) hardware accelerators are used with the Microblaze to improve the execution time of the numerous trigonometric functions intrinsic in the model. Indeed, the CORDIC hardware accelerator, can perform trigonometric functions faster than a Microblaze [14]. In the third configuration C, we used the SDRAM external memory for instruction and data storage. The above described configurations are illustrated in Fig. 6.

To complete the first part of this study, we need to design a communication medium which must be fast and reliable. In the first configuration X, we used OpenSplice [10] DDS [9] (Data Distribution Service) to perform the communication between two computers though an Ethernet bus in a Linux (Fedora) Operating System environment. The DDS communication middleware offers two modes. The "Best Effort" will deliver the data as faster as it can without re-
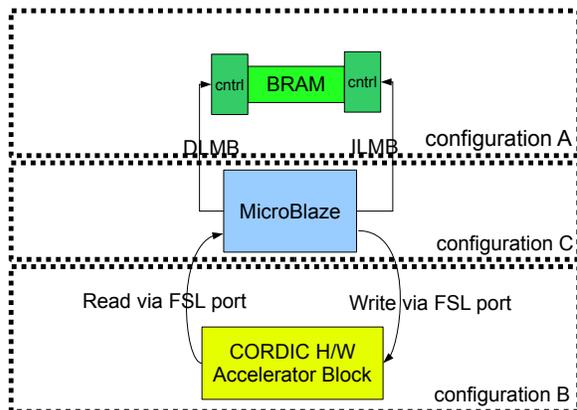
liability concept. While "Reliable" will bring consistency to the communication but the data delivering time will be worse compared to the "Best Effort" mode. In the "Reliable" mode, a maximum time for data delivering can be added as a parameter. In the second configuration Y, we use a PCIe x8 bus to perform a communication through our hybrid architecture using the same OS environment (Linux). In the third configuration Z, we improved the Configuration Y using a real time patch (Xenomai [17]) to add real time capabilities for our OS environment. For each step, we measured the time needed for a data to perform a round-trip between two nodes of the tested architectures.

Table 1: Models implementation results.

Configuration	A	B	C
Used Slice Flip Flops (%)	2	4	4
Used BRAM (%)	12	20	13
Execution time (ms)	30	0.9	0.94

The objective of the first part of this study is to obtain several configurations characterized with different tradeoffs in terms of performance and resource occupation in the FPGA. According to the implementation results, several remarks can be drawn. First, configuration A that runs the flight mechanic model as a software on the Microblaze has the highest execution time (30 ms). With respect to the performance constraint, the execution time of flight mechanic model is not respected. Thus, configuration A cannot be adopted. Configuration B that uses dedicated hardware accelerators increases the system performance up to 30x allowing a 0.9 ms execution time. However, it consumes more resources than : 20% of the available BRAM and 4% of the total number of slices. Configuration C offers a good trade-

Table 2: Communication implementation results.

Configuration	X Best Effort	X Reliable	Y	Z
Average time (us)	400	500	18	7
Maximum latencies (us)	25400	30000	7000	86

off between execution time (0.94 ms) and area utilization. Indeed, this configuration requires only 13% of the BRAM and 4% of the total number of slices. In a common designer strategy, the choice of a given configuration will depend on several parameters such as the number of models and the test system constraints.

The objective of the second part of this study is to obtain the best communication configuration for our hybrid test system in terms of both performance and reliability. According to the implementation results, several remarks can be drawn. First, the configuration X with Best Effort mode offers less average communication time (400 us) than the Reliable mode (500 us) which respects the fixed maximum time parameter set to 600 us. However, for both modes, intolerable communication latencies (respectively 25.4 and 30 ms) have been detected. The configuration Y reduces significantly the average communication time up to 18 us, but always intolerable latencies (7 ms) are again detected. Those latencies are introduced by the Linux OS. For the previous configurations, it becomes clear that we need to run our CPU/FPGA communication by using a real time environment to remove all those latencies. Moreover, in order to keep our Linux environment, we choose the Xenomai solution.

5.2 Xenomai: Real-Time Framework for Linux

Xenomai [17] is a real-time development framework cooperating with the Linux kernel, in order to provide a pervasive, interface-agnostic, hard real-time support to user-space applications, seamlessly integrated into the GNU/Linux environment. Xenomai is based on an abstract RTOS core, usable for building any kind of real-time interface, over a nucleus which exports a set of generic RTOS services. Any number of RTOS personalities called skins can then be built over the nucleus, providing their own specific interface to the applications, by using the services of a single generic core to implement it. The benefits of this approach is mainly to keep the development process in the GNU/Linux user-space environment, instead of moving to a rather hostile kernel/supervisor mode context. This way, the rich set of existing tools such as debuggers, code profilers, and monitors usable in this context are immediately available to the application developer. Moreover, the standard GNU/Linux programming model is preserved, allowing the application to use the full set of of facilities existing in the user space (e.g. full POSIX support, including inter-process communication). The Fig. 7 shows a Xenomai patched Linux kernel

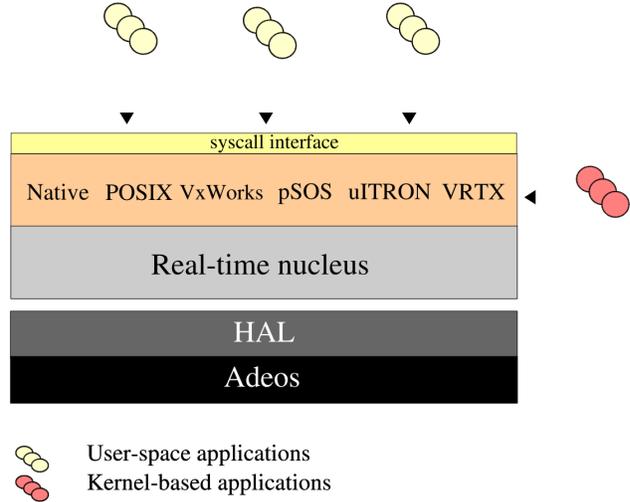using the Adeos [1] virtualization layer, which in turn ensures RTAI low interrupt latencies.

Figure 7: The Xenomai architecture

For the configuration Z, we did not modify the PCIe driver, but only on the program performing the data round-trip for a real time run. This configuration gives us the best round-trip communication time (7 us) and acceptable latencies (86 us) for our application in Linux/Xenomai environment. Moreover, the modification of the Linux kernel does not change the global behavior of our environment. Xenomai brings a good tradeoff between a standard Linux environment and a real time OS. In addition to this, the PCIe standard allows an efficient solution to deal with the interoperability between hardware and software models mapped respectively on FPGA and CPU nodes.

6 . Conclusion

In this paper, we have emphasized first the benefits of using hybrid CPU/FPGA architectures in the particular context of adaptive avionic test systems. Indeed, FPGAs bring flexibility and reliability to the helicopter test cycle. Second, a real time environment has been developed to satisfy tight computing and communication deadlines. It allows to run heterogeneous hardware/software models mapped on a hybrid CPU/FPGA architecture. It is so generic that can adapt different test avionic scenarios. In order to deal with the design complexity of hybrid systems, we have defined a test

methodology that improves the productivity and reduces the development cost.

In the case study, different hardware and software solutions have been explored to target an optimized implementation of the test system. In future works, our investigation will concern the automatization of the test environment using MDE.

References

[1] Adeos. A flexible environment for sharing hardware resources among multiple operating systems. http://home.gna.org/adeos/.

[2] S. Asano, T. Maruyama, and Y. Yamaguchi. Performance Comparison of FPGA, GPU AND CPU in Image Processing. In *19th IEEE International Conference on Field Programmable Logic and Applications, FPL*, Prague, Czech Republic, Aug. 2009.

[3] N. Belanger, J. Bovier, J.-F. Gilot, J.-P. Lebailly, and M. Rubio. Multi-Core computers and PCI Express The future of data acquisition and control systems. In *ETTC International Conference*, Toulouse, France, 2009.

[4] N. Belanger, N. Favarcq, and Y. Fusero. An open real time test system approach. In *IEEE International Confererence on Advances in System Testing and Validation Lifecycle*, Porto, Portugal, Sept. 2009.

[5] N. Belanger and J.-P. Lebailly. Promoting avionic test systems as productivity enablers. In *The fifth International Conference on Systems*, Les menuires, France, 2010.

[6] CriticalBlue: Prism. An analysis exploration and verication environment for software implementation and optimization on multicore architectures. http://www.criticalblue.com.

[7] GAUT: High-Level Synthesis tool From C to RTL . http://www-labsticc.univ-ubs.fr/www-gaut/.

[8] Intel Corporation: Intel Parallel Studio. http://software.intel.com/en-us/intel-parallel-studio-home.

[9] Object Management Group. The open, multiplatform, interoperable publish-subscribe middleware standard. http://www.omgwiki.org/dds/.

[10] OpenSplice: Open source Data distribution service. http://www.opensplice.com.

[11] Planet MDE. Model Driven Engineering, 2009. http://planetmde.org.

[12] H. Plankl. Embedded solutions for development tests, component tests and system integration in the test centre. In *Aerospace Testing*, Munich, Germany, 2009.

[13] PLX Technology. The case for pci express in the blackplane, 2010. http://www.plxtech.com.

[14] Richard Griffith and Felix Pang. Microblaze system performance tuning. 2008.

[15] ROCCC: Riverside Optimizing Compiler for Configurable Computing. http://roccc.cs.ucr.edu.

[16] VectorFabrics: vfAnalyst. Analyze your sequential C code to create an optimized parallel implementation. http://www.vectorfabrics.com/.

[17] Xenomai: Real-Time Framework for Linux. http://www.xenomai.org.