2013 IEEE 27th International Symposium on Parallel & Distributed Processing Workshops and PhD Forum

HETEROGENEOUS CPU/FPGA RECONFIGURABLE COMPUTING SYSTEM FOR AVIONIC TEST APPLICATION

George Afonso[1], *Zeineb Baklouti*[2], *David Duvivier*[2], *Rabie Ben Atitallah*[2], *Eli Billauer*[3], Stephan Stilkerich[4]

[1]EADS IW, INRIA Lille Nord Europe
[2]LAMIH, University of Valenciennes, INRIA Lille Nord Europe
[3]Xillybus company
[4]EADS IW, Germany

ABSTRACT

Real-time computing systems are increasingly used in aerospace and avionic industries. In the face of power wall and real-time requirements, hardware designers are directed towards reconfigurable computing with the usage of heterogeneous CPU/FPGA systems. However, there is a lack of real-time environments able to deal with the execution of applications on such heterogeneous systems dedicated to avionic Test and Simulation (T&S). This research investigates the problem of soft real-time environments for CPU/FPGA systems and proposes first a high-performance hardware architecture used to implement intimately coupled hardware and software avionic models. Second, this paper presents the description of an efficient real-time software environment for the model's execution, the multi-core CPU monitoring and the runtime task re-allocation to avoid the timing constraint violation. Experimental results underpin the industrial relevance of the presented approach for avionic T&S systems with real-time support.

1. INTRODUCTION

In the manufacturing process, the Test and Simulation (T&S) phases have always been considered as an essential part in the avionic development cycle. In a reasonable time, system designers have to check the functioning of safety-critical avionic units embedded in modern helicopters. Indeed, these systems often operate in uncertain conditions and they must provide safety, fault tolerance, and deterministic timing guarantees.

Hitherto, T&S computational requirements were met by real-time computing systems. Generally, these systems are based on specific CPU (Central Processing Unit) boards that run proprietary real-time operating systems and plugged with Input/Output (I/O) boards to communicate with the equipments under-test. In current industrial practice, the well-spread VME CPU boards are widely used [1]. Due to the present T&S system performance requirement, an increase in the computation rates is needed, but it cannot be delivered by the VME CPU boards any-more. Furthermore, this solution is considered as an expensive maintainable technology. In order to bring reliability and competitiveness to the avionic industry, hardware designers are exploring new alternatives for providing performance in future real-time computing systems.

In order to meet real-time, power, and flexibility goals, combination of general CPU and reconfigurable fabrics like Field-Programmable Gate Arrays (FPGAs) is a promising solution leading to heterogeneous computing. In such systems, multi-core CPU provides high computation rates while the reconfigurable logic offers high performance per watt and adaptability to the application constraints. Designers could exploit the existing partitioning in the application which leads to several feasible implementations with different performances. Due to the high parallelism rate of the application, FPGA technology could offer better performances comparing to CPUs or GPUs up to 10x [2] at lower frequencies. Using heterogeneous CPU/FPGA systems allows to adapt the architecture according to the application constraints and thus to optimize hardware resources. Such architectures can be also customized at run-time using Dynamic Partial Reconfiguration (DPR) feature offered by recent FPGA technologies. The ITRS [1] and HiPEAC [2] roadmaps promote that heterogeneous and adaptive architectures will dominate next generation computing systems. All these benefits emphasize hardware designers to redirect their efforts on reconfigurable computing domain.

We are in line with this vision, but we have to overcome several obstacles linked with real-time requirements of our application domain. Indeed, respecting the soft real-time constraints is the main challenge for our avionic T&S applications. Significant efforts should be devoted at the archi-

[1]http://public.itrs.net/
[2]http://www.hipeac.net/system/files/hipeacvision.pdf

978-0-7695-4979-8/13 $26.00 © 2013 IEEE
DOI 10.1109/IPDPSW.2013.111

tectural level in order to implement intimately coupled hardware and software avionic models with very low communication latency. Furthermore, the host software execution environment should guarantee the user timing constraints and avoid the failure of the T&S phase.

To address the above mentioned challenges, we established a generic environment based on a hardware architecture composed of tightly-coupled multi-core CPUs and Xilinx Virtex-6 FPGA using the PCI Express (PCIe) communication standard. Within our environment, a great care has been devoted to the real-time aspect in order to satisfy tight computing and communication deadlines. To do so, a specific front-end interface have been used for core-to-core and core-to-FPGA communication. After that, at the software level, we go farther into the support of the monitoring and the supervision functionalities in our environment in order to meet soft real-time requirements. Indeed, without an efficient monitoring implementation, a task scheduler will not be able to recover all necessary informations to detect a possible failure of the T&S session. Furthermore, we implement a supervision strategy that allows run-time model reallocation in order to anticipate the onset of overloads. In this work, we exploit the available hardware resources for switching tasks dynamically between the multi-core CPU and the FPGA.

This paper is organized as follows. After this first section, Section 2 presents the related work, Section 3 describes the reconfigurable heterogeneous architecture for avionic T&S system. As a part of our system, we will present the Xillybus solution making CPU/FPGA communication through the PCIe bus easier. Section 4 presents our soft real-time environment based on a CPU/FPGA architecture. Experimental results are provided in Section 5.

2. RELATED WORKS

Recent predictions announced by the ITRS and HiPEAC roadmaps show that high performance and embedded computing will rely more and more on heterogeneous architectures. Almost of existing commercial solutions target specific application domains with the usage of special processing architectures (DSP, GPU, etc.) well-tuned according to the algorithm instruction set. Though such architectures lead to better performances, they are still not appropriate and flexible to address general purpose or high performance computing. In an attempt to deal with this challenge, a lot of industrial and academic research efforts have been put to provide more flexibility and adaptability to the system using reconfigurable fabrics. As examples of commercial systems, we quote the ClearSpeed [3] and FARM [4]. However, these machines are not accessible for general usage due to their expensive prices and complexity of programming. For this reason, some academic projects proposed to establish prototype environments for CPU/FPGA heterogeneous systems. They make profit from the huge logic fabrics provided by nowadays FPGA boards that can be connected easily to the host machines using fast links. Among these projects, Brandon et al. [5] proposed a generic platform solution for reconfigurable acceleration in a general-purpose system. They focussed also on techniques to more efficiently integrate a reconfigurable device in a traditional computer system, addressing issues related to the memory accesses, programming interface, and system-level support for high performance computing, not for real-time requirement.

Thanks to their pure hardware architecture, the FPGA technology provide hard real-time capabilities. Indeed, real-time architecture such as Bluehive [6] are pure FPGA based. Real-time architectures based on SoC such as the Xilinx Zynq 7000 are also possible using real-time Operating Systems [3] accessing to the reconfigurable zone from the CPUs available. Moreover, at a pure software level, recent works on Open Source real-time systems offers lots of different solutions. As an example, FreeRTOS[4] supports about thirty different architectures with an active community. Xenomai[5] proposes also an interesting solution based on Linux Kernel patching and provides generic building blocks for implementing real-time APIs (such as VxWors, pSOS, etc) called "skins". These skins allow software designers to use their knowledge on from proprietary real time operating systems or reuse some parts of existing software. In these previous examples, there are an important training and a maintenance cost in order to develop sustainable designs such as avionic test systems on these environments. Our solution described in this paper will offer soft real-time capabilities for CPU/FPGA architectures overcoming the constraints mentioned above.

3. CPU/FPGA AVIONIC T&S SYSTEM

Avionic T&S application domain targets the validation of avionic embedded systems before first test flights in order to reduce the time to market. This phase is critical and has to respect constraints in order to provide the duplication of real flight conditions. In order to perform a complete T&S session, we need to modelize each part of the helicopter and the environmental parameters (weather conditions, geographical factors, etc.). Fig. 1 presents a simplified T&S loop that simulates the helicopter behaviour including three models: the *flight mechanic model*, the *navigation model* and the *automatic pilot model*. Depending on the T&S scenario, each software model such as the *automatic pilot* can

[3] http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/operating-systems/index.htm

[4] http://www.freertos.org/

[5] http://www.xenomai.org/

be replaced by the corresponding real hardware which calls for additional I/O communication capability.

From the T&S architecture perspective, real-time constraint respect and performance requirements are necessary. To address these application needs, we propose the usage of a heterogeneous CPU/FPGA system composed mainly of two nodes as shown in Fig. 2. The first node is a general purpose multi-core processor (i.e.: AMD/Intel) while the second node represents a reconfigurable FPGA. 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. For example, the *flight mechanic model* can receive the flight control from a simulated model or from an I/O avionic IP located in the FPGA in the case of using a real *automatic pilot* system in the loop. These elements are essentials for the configuration of each T&S scenario depending on the timing constraints and taking into consideration the communication overheads. Thus, it becomes necessary to design different versions of T&S models (hardware or software) which will be implemented depending on the scenario.

The heterogeneous mapping of software and hardware tasks onto several CPU cores and FPGA resources, as shown in Fig. 2, needs appropriate and easy to use mechanisms for communication, task mapping, sharing data, and synchronization. In the next subsections, we will focus first on the communication of the multi-core system and secondly, we will detail the CPU/FPGA coupling.

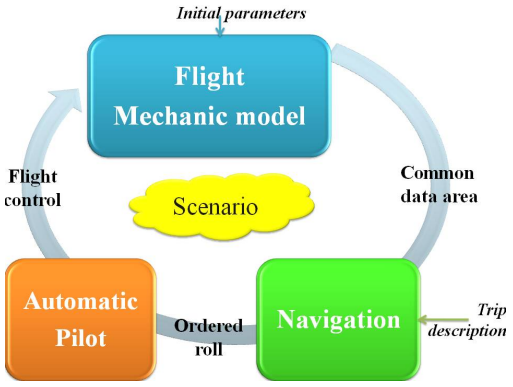

Figure 1: A simplified T&S loop

3.1. The 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)[6], 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. The target systems using such API will cover multiple dimensions of heterogeneity (e.g. core, interconnect, memory, and programming language heterogeneity). Mentor Graphics provides an open source implementation of the MCAPI standard called OpenMCAPI[7]. The corresponding solution is based on a Shared Memory (SHM) layer for all cores. A provided Linux driver offers access to a user customized size memory, an implementation of interrupt handler for core synchronisation, and a user front-end interface. More details about the settings of the core-to-core communication is described in [7].

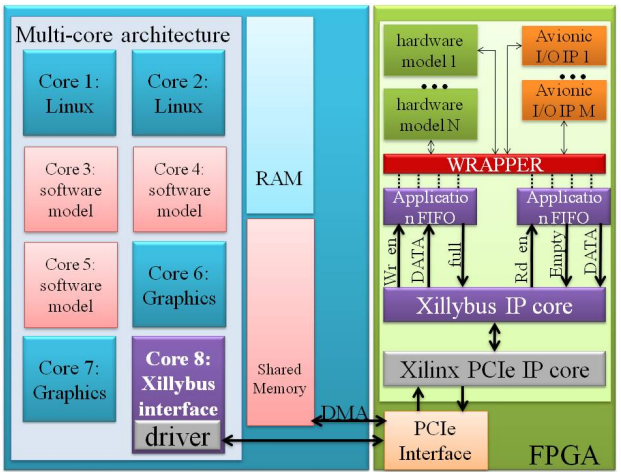

Figure 2: The CPU/FPGA T&S system

As well as we need to optimise our avionic models in order to obtain the better performances, we need also to focus on the communication which is crucial in heterogeneous architectures. The link has to be fast, efficient, and widely used in industrial systems. Nowadays, 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 models mapped respectively on FPGA and CPU nodes with high throughput. In such architecture, communication latency with respect to the real-time constraints is considered the most important metric. Nevertheless, it is first necessary to define the application requirements in order to propose a customised solution that offers the better trade-off between

[6] http://www.multicoreassociation.org/workgroup/comapi.php
[7] https://bitbucket.org/hollisb/openmcapi/wiki/Home

the communication bandwidth and the design cost.

3.2. Xillybus : making FPGAs talk PCIE easier

Due to the PCIe bus complexity, the communication in a heterogeneous architecture remains complex. Most of the time, all PCIe capabilities are not even required (i.e. prototyping), an abstracted communication level would improve the design cycle. Xillybus[8] proposes a simple interface for the FPGA and the application designer: The FPGA application logic connects to the IP core through standard FIFOs (for read and write), and the user application on the host machine (Microsoft Windows or Linux) performs plain file I/O operations on pipe-like device files.

Streaming data move naturally between the FIFO and the file handler opened by the host application. There is no specific and intrusive API involved, allowing the hardware and software designers to focus on the requirements of their application. This setting relieves the FPGA designer completely from managing the data traffic with the host. Rather, the Xillybus core checks the FIFOs "empty" or "full" signals (depending on data direction), and initiates data transfers when the FIFO is ready for it. As the number of streams and their attributes are configurable, this solution scales easily as the design requirements expand.

Fig. 2 depicts a simplified block diagram showing the connection of one data stream in each direction. The application on the computer interacts with device files that behave like named pipes. The Xillybus IP core and driver on the host offer efficient data streaming (using DMA) between the FIFOs in the FPGAs and their respective device files on the host. The Xillybus IP core implements the data flow utilizing PCIe transport layer level, generating and receiving Transaction Layer Packets (TLPs). For the lower layers, it relies on Xilinx official PCIe core, which is part of the development tools. Making the communication simple is sometimes not enough, the PCIe bus offers several optimization functionalities with high throughput. The goal is to find the best trade-off between simplicity, reliability, design time and performance in order to address all requirements of our application.

4. REAL-TIME SOFTWARE ENVIRONMENT FOR AVIONIC TEST SYSTEMS

After focusing on the hardware aspects of the avionic T&S system, we will detail in this section the software environment in order to meet robustness in terms of real-time constraints respect and performance. Let us highlight that in avionic test applications we consider soft real-time requirement that means response times are in the order of milliseconds. Fig. 3 presents our solution composed mainly of the T&S project, the monitor, and the supervisor. The next subsections will detail our environment.

[8] http://xillybus.com/

4.1. The T&S project

A T&S project (P1, P2 or Pn in Fig. 3) contains several avionic models as explained in the previous section. Each model has its own specifications (input data, output data, required execution time, etc.) and different versions (i.e. software, hardware, hybrid software and hardware) providing the best performances depending on the scenario as stated before.

Based on a discrete event simulation, each T&S project is executed periodically respecting a timing constraint defined by the user in the order of milliseconds. To do so, a timer periodically awakes each task (see if defined, as/by a graph of models linked by precedence constraints) using a predefined fixed period. In a standard Linux configuration, the response time of any task cannot be bounded while using the default OS preemptive scheduler due to several parameters such as the global system load, the scheduler frequency, the priority of the different tasks, etc. For this reason, we proposed the usage of the processor affinity OS service in order to execute the tasks with soft real-time requirements. Thus, each task (software avionic model) in the queue has a tag containing the target processor or core number on which it will be executed. In addition, we stopped all the OS interrupts on these cores to avoid undesirable latencies. Furthermore, to avoid the timing constraint violation, we modified the host standard kernel configuration by disabling the following services: the swap capability in order to be sure that any model will not be "swapped", the *CPU Frequency Scaling* to keep the cores at their maximum frequency and the OS services linked with the paravirtualisation or the virtualisation. In order to refresh the processor's cores load at a higher frequency, we modify the CONFIG_HZ parameter in the kernel configuration environment, increasing from 250 Hz to 1000 Hz the Linux scheduler refresh frequency.

This architecture supports multi projects in order to offer hardware sharing capabilities. The specification of the models and their allocation (CPU or FPGA) are contained in the configuration file (Config.dat in Fig. 3). The T&S project starts by the *launcher*. It is a key component in our architecture. It reads the models' parameters (mapping, precedences...) from the configuration file and store them in a data-structure in memory. Then according to this structure, it starts the project: it allocates the models to the corresponding computation node and establishes the inter-models communications and synchronizations links. It finishes its jobs by establishing similar links between the tasks and the supervisor.

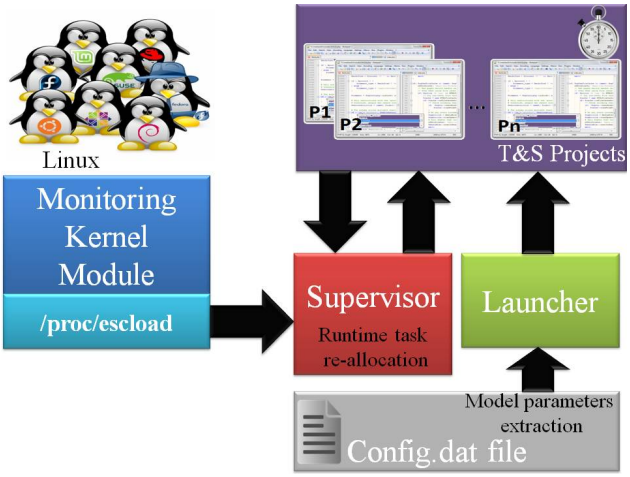

Figure 3: The CPU/FPGA supervising environment

4.2. The monitor module

In our environment, the *monitor* is defined as a standalone kernel module in order to achieve very fast access to the internal structure of the OS kernel. It takes in charge to compute the load of the different CPU cores every millisecond and to store these informations in an internal data-structure. These informations are available via an entry, named ”escload”, in the pseudo file system /proc. This internal structure is protected against ”race conditions” so as to ensure a decoupling between internal refresh rate of the loads and external reading of the loads via /proc/escload. Thus, the supervisor can safely access to these informations by reading /proc/escload. Our solution is generic allowing the implementation of different heuristics depending on the requirements of the application in different standard Linux Operating Systems. In our case, we need only the cores load and the models execution time as parameters, there is no dynamic hardware supervising, the used area being constant after all IP core load in the components.

4.3. The supervisor

The *supervisor* component contains mainly an efficient Longest Processing Time (LPT) Heuristic [8] that maps the T&S project models onto the CPU/FPGA architecture. Indeed, at the initialization phase, it will choose the optimal avionic models mapping depending on the scenario and the current host load. This process has a role similar to the launcher in the initialization step: once started, it creates a data structure that contains the project parameters collected from the configuration file. Then its specific role is to wait for any overload and to take the appropriate decision in order to re-allocate the T&S project or to stop the simulation.

When an overload of 70% occurs, the supervisor is alerted by the T&S project using a signal. To start the heuristic mapping, the supervisor needs 2 parameters. The first is the current load of the different cores which is calculated by the monitor kernel module. This information can be loaded while reading from the pseudo file system /proc. Thus, the supervisor can access to these information by reading /proc/escload. The second parameter is the software model execution time that will be send through a named pipe. For the FPGA part, there is no dynamic hardware supervising, the used area being constant after all IP core load in the circuit.

Having all the required parameters, the supervisor will start the LPT Heuristic in order to choose the optimal mapping of models on the architecture. Therefore, only the structure and the named pipe mask value will be updated; the ”Config.dat” file will not be changed and hence a very fast task migration is obtained. After being alerted from the supervisor, the model will search its new allocation by reading the mask named pipe and migrate to the selected core. In the case of reaching 90% of load, the T&S project alerts the supervisor and stops the simulation.

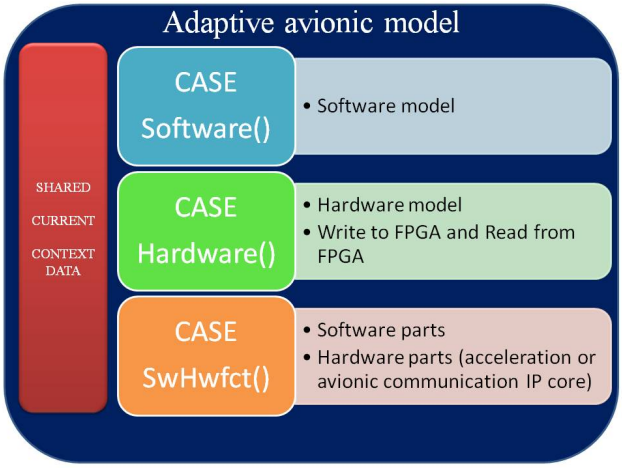

Figure 4: Adaptive avionic model

Fig. 4 shows how an avionic model with different versions (i.e. software, hardware, etc) is designed. A common high level model is developed in order to encompass different functions which correspond to the different implementations. The necessary data (input, output, current context) is contained in a global data structure allowing easier task reallocation from a software node to a hardware node for example. The functions will be chosen depending on the test scenario requirements and the best mapping solution found by the heuristic.

Indeed, if a hardware accelerator is needed, hardware IP-cores are already implemented in the FPGA architecture waiting for data. In this case, the ”hardware” function will be launched. When used, the ”hardware” or the ”hy-

brid" function will just send and receive data to/from the FPGA node trough a PCIe driver increasing the simplicity of an avionic model execution and reallocation. Indeed, the global data structure on each model contains the necessary "context" data to reallocate the model with the current context T&S session in order to continue the session without a full restart. For now, the large number of FPGAs allow us to implement all the needed models and use those only when necessary. Finally, in function of the chosen mapping, the launcher will send a parameter to each avionic model to inform it about the selected function.

In the next section we will present our monitoring performance test results, execution time comparison between different avionic models implementations and communication performance measurement.

5. EXPERIMENTAL RESULTS

5.1. Hardware experimental environment setup

Our real-time prototype environment consists of an Intel i7-2860QM CPU, 8-GByte DDR3 memory, and the ML605 Virtex-6 Xilinx board. The FPGA board was easily plugged onto the host motherboard through a PCIe slot that can support 2.5 GBytes/sec throughput. The system runs with Linux kernel 2.6.37. However, our approach is generic and can be used with different computer system configurations or Linux kernel versions. After that, we enhanced our prototype environment with soft real-time functionalities in order to satisfy critical timing constraints. This aspect is quite pertinent for industrial application domain. For this reason, we used processor affinity optimization as described in [7].

Using FPGAs as hardware accelerators could become useless if the communication bus is not efficient enough. It is important to find the best trade-off associated to the application needs. For example, Dolphin[9] proposes a communication solution fully dedicated to high performance computing. The supersocket API provides an optimized version of the interface functions described in *socket.h* freeing itself from informations in message headers and acknowledgements. Indeed, there is no data control in Dolphin PCIe network. The communication performances have been measured using Dolphin PCIe Kit containing two FPGA-based boards. Fig. 5 shows the results of a customer-server communication test performance, several remarks can be drawn. First, an interesting bandwidth up to 2.5 GB/s was reached for data buffers larger than 2048 Bytes which is equal to the theoretical value. Indeed, in the Dolphin framework, the data size needs to be much higher than the header size (128 bits maximum for PCIe). Second, for the measured latency according to the buffer size, we obtained a linear curve with a worst-case value of 25 μs. Unlike the bandwidth profile

[9]http://www.dolphinics.com/products/dolphinsupersockets.html

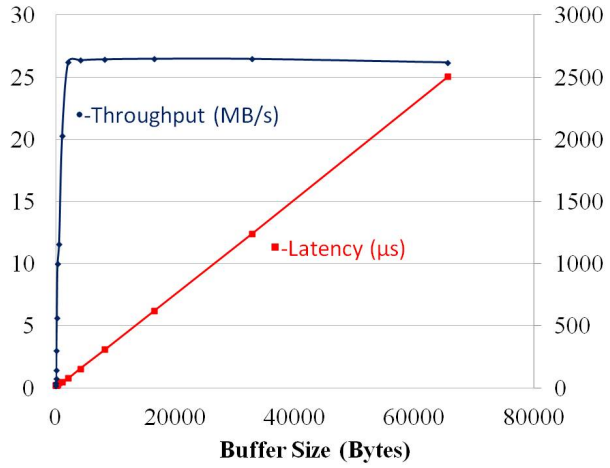

Figure 5: Dolphin bandwidth and latency performance

which tends to stabilise from a 2048 Bytes buffer size, the latency continues to increase with higher buffer sizes. This is essentially due to the data segmentation into 2048 Bytes blocks in the driver. Experimental results with the Xillybus solution offers a latency of 48 μs for a 4096 Bytes data round trip between the host and the FPGA. Although Dolphin is showing better performances, Xillybus remains the suitable solution for our application domain. Indeed, the Xillybus brings also interesting performances in addition to the transparency of CPU/FPGA communication for the designer.

5.2. Software experimental environment result

After implementing the different components (T&S project, the monitor, and the supervisor) to execute and to monitor the avionic models with real-time requirements, several experiments were conducted in order to evaluate the effectiveness of our software environment in terms of robustness and performance. First, we evaluate the robustness of our environment according to the load of the model. We define the load as the ratio between the model execution time and the duration of the defined period. To do so, we fix the period to 10 ms and we run the simulation during 1 hour (that means 360000 iterations of the same model) for different loads. The robustness is measured in term of number of runtime reconfigurations in order to achieve the entire simulation without any 90% of overload that can lead to the T&S project failure. Fig. 6 illustrates the obtained results, several conclusions can be drawn. First, our software environment succeeded to achieve the entire simulation with a negligible number of reconfigurations upto a model load of 69,4%. This result is due mainly to the fixed threshold of 70% in our environment for an anticipated overload alert. In addition, we defined a condition that stops the simulation

after n successive runtime reconfigurations, where n is the number of available cores. In our case n is fixed to 6, for this reason between 70% and 90%, the T&S project fails. Hence, the robustness of the software environment depends mainly the model load. For this reason, we will emphasise in the Section 5.3 the usage of hardware implementation on the FPGA for critical models. Second, we are interested in the time repartition inside the model for the different steps during the simulation. In the Fig. 6, we take as an example a load of 67,3% to carry out the model, the remaining part of the period is almost entirely occupied by the "waiting" for the synchronisation signal from the timer. The time required by the reconfiguration and the inputs/outputs (I/O) operations is negligible, respectively 1,27% and 0,24% of the period).

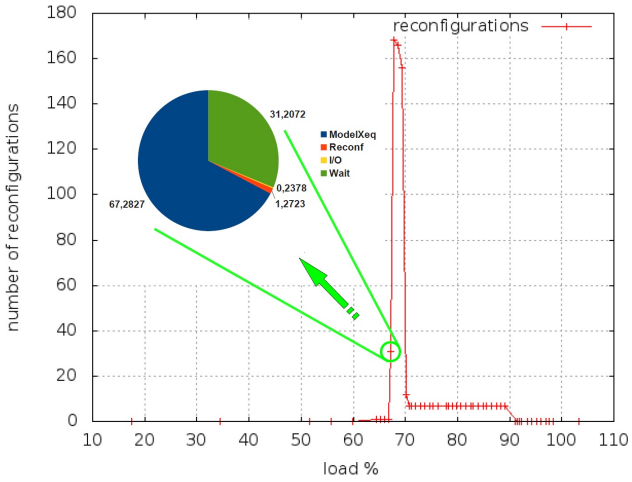

Figure 6: Number of runtime reconfigurations according to the model load

In the Fig. 7, we detail the time repartition of each step inside the model according to the load. The increase of the execution time of the model is directly proportional to the load and inversely proportional to the "waiting" time. The value entitled "complete" corresponds to the percentage of the 360000 iterations that are completed successfully before the end of the simulation. The total duration of the reconfigurations is really low whatever the load is. There is however a small increase when the load is greater or equal to 70% (always smaller than 1.25%). Similar observations may be established when considering the I/O with a maximum percentage of 0.20%.

Finally, in the Fig. 8 we detail the average time repartition inside the supervisor while considering the different steps. These values are collected during all the simulations. Fig. 8 shows that the supervisor spends almost all its time (98.74%) waiting for an event from the model. The percentage related to I/O and LPT are respectively equal to 1.08% and 0.3%. The percentage of the time spend to retrieve the

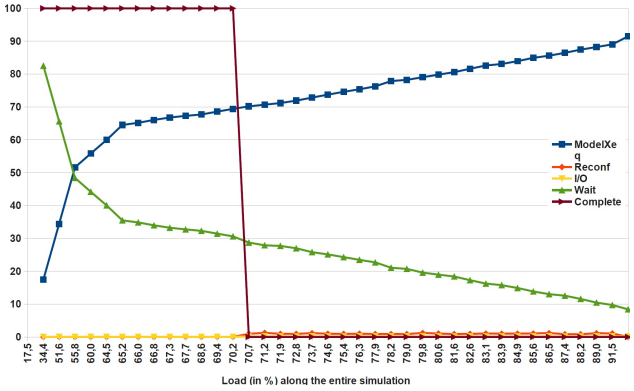

Figure 7: Time repartition according to the load inside the model

loads of the cores from the monitor is negligible (0.01%) whereas the percentage spent in the specific action of reconfiguration is not measurable (almost 0%) since the reconfiguration is done in the model itself. These figures show that the supervisor and the monitor have enough capacity to supervise and to monitor several models. This paves the way for future versions of our framework.

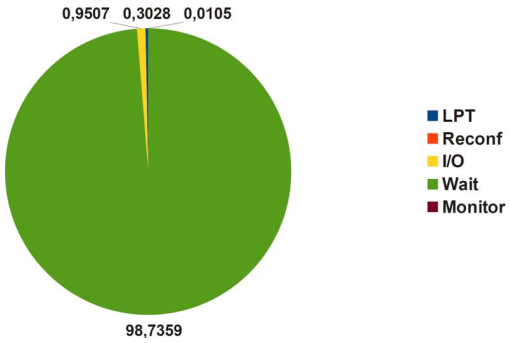

Figure 8: Time repartition inside the supervisor

5.3. Hardware model design

Despite parallelism tools such as Pareon do not show a large parallelism degree for avionic models as the *Flight Mechanic model*, we decided to implement it in a hardware version. Indeed, this is the most representative of the requirements and essential in the T&S scenario. Moreover, the *Flight Mechanic model* analysis shows few data dependency between the mathematical equations intrinsic in this model. It becomes obvious that a parallel implementation of the independent equations will provide the greatest performances than can be reached in FPGA architecture. The pure software version gives a 20 μs execution time using a Quad-core processor set at 2.5 GHz. In our previous work, we have already implemented this model using a Xilinx Mi-

croblaze soft-core with Xilinx CORDIC hardware accelerators for trigonometric functions. This design presented an execution time of 0.9 ms 30x faster than a software implementation without the usage of CORDIC IP cores. In the light of those observations, we decide to compare our results to a full hardware design. In order to take benefit from the large granularity of the *Flight Mechanic model*, we designed the intrinsic equations as an electronic circuit with a pipeline architecture. The pipeline structure offers the possibility to provide data faster. Indeed, the first computing cycle will provide the longest delivery time (T1) and each next delivery cycle will be equal to the first computing cycle time divided by the number (N) of stages (T1/N) using all stages continuously with new data. In our case study, the VHDL implementation offers a T1=8 μs execution time, the 8 stages allowed us to provide data every 2 μs, corresponding to the longest stage execution time. This result offers the opportunity to move model A from a CPU to the FPGA in order to bring performance and real-time constraint respect at the price of the hardware design time.

5.4. Software-hardware context switch scenarios

As an example of a use-case, we run the T&S loop presented in Fig. 1 periodically using software models on the host machine with a timing constraint of 10 ms. As presented before, a real hardware (i.e. *Automatic Pilot*) can replace its corresponding simulation model during the T&S phase. In order to reduce the communication latency between the real hardware and a simulated model (i.e. *Flight Mechanic model*), this latter can be migrated from the host to the FPGA containing the I/O IP core. Fig. 4 shows how a software/hardware (or vice-versa) context switch can happen at runtime. In fact, the **HwFunction()** and **SwFunction()** share the same data context and the I/O data structure in order to perform the calculation node switching more efficiently. Let us highlight that the **HwFunction()** communicates with the hardware core using the Xillybus solution, and thus bringing a total transparency for the system. Our solution avoids additional timing cost for the software-hardware context switch. As a second scenario, an anticipated overload alert can be generated for a task re-allocation in order to avoid the violation of timing constraints and thus the failure of the T&S phase. Our run-time mapping heuristic is developed to deal with the model overloads and to make a decision about the dynamic context switch according to the available software or hardware implementations.

6. CONCLUSION

In this paper, we have emphasized first a future avionic T&S environment based on emerging heterogeneous CPU/FPGA systems. Second, we presented a soft real-time software environment which can map various T&S scenarios on CPU/FPGA platform. The experimental results demonstrate the effectiveness of our environment for avionic T&S application. As future works, our investigation will concern first the design and the validation of a run-time task mapping with highly communicating software and hardware models. Secondly, our objective will focus on the mapping of several scenarios in the same CPU/FPGA architecture with T&S models sharing. Finally, our prototype will include all the solutions explored in our research such as MCAPI for the communication.

7. REFERENCES

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

[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] ClearSpeed, http://www.clearspeed.com/, 2011. [Online]. Available: http://www.clearspeed.com/

[4] T. Oguntebi, S. Hong, J. Casper, N. Bronson, C. Kozyrakis, and K. Olukotun, "Farm: A prototyping environment for tightly-coupled, heterogeneous architectures," in *FCCM'10: The 18th Annual International IEEE Symposium on Field-Programmable Custom Computing Machines*, Charlotte, North Carolina, USA, May 2010.

[5] A. Brandon, I. Sourdis, and G. N. Gaydadjiev, "General purpose computing with reconfigurable acceleration," in *International Conference on Field Programmable Logic and Applications, FPL*, Los Alamitos, CA, USA, 2010.

[6] S. Moore, P. Fox, S. Marsh, A. Markettos, and A. Mujumdar, "Bluehive - a field-programable custom computing machine for extreme-scale real-time neural network simulation," in *FCCM*, 2012.

[7] G. Afonso, R. B. Atitallah, A. Loyer, J.-L. Dekeyser, N. Belanger, and M. Rubio, "A prototyping environment for high performance reconfigurable computing," in *ReCoSoC*, 2011.

[8] O. Souissi, R. Ben Atitallah, A. Artiba, and S. E. Elmaghraby, "Optimization of run-time mapping on heterogeneous cpu/fpga architectures," in *9th International Conference of Modeling, Optimization and Simulation - MOSIM12*, Bordeaux, France, 2012.