HYBRID AND MULTICORE OPTIMIZED ARCHITECTURES FOR TEST AND SIMULATION SYSTEMS

G. Afonso[(a)], N. Damiani[(b)], N. Belanger[(b)], R. Ben Atitallah[(c)], M. Rubio[(b)]

(a)EADS Innovation Works – INRIA Lille Nord Europe

(b)Eurocopter Group

(c)LAMIH, UVHC

george.afonso@inria.fr, nicolas.damiani@eurocopter.com, nicolas.belanger@eurocopter.com, rabie.ben-atitallah@lifl.fr, martial.rubio@eurocopter.com

ABSTRACT

Continuously growing aerospace industry competitiveness pushes avionic actors to revisit and strengthen their Verification & Validation (V&V) lifecycle related means. In this perspective, the areas of Test and Simulation (T&S) systems are now an unavoidable convergence path. Yesterday considered as different expertise fields, T&S should rely on common frameworks in the future supported by cutting-edge architectures. In the first part, the paper introduces the current main changes of Eurocopter's V&V process. Then we present the RISE simulation tool used in different applications fields such as Training Media, investigation studies and rapid prototyping. We highlight RISE multi-threading management capabilities and its associated simulation benefits. Capitalizing on simulation state-of-the-art, we introduce the new research project CHARTS. In order to meet performance, power and flexibility goals requested by T&S systems, this project proposes a challenging and a scalable hybrid architecture based on a multi-core processor tightly-connected to FPGA.

Keywords: Simulation, multicore and hybrid architecture, FPGA, avionics real time test bench

1. INTRODUCTION

For many years, it has been recognized that the more design issues are discovered early in the development cycle, the more it reduces the cost and the time associated to the project. It is also recognized that the majority of defects discovered in the integration testing phase can be attributed to the specification errors that could be detected earlier if the appropriate tools and models were available. In this perspective, it is necessary to consolidate the usual development cycle with a solid system approach allowing to early check/validate the adequacy and the consistency of the models specifying the system level. Such a system approach can be conceived only if the means of Verification/Validation (V&V) are present upstream. In another meaning there is a solution to simulate as soon as possible the (sub-) system level and not only the component level. The testing phase should begin at the earliest: testing in simulation before testing on benches and on benches before testing in the real world. This paper addresses the above challenges in the Eurocopter product development cycle. In the present industrial practices, different simulation tools and test benches are used for the verification of various helicopter ranges (EC175, EC135, etc.) and Unit(s)-Under-Test (UUTs) (automatic pilot, navigation, etc.). This methodology calls for separate teams with different domain experts in order to achieve the simulation and the test of each part. The overall avionic system verification is done through the first prototype of the helicopter. Today, this process is very complex and expensive to perform. Actually, there is an essential need of a seamless methodology that could help designers during the V&V cycle starting from a full software simulation to the integration testing benches.

2. V&V TRANSFORMATION

Historically, tools used for Avionic Simulation and for Real-Time Integration Testing have often been decoupled. This matter of fact could be explained by technical choices: Real-Time Operating System (RTOS) competitiveness, hardware access (Arinc 429, Milbus 1553, Analog/Digital, etc.), and models management capabilities. Due to the hard time-to-market requirement, practices have started to change during the last years. With the new technologies in the fields of RTOS and the emergence of virtualization solutions, aerospace actors are reconsidering their methodologies to verify and validate critical embedded systems. In the next subsections, we present gradually the evolution of test bench architectures and the impact on the related software supports at Eurocopter. The result of this wide technological motion is the vital need to converge toward unified Test and Simulation tools.

2.1 Current integration test bench architecture

For the twenty past years, full VME architecture was the best solution to manage the hard real-time testing of avionic systems (Peckham 2006). VME bus is particularly efficient to allow I/O event management: multiprocessing synchronization and access to different hardware resources (CPUs and I/O boards) in a transparent way. As most of aeronautic actors, Eurocopter has integrated the VME bus as a standard backbone in the test integration benches of helicopter embedded systems. The current

Simutools 2013, March 05-07, Cannes, France

DOI 10.4108/icst.simutools.2013.251759

Integration Test System Architecture used at Eurocopter is introduced on the Fig. 1:

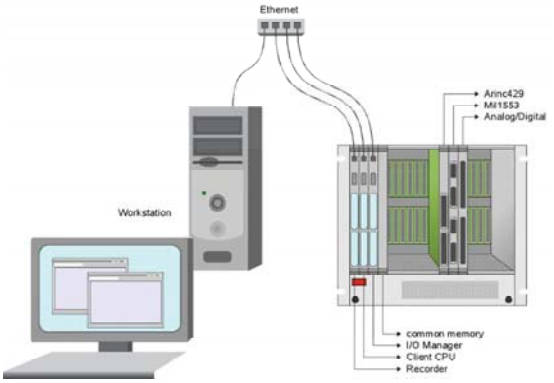

Figure 1: Current integration test bench architecture.

Eurocopter validation tool for integration testing uses a VME backplane communication. A memory area is mapped on the VME and is shared by all CPU's and I/O boards using Scramnet intercrate reflexive optical memory link. An interface between users and the real-time simulation is ensured through an Ethernet link. Until now, this robust architecture has demonstrated over years its efficiency to validate complex avionic systems in the respect of real-time constraints. But some limitations pop out as the helicopter systems complexity is heavily growing. This matter of fact requires increasingly external application plug-in to perform a real-time integration testing. In parallel, real-time code debugging is very difficult relatively to "Host to Multi Targets" architecture. In order to anticipate the next obsolescence of its Test Systems, Eurocopter chose a new PC-based solution regarding the mandatory needs to master the increasing complexity of avionic systems.

2.2 New test bench architecture: an approach by memory mapping from Ethernet to PCIexpress

Eurocopter real-time test bench architecture requires for coming years openness to external various applications (automation test description, data analysis tools, etc.) and an easier real-time debugging framework. Taking into account the important and the costly hardware legacy on benches, the new Test System solution has been designed in a cost saving perspective with hardware patrimony conservation. To do so, a new PC solution based on the introduction of the PEV1100 VME Bridge from IOxOS has been designed (Belanger & al. 2009).

By allowing high performance multi-core computers to drive directly I/O resources, the PEV1100 changes dramatically the way test benches can be designed. It offers to system engineers a way to upgrade their existing system (legacy VME I/Os) with latest technologies while protecting their investment in terms of software and hardware.

The emergence of PCI Express (PCIe) (PCI Special Interest Group 2007) as a de facto standard in the computer industry makes it the key element of these new architectures. It offers:

- High bandwidth (up to 5 GT/s)
- Backward compatibility with PCI (PCI Special Interest Group 2002)
- Memory mapping (no protocol)

The PEV1100 is a 6U VME board allowing a local host to interface a VME64x bus using a PCIe External cable (PCI Special Interest Group 2006) in order to offer transparent access to I/O boards housed in the VME crate.

Taking benefits from this technology, we decided to design a Linux PC solution with real-time patch and "passive VME" (former CPUs cards racked in the VME crate migrating into the PC cores). Although this seems to be an improvement, it can only be considered as a "*half generation*". Indeed, VME and I/O boards still exist in this configuration. However, solutions based on hybrid architectures CPUs/FPGA allowing complete new generation of test bench architecture exist and are currently under evaluation at Eurocopter.

Basically, the PEV1100 includes in a single board the following features:

- VME64x Master/Slave interface (with slot 1 capability)
- Local Memory
- DMA controller
- Two slots for PMX/XMC mezzanines
- Mailboxes and Interrupt Controller

However, the computing element is now kept outside the backplane in order to let the system integrator free to choose the best solution among those available cores. The complete features and performance considerations are described in (Belanger & al. 2009).

2.3 Helicopter 0: the enhanced integration means

New dimensioning and enhanced integration means are being defined in the scope of Eurocopter "**Helicopter Zero**" project. This initiative aims at adding a ground based engineering test mean in between the classical Systems Integration test benches and the Flight Test program as introduced in the **V** lifecycle illustrated in Fig. 2. The Helicopter Zero will enable testing to be performed at the helicopter system level, approximately one year before the flying of the first prototype, thereby anticipating the detection of design issues and transferring test capability from flight to the ground level. The major helicopter systems will be replicated into two building blocks (a cockpit section and the rear part of the vehicle),

which could be coupled or decoupled depending on the testing strategy.

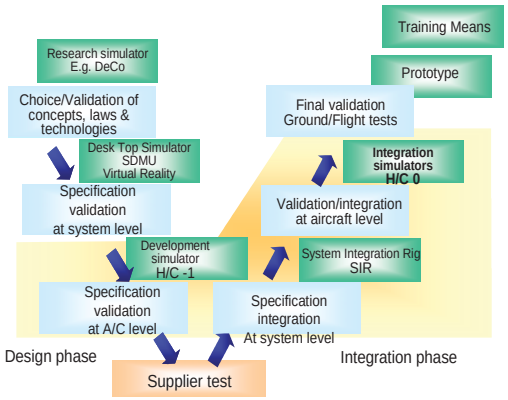

Figure 2: Position of the Helicopter Zero (H/C 0) on the **V** lifecycle.

The Helicopter Zero is basically composed of the two following components:

- **A full size cockpit replica so called Integration COckpit (ICO)** to provide representative cockpit bench. The Integration Cockpit will be composed of representative cockpit frame with slots for real equipment (e.g. Multi-Function Display), equipment racks, and a simulation tool connected to a dedicated reconfigurable system to allow integration test bench parameters settings.
- **A full size helicopter mock-up so called Vehicle Integration Rig (VIR)** to provide representative benches for vehicle system (Electrical & hydraulic), ancillary units, landing gear, and electrical integration. In addition to the corresponding vehicle airborne equipments, the VIR will be composed of a representative industrial structure, industrial motors for power generation, equipment racks, a simulation tool and a reconfigurable system to allow integration test bench parameters settings.

The Helicopter Zero is becoming an essential part of the new V&V approach; it also addresses the convergence of simulation and testing tools. With this new perspective, it is clearly suitable to have a tightly coupled Test and Simulation systems.

3. SIMULATION MEANS AT EUROCOPTER

3.1 RISE presentation

RISE (Real Time Simulation Environment) is a framework aimed at being used to support different kind of simulations within Eurocopter:

1. "Engineering simulation" following the iterative **V** development cycle. It can have different applications depending on the needs which can be:
2. Dedicated workshop for (sub)-systems modeling or prototyping
   - Software benches enabling functional analysis and tests on complex avionic systems
   - Complex environment to test and to demonstrate/validate that the different systems fulfill their requirements and correctly interact in any flight condition
3. "Training simulation":
   - Pre-integration test benches are needed to validate and to investigate on problems reported on the SW data package provided and maintained by Eurocopter to build Full Flight Simulators
   - "Light Training Devices" or "avionic trainer" are an other type of "low cost"/"PC based" products aimed at providing a way to train different parts without needing heavy investment and easily installed/transported in a small area

RISE offers a unified framework to build and execute these various simulation applications. Whatever the scope, a common need to build any of these applications consists in enabling the real-time communication of many processes representing the different parts of the simulated helicopter.

3.2 Dimensioning simulations performed with RISE

Any simulation can be represented according to the following representation (Fig. 3):

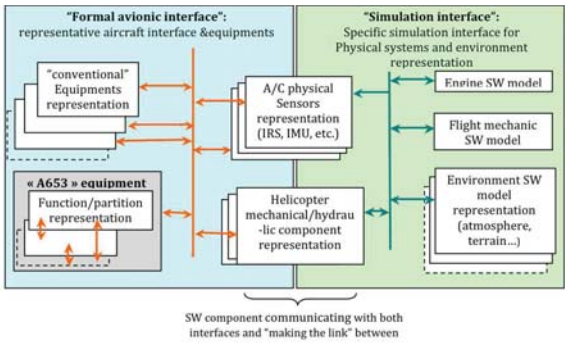

Figure 3: Simulation representation.

- The challenge of most important simulations developed in Eurocopter consists in representing:
- "Formal avionic" and "Simulation" interfaces represented in the simulation as a data pool definition

- Most of the equipments existing in the helicopter as a corresponding model
- Many physical systems as a corresponding model
- For instance, the EC225 pre-integration test bench simulation is composed of:
- 30000 data corresponding to a size of 125000 bytes to represent the complete interface
- 20 models and 40 panels to represent the equipments and the physical systems

In addition to the big amount of data to manage, real-time constraints have to be respected:

- First, the scheduling constraint which represents the frequency imposed to periodically call a model in order to reach the desired period: for instance 10 milliseconds for the models representing the flight loop (Flight mechanic, Engine, Flight control system, mechanical kinematics)
- Second, many models require relatively high performance CPU resource: back to the previous flight loop example, the global execution time of these 4 models during one cycle is around 6 millisecond (on 2,1 GHz CPU) compared to the 10 millisecond authorized.

As consequence from these considerations, complex and complete simulation shall be designed/organized to use multi-core or multi-CPU architectures. RISE is designed to reach this objective with the main principle to avoid any adaptation or constraints on the models aimed at being integrated in the simulation.

That means from the model to be integrated, RISE is in charge of managing for each of them:

- The scheduling (frequency calls and sequence)
- The CPU affinity
- The communication interface between any models (solving the problematic/conflict to establish this communication as well between models having different CPU/core affinity or scheduling constraints)

4. OPTIMIZATION OF MULTICORE ARCHITECTURE FOR SIMULATION

The communication layer ensuring the real-time data exchange between the simulated components aimed at being executed/scheduled in the different CPUs or cores is one of the most important preoccupation of RISE framework.

4.1 RISE architecture principles

Shared memories (shmem) are the backbone of RISE, which uses this implementation to define the data pool corresponding to the interface of the models. It supposes that whatever the model features (CPU affinity, scheduling period), any data representing their interface is translated in a shmem managed by RISE to ensure the real-time data exchange. The difficulty of such architecture is the fact that the different models scheduled in the simulation are supposed to be possibly executed in one of the available CPU/core. As a consequence, conflicts about simultaneous read/write on the same data by two different models may lead to serious/unacceptable incoherencies, and shall be strictly avoided. For that reason a mechanism is implemented in order to ensure that models cannot directly access the shared memory corresponding to their interface data in order to read or write.

4.2 Focus on RISE "Cores Specialization" principles

The scheduler is one of the most important services of RISE, executing the appropriate models through "cyclic entry point (s)" accordingly with the specified characteristics. For each model (e.g. each cyclic entry point) the following scheduling characteristics shall be considered:

- The desired execution **period**, (expected from model supplier)
- The **task**: corresponds to a set of models to be scheduled at the same period, and according to precise sequence between them.
- The desired **sequence** (in the task),

Any models integrated in the simulator belong to a RISE task, which is designed in order to be executed on any available CPU/core. A task is implemented as a thread being controlled by RISE scheduler:

- Coherently with the desired execution period, RISE is in charge of managing each task on a common/unique tick timer, in order to ensure their synchronization.
- Each task can then be set on the desired CPU affinity.

Based on this implementation, model interface management is described accordingly to the following rules:

- **Rule 1:** At the beginning of each task, the models inputs shall be updated according to global simulation interface (shared memory interfaces).
- **Rule 2:** At the end of each task, the global simulation shared memory interface content is updated according to models outputs,
- **Rule 3:** Inside a task, depending on the desired sequence, each model inputs are updated according to the previous models outputs.

The mechanism (implemented within RISE framework, transparently from models) retained to match these 3 constraints, consists in the following operations:

- **Operation 1:** Building "acquisition processes" to be called inside each task, and before any model call,
- **Operation 2:** Building "reconstitution processes" to be called inside each task, and after the calls of the models,
- **Operation 3:** Building an "inter-model" interface for intermediate communication (inside a given task, and between acquisition and reconstitution processes)

The following Fig. 4 illustrates these principles in an example based on few models scheduled at different period on different "tasks":

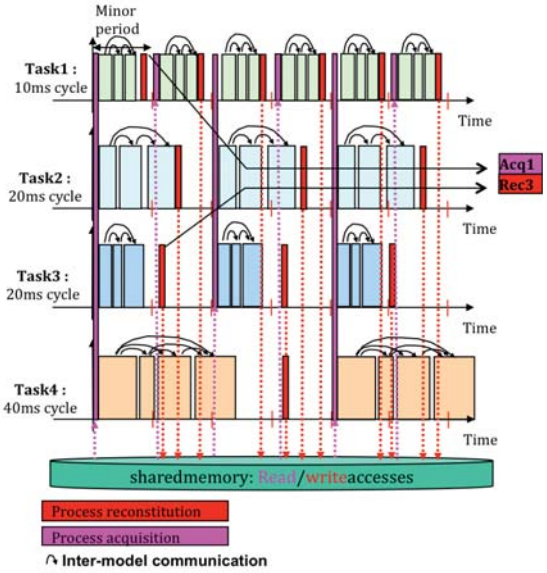

Figure 4: RISE tasks management principles: The start of the "reconstitution" for a given task (ex. Rec3) is supposed to wait the end of "acquisition" of any tasks in progress (ex. Acq1).

4.3 Hardware accesses solutions for simulation systems

Simulation tools like RISE can take benefit from the performances available in the hardware such as the PEV1100. In the same way, we describe below the Arion-I/O module: a single external software agent (middleware) that can provide the wrapping between RISE shared memory and I/O world for hardware integration in the loop functionality. Thus, a first transition between the simulation and test worlds is established.

The purpose of Arion-I/O module is to enable the direct management of inputs and outputs from Arion-Network. The communication area Arion-Network creates a shared memory area for all the equipments; it behaves as a single memory block. Thus, Arion-IO module will connect directly to the communication area and ensure a transparent transposition "Arion Object ⇔ physical I/O" in accordance with the performance and services offered by Arion-Network. This allows any application to access I/O hardware outputs simply by elementary Read/Write commands and share them between different application processes around the system via the bus software.

Therefore, for a virtualized environment like RISE, a single middleware (external agent which links the simulation tool with I/O world) can be used to wrap RISE internal shared memory with Arion-IO Object and so easily perform an hardware in the loop to extend RISE capabilities.

4.4 Towards more and more integration for hardware in the loop solutions

Based on the foregoing, we can now go further and design a test/simulation system based on a PCIe hierarchy with multiple multi-core computers and multiple FPGAs embedding several avionic I/O IP cores. We have seen in the previous sections that various middlewares can be considered to wrap a simulation tool like RISE to I/O world to address hardware in the loop : from Data Distribution Service (DDS) such as Arion-Network to a simple external agent that can translate the memory mapping between RISE shared memory and I/O resources through a PCIe using a transparent mode. In addition, the availability of a user area in the various PEV1100's FPGA allows us to consider relocating some computing closer to the I/Os in a hardware fashion. In fact, FPGA technology can execute behavioural models faster than CPUs and GPUs.

5. HYBRID CPU/FPGA ARCHITECTURE

A scalable heterogeneous CPU/FPGA architecture has been implemented in the frame of CHARTS project (Codesign modelling and High-performance Architecture for Test & Simulation (Afonso & al. 2011; Belanger and Afonso 2011)) composed mainly of two nodes. The first node is a general-purpose multi-core processor (i.e.: AMD/Intel) while the second node represents a cluster of FPGAs. The multi-core will offer performance with a limited parallelism capability due to the fixed number of cores. FPGAs are the support of the reconfigurable logics needed to implement challenging tasks as hardware accelerators. The heterogeneous mapping of software and hardware tasks onto several CPU cores and FPGA resources, needs appropriate and user-friendly mechanisms for task allocation, sharing data, communication, and synchronization. In the next subsections, we will focus first on the programming of the multi-core system and secondly, we will detail the CPU-FPGA coupling.

CPU task allocation: Nowadays Operating Systems (OS) such as Linux allocate dynamically tasks onto the available cores which may introduce latencies and make fail the system regarding the time constraint. This is due

to the fact that general purpose OS do not support real-time functionalities and allocate priorities on each executed tasks. Processor affinity service is a modification of the native central queue scheduling algorithm in a symmetric multiprocessing (SMP) operating system. Each task (process or thread) in the queue has a tag containing the target processor or core number on which it will be executed. In our architecture, we propose to allocate each kind of tasks (Operating System, User Software applications, etc) in the available cores under bounded soft real-time constraints.

Fig. 5 shows an example of task allocation; core 1 runs the OS while the user applications are mapped on the core 3 and 4. In order to obtain better performances, we start our OS on the first core (Listing 1, Line 3) and we stopped all OS interrupts to avoid undesirable latencies for the cores 3 and 4. To do so, we stopped first the *irqbalance* Linux service as illustrated on Listing 1, Line 8 and after that we carry out the user applications on the corresponding cores (Listing 1 Lines 10 and 11). The task mapping using Processor affinity OS service can be done dynamically or on the fly depending on the user preferences and the available cores.

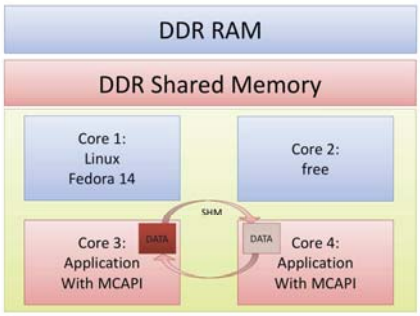

Figure 5: Communication toward shared memory in multi-core architecture.

```
1 /*To start our Operating System on core 1*/
2  /*add this line in file "grub.conf"*/
3  isolcpus=1
4  /*stop irqbalance service, to avoid having distributed interrupt on the cores (2, 3 and 4) we need free from Linux interrupts*/
5  service irqbalance stop
6  /*Launch program in core 3*/
7 taskset c 3 ./application1
8  taskset c 4 ./application2
```

Listing 1. Pseudo-code of task allocation.

CPU-CPU communication: In the standard Linux OS, several services (e.g. socket) offer communication capabilities between different processing units for multi-core architecture. However, these services cannot deliver interesting communication time. For this reason, several frameworks have been proposed in the context of fast, reliable, and efficient inter-core communication. Among these frameworks, the Multi-Core Association proposes the Multi-core Communication API (MCAPI), which relies on a message-passing mechanism in order to capture the basic elements of communication and synchronization that are required for closely distributed complex systems (multiple cores on a chip and/or chips on a board). The target systems using such API will cover multiple dimensions of heterogeneity (e.g. core heterogeneity, interconnect heterogeneity, memory heterogeneity, and programming language heterogeneity). Mentor Graphics provides an open source implementation of the MCAPI standard called OpenMCAPI.

The corresponding solution is based on a Shared Memory (SHM) layer for all cores. A provided Linux driver offers access to this memory and an implementation of interrupt handler for core synchronization and user front-end interface. The main advantage of using this driver is the transparency according to the cores running the same Linux OS.

Listing 2 shows an example of an application using OpenMCAPI. We start first with the SHM allocation as illustrated in Listing 2 Line 2, the memory size is specified by the user. Lines 5 and 9 show respectively the user interfaces for sending and receiving data to/from another core. Using processor affinity service for allocating tasks for core 3 and 4, the test scenario shown in Fig. 5 can be realized thanks to OpenMCAPI interfaces.

```
1    /*SHM allocation*/
2    memset(outgoing, 0, 16);
3    sprintf(outgoing, "hi from node %d", node);
4    /*sending data with MCAPI routine*/
5    mcapi_pktchan_send(send_handle, outgoing,
6    strlen(outgoing)+1, &status);
7    mcapi_assert_success(status);
8    /*waiting data from another core*/
9    mcapi_pktchan_recv(recv_handle,
10   (void *)&incoming, &Bytes, &status);
11   mcapi_pktchan_free(incoming, &status);
12   mcapi_assert_success(status);
```

Listing 2. Pseudo-code of communication between two cores.

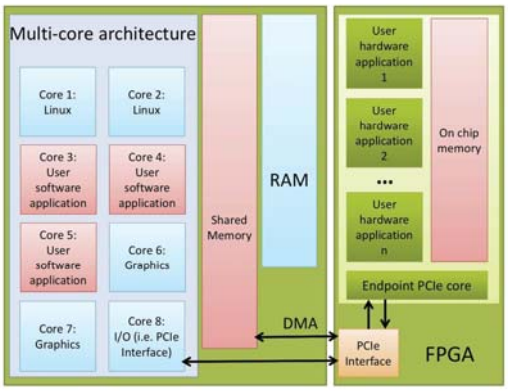

Figure 6: Main parts of the heterogeneous CPU/FPGA system.

The CPU-FPGA programming: The second node of our heterogeneous architecture is composed of one or several FPGAs. Indeed, FPGA technology could implement heavy applications in a hardware fashion with the management of the parallelism degree to satisfy performance and real-time constraints of the application. It becomes also possible to reduce the communication latencies by the means of embedding different parts in the same chip: user applications, data and instruction memory, and I/O as shown in Fig. 5. The mapping of user applications on the FPGA must take into consideration the shared data between models. Indeed, to increase performance, we need to reduce the communication and the access memory latencies. Furthermore, each FPGA node can embed hardware and software models using VHDL code and soft-cores such as the Xilinx Microblaze which has the ability to use a dedicated link such as the Fast Simplex Link (FSL) bus interface.

CPU-FPGA communication: We need to communicate CPU and FPGA nodes efficiently through a fast link. Today, almost host machines or workstations are equipped with PCIe slots for expansion boards. In addition, a large range of commercial FPGAs integrate a hard endpoint PCIe core for industrial usage. Our proposal is to make profit from these features in order to design an efficient solution that can deal with the interoperability between hardware and software applications mapped respectively on FPGA and CPU nodes with high throughput. Fig. 6 describes the communication between the different nodes using the standard PCIe communication bus. Through point-to-point or shared memory, communication between hardware and software applications can be established. The PCIe driver provides the user interfaces for sending and receiving data to/from FPGA or CPU. Listing 3 shows a simple write and read communication between the CPU and the FPGA using memory allocation. It performs a data round-trip between these two nodes.

In such architecture, communication latency with respect to the real time constraints is considered the most important metric. To avoid intolerable communication latencies, we can rely on two solutions. The first one uses the processor affinity OS service in order to allocate a specific core for the CPU/FPGA communication as mentioned in Fig. 6 for the core 8. The second solution is to enhance our operating system with real-time functionalities. The different solutions will be implemented and discussed in the case study section.

```
1   /*using device USERPCIe*/
2   char* devfilename = USERPCIe;
3   /*Opening device USERPCIe*/
4   g_devFile = open(devfilename, O_RDWR);
5   /*Memory allocation*/
6   gReadData = (TransferData*)
malloc(sizeof(struct TransferData));
7   gWriteData = (TransferData*)
malloc(sizeof(struct TransferData));
8   /*Writing and reading data from FPGA*/
9   WriteData((char*) gWriteData, 100);
10  ReadData((char *) gReadData, 100);
```

Listing 3. Pseudo-code of CPU-FPGA communication.

Data sharing: In order to share the data between different cores and the FPGA, we implemented thanks to OpenMCAPI, an available DDR SHM for all computing nodes, even the FPGA resources. In fact, Xilinx proposes a Direct Memory Access (DMA) IP-core for PCIe. The DMA allows the FPGA node to access the host shared memory for reading and/or writing independently of the central processing unit. The Multi-Core Association proposes the Multi-core Resource Management API (MRAPI) that captures the essential capabilities required for managing shared resources in a closely distributed system (multiple cores on a chip and/or chips on a board). These resources include multiple cores or chips, hardware accelerators, memory regions, and I/O. MRAPI supports the ability to declare and allocate/destroy shared memory (SHM) regions and to identify nodes which have access to each region.

CPU-FPGA synchronization: In order to enforce the data dependencies between the software and hardware tasks or applications, synchronization must take place. The synchronization mechanism must allow hardware task to wait for a data being produced by a software task

and vice-versa. In our environment, the synchronization between the CPU and the FPGA is performed by the PCIe Message Signaled Interrupts (MSI). MSI allows the device to write a small amount of data to a special address in memory space. The chipset will deliver the corresponding interrupt to a CPU. While conventional PCI was limited to 4 interrupts per card, MSI allows dozens of interrupts per card. This is very useful in our case, because the reconfigurable device can host multiple hardware accelerated tasks. Status and priority information of each accelerator are stored inside the interrupt handler allowing the arbitration and prioritizing concurrent interrupts.

6. CASE STUDY

In this section, we will introduce first the experimental environment setup composed of a standard host machine and a Virtex-6 Xilinx board. Second, different CPU/FPGA communication solutions will be compared in terms of reliability, performance, and software development effort.

Experimental environment setup: Our prototype environment consists of an AMD Athlon 2.5~GHz X2 Dual Core Processor 4800+, 2-GByte DDR memory, and the ML605 Virtex-6 Xilinx board. This board was easily plugged onto the host motherboard through a PCIe 8x slot that can support 2.5 GBytes/sec throughput. The system runs with Linux kernel 2.6.35.1. However, our approach is generic and can be used with different computer system configurations or Linux kernel versions.

6.1 Processor affinities for soft real-time constraints respect.

As our objective is to cover a large spectrum of applications, we enhanced our prototype environment with real time functionalities in order to satisfy system with hard timing requirements. This aspect is quite pertinent for industrial application domain. For this reason, the real-time framework Xenomai for Linux is used in our environment. Xenomai 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, useful 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 approach benefit 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 useful 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 facilities existing in the user space (e.g. full POSIX support, including inter-process communication).

For all experiments, we measured the data-round trip time according to the data size varied from 4 to 1024 Bytes. Fig. 7 illustrates the average communication time for each configuration. Configuration C presents the minimum average time independently of the data size thanks to the inserted real-time communication routine. Configuration B offers comparable performances to those obtained with C. This is due mainly to the mapping of the communication routine on a specific core and avoiding OS interrupts for it. Configuration A offers the highest average time due the latencies introduced by a standard Linux OS.

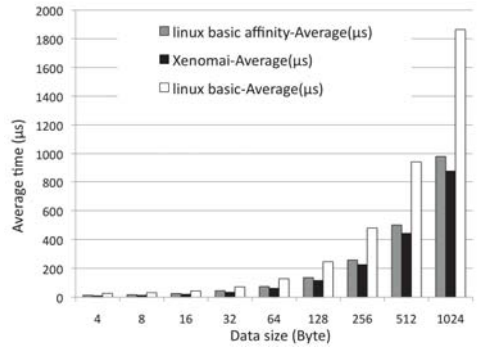

Figure 7: Average time for 4 - 1024 Byte data round-trip.

Fig. 8 shows the maximum communication latencies captured from each configuration for different data sizes. This parameter is very important for applications bounded under soft or hard real-time constraints. Configuration A suffers from significant communication latency overflow whereas configurations B and C offer similar and acceptable latencies.

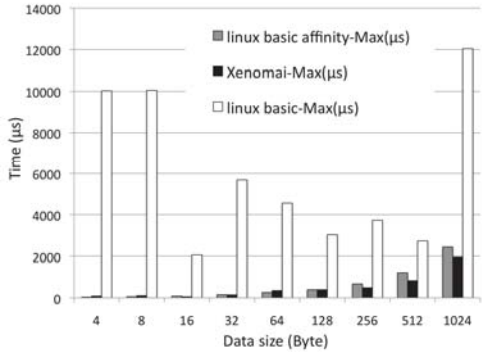

Figure 8: The maximum of latencies for 4 to 1024 Bytes data round-trip.

In order to measure the variation of the communication from the average latency, we compute the standard deviation for each configuration. This parameter reflects the reliability of the communication solution. A low standard deviation indicates that the data points tend to be very close to the average, whereas high standard deviation indicates that the data are spread out over a large range of values. Fig. 9 shows the measured deviation for different data sizes varied from 4 to 1024 Bytes.

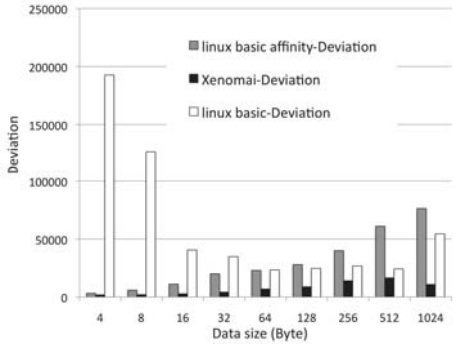

Figure 9: Standard deviation for 4 to 1024 Bytes data round-trip.

Several remarks can be drawn. First, the configuration A presents a very high deviation only for small data sizes. However, the deviation for the configuration B increases gradually according to the data size. The configuration C offers a low deviation with a minor correlation with the data size, thus it presents the most reliable communication solution. In conclusion, Xenomai and Linux with processor affinities offer similar performance and acceptable latencies. In the case of hard real-time requirements, a real-time kernel or patch is essential. Nevertheless, for soft real-time constraints, processor affinity-based solution can be enough to provide acceptable performance. In term of software development effort, Xenomai-based solution requires a code transformation (specific libraries (VxWorks, PSOS, etc) use) which increase the software design time. With processor affinity-based solution, only a command line is needed for the appropriate core setting.

Finally, we calculated the experimental throughput for the different configurations. Fig. 10 depicts the results. First, larger data transfers lead to higher throughputs. This is because of the PCIe protocol efficiency which increases according to the payload length. However, increasing the data size over a certain limit (1024 Bytes) tends to be ineffective, as the maximum of the PCIe bus capacity is reached. Second, configurations A, B, and C lead respectively to 1.1 GBytes/s, 2.1 GBytes/s, and 2.35 GBytes/s throughputs. Operating system supporting real-time functionalities achieves a communication throughput (2.35 GBytes/s) roughly equal to the theoretical maximum (2.5 GBytes/s). Configuration B based on processor affinity has much better performances comparing to the configuration A which presents a throughput (1.1 GBytes/s) significantly lower than the theoretical maximum.

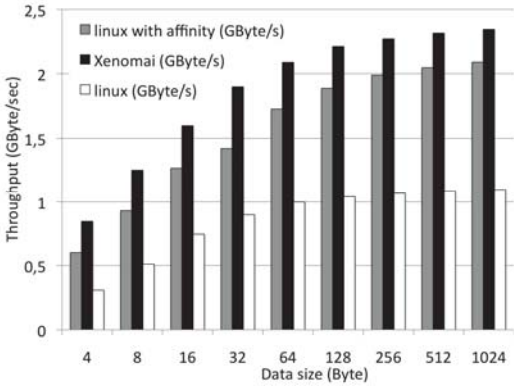

Figure 10: PCIe x8 throughput for data size between 4 to 1024 Bytes.

6.2 FPGA technology for model execution

In this section, different implementations of a software model (called “A”) are explored. We consider first a software implementation of the “A” algorithm executed on an AMD Athlon X2 Dual Core Processor 4800+ with a Linux environment. Second, the same algorithm can be implemented using several hardware accelerators on the FPGA. With the management of the parallelism intrinsic in the application, several trade-offs between execution time and area utilization can be obtained. To do so, the vfEmbedded (Minjang, Hyesoon and Chi-Keung 2010) tool analyses partitions and maps applications on specific platforms as heterogeneous ones. It can also estimate the performance of the parallelized software before implementing it. Moreover, it can trim any overhead in your hardware to reduce cost and ensure that all critical behaviors in your program are exercised. The ”A” main loop fits perfectly to parallel transformation and can be implemented in a FPGA hardware fashion in order to improve its performance execution.

To make this step more efficient, tools such as Riverside Optimizing Compiler for Configurable Computing (ROCCC) (Villarreal, Park and Robert 2010) can focus on FPGA-based code acceleration from a subset of the C language. ROCCC does not focus on the generation of arbitrary hardware circuits. Its objectives are to maximize parallelism within the constraints of the target device, optimize clock cycle time by efficient pipelining, and minimize the area utilized. It uses extensive and unique loop analysis techniques to increase the reuse of data fetched from off-chip memory.

So, with the help of the ROCCC tool, a compilation step from C to VHDL is performed. Furthermore, we make profit from the loop unrolling feature provided by ROCCC in order to obtain varied implementations for the "A" main loop. We synthesized 17 samples with varying the parallelism degree. Implementation results are reported in the Fig. 11. We obtained an execution time varying from 19 to 14us respectively with an area utilization varying from 5,800 to 14,300 slices. Indeed, more we parallelize the "A" main loop, the number of occupied slices increases and the execution time is reduced.

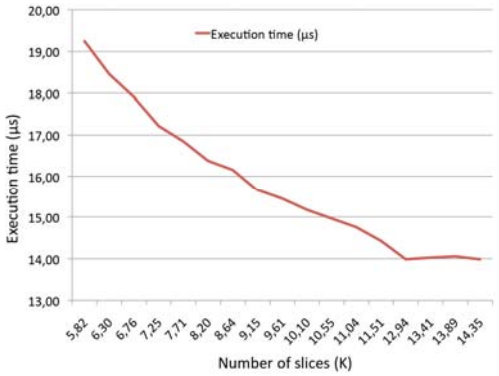

Figure 11: Exploration of different hardware implementations of the model A.

The software implementation on the host offers an execution time equals to 18us with unexpected latency of 35ms introduced by the Linux environment which is unacceptable for applications bounded under real-time constraints. The selection of the best implementation will depend on the global system constraints, the data mapping, and the available resources.

7. CONCLUSION

We have introduced in this paper, the current and next future integration test benches architecture at Eurocopter. We insisted on the key period that we are facing in terms of V&V means transformation. To illustrate the current convergence of simulation and test tools, we presented multipurpose simulation tool RISE with focus on its multicore optimization principles. Those principles bring real added value and key performance to integration and management of models.

Regarding RISE multicore optimization principles and capabilities offered by FPGA in terms of computing, I/O access and reconfiguration we imagined the new hybrid CPU/FPGA architecture CHARTS on which Test & Simulation tools can both rely. Beyond the I/O access through IP Cores, CHARTS also introduces a new capability for models management in terms of their optimized targeted hardware architecture. It demonstrates hybrid architectures CPUs/FPGAs can bring new capabilities in the direction of Test and Simulation tools convergence and enhance models management at HW/SW co-design time. CHARTS architecture is detailed in terms of internal communication between components. The presented case study and associated results validates the use of Xenomai for hard real-time constraint respect and processor affinity use for soft real time constraints as simulation purpose.

8. REFERENCES

Afonso G. & al., 2011, "Toward Generic and Adaptive Avionic Test Systems", *Proceedings of NASA/ESA AHS, IEEE Conference*, San Diego, USA.

Belanger N. and Afonso G., 2011, "The CHARTS Project: an innovation matrix", *Proceedings of SAE AeroTech Industrial Congress & Exhibition*, Toulouse. France.

Belanger N. & al., 2009, "Multi-Core computers and PCI Express : The future of data acquisition and control systems", Proceedings of European Test & Telemetry Conference, Toulouse, France

Minjang K., Hyesoon K. and Chi-Keung L., 2010, "A scalable approach to dynamic data-depedence profiling," in Proceedings of 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)

PCI Special Interest Group, 2002, "PCI Local Bus Specification 3.0", available from: http://www.pcisig.com/specifications/conventional [Accessed 29 02 2012]

PCI Special Interest Group, 2007, "PCIe Base 2.0 Specification", available from: http://www.pcisig.com/specifications/pciexpress/base2/ [Accessed 29 02 2012]

PCI Special Interest Group, 2006, "PCIe External Cabling Specification 1.0", available from: http://www.pcisig.com/specifications/pciexpress/pcie_cabling1.0/ [Accessed 29 02 2012]

Peckham C., 2006, "VMEbus at 25", RTC Magazine

Villarreal N.W., Park A. and Robert H., 2010, "Designing Modular Hardware Accelerators in C with ROCCC 2.0," in Proceedings of the 18th IEEE Annual International Symposium on Field-Programmable Custom Computing