In flight real-time adaptation

Bertrand Granado, Marc Gatti, Julien Denoulet, Martin Rayrole

2017-01-2169

In flight real-time adaptation

B.Granado[1], M. Gatti[2], J. Denoulet[1], M. Rayrole[2]

1 – Sorbonne Universités, UPMC Univ Paris 06, UMR7606, LIP6, F70005 Paris, France, 2 – Thales Avionics, 75-77 Avenue Marcel Dassault, 33700 Mérignac

Abstract

Avionics is one kind of domain where prevention prevails. Nonetheless fails occur. Sometimes due to pilot misreacting, flooded in information. Sometimes information itself would be better verified than trusted. To avoid some kind of failure, it has been thought to add an new kind of embedded monitoring that enable adaptation.

1 Introduction

It is well known that avionics is a very restricting domain for obvious safety reasons. Along with miniaturization comes the idea of integration. More functionality on one spot requires a good management of privacy and congestion on shared platforms. This is why determinism is one of the keywords of avionics works. This led to protocols like ARINC653[1] assuring that, multitask embedded programs respect a predictable policy applied by the operating system (OS). Another key protocol is ARINC664, which guarantees that multiple communicating systems efficiently share the network. These two protocols are pillars of the Integrated Modular Architecture (IMA) concept [2]. IMA concept consists of multitask module hosting ARINC653 OS, interconnected with ARINC664 data network. Compared to federated avionics architecture, it considerably reduces the overall weight and power consumption for aircraft, reduces development expenses and design cycle times as well as maintenance costs. With the intention to step forward with this concept, the CORAC (The Council for Civil Aeronautics Research) develops a technological demonstration platform (PDT) called Extended Modular Avionic (AME) [3]. Therefore, as partner of the project, we work on a project dedicated to monitor the system.

In this paper, we use our embedded real time monitor to take real time in-flight decision to adapt the behavior of the digital processing. We first present our embedded SystemC simulator that can be embedded in a plane as a processing module to monitor data traffic generated by key avionic applications in order to detect suspicious behavior such as missing data, unexpected communication of simply incoherent data. Second we explain how, by exploring several architecture configurations, the simulator can help the IMA system to decide how it should be configured, how the processing resources should be assigned in order to efficiently manage specific or critical situations. In the next sections, we first introduce our method and its related tools, most notably the SystemC language and the modifications required to make this language compliant with avionic constraints. Next, we show how the simulator can be implemented in a QorIQ T2080-based board. We then introduce the use cases (monitoring and architecture exploration) to illustrate the benefits of our approach. Finally, we will conclude.

2 Embedded simulator

2.1 Method presentation

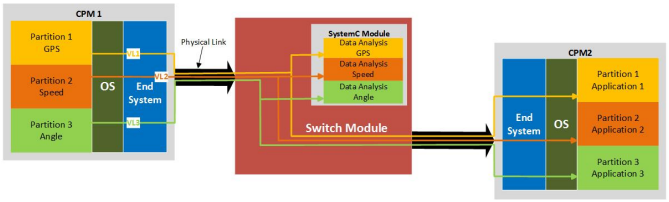

Figure 1: Embedded simulation methodology

In Figure 1, we present the principle of our method. We consider an avionic architecture featuring core processing modules (CPM) implementing several applications and generating data traffic and avionics switch modules (ASM) which route data packets to their destination CPM.

As an example, CPM1 in **Erreur ! Source du renvoi introuvable.** features three partitions, each one hosting an application dedicated respectively to GPS, Speed, and Angle estimation. Through an ARINC664 communication End System, data generated by these applications are sent through several Virtual Links (VL) of the data network. While performing data traffic management, the ASM also implements a simulator that runs a timed model of the expected communication traffic, considering the OS and network parameters. The ASM is the privileged place to implement a simulator, since its CPU only manages message traffic and has available time.

The simulator performs two types of verification: temporal consistency which checks whether communication occurs at the expected time, according to the system scheduling, and data consistency which analyses N consecutive data values to determine if their evolution is coherent or if we can assume an error has occurred.

The simulator can also be used as an architecture exploration tool. It can model different application mappings on the system and test which one is the most efficient to handle specific scenarios. This could help the system to perform dynamic reconfiguration when it comes upon critical situations.

To achieve such a goal, we have chosen the SystemC [4] language as an appropriate candidate to model as well software (application) and hardware system (processors and communication modules) under time constraints (defined by ARINC653 and ARINC664). The next subsections briefly presents the SystemC language specifications, as well as SystemCASS, a SystemC simulation kernel we modified so that it can meet avionics requirements. We finally show the implementation of our SystemCASS simulator on a QorIQ T2080 design board.

2.2 SystemC

SystemC is a C++ class library based on object-oriented design concept (OOD) providing common Hardware Description Language (HDL) features. As such, it allows hardware description along with software development. Hardware behavior concurrency is simulated by the way simulation time is being managed by the simulator.

Hardware components are modeled using the sc_*module* class and are interconnected to each other with *sc_por*t class objects. Module internal registers are represented by *sc_signals*, and module behavior by processes, which can be described as functions triggered by the update of ports or signals that are registered in a sensitivity list. A SystemC program usually consists in an elaboration phase where all the elements of the described system are declared and assembled, and where all processes are listed. Then comes the simulation phase, which is initiated by the *sc_start* method, which is a function of the simulator. Finally, the cleanup phase ends simulation, by cleaning objects and structures.

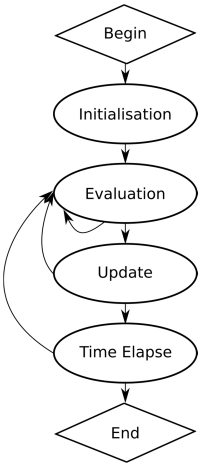

Figure 2: SystemC flow

The role of a SystemC simulator is to manipulate the timestamp to simulate the concurrency of hardware behavior. It determines in which order processes must be executed, and when values of ports and signals must be updated. The Accelera Systems Initiative (ASI) provides an event driven simulator with the language library. The simulator operates according to **Erreur ! Source du renvoi introuvable.**.

The simulation phase features three steps: ***Evaluation***, in which the simulator checks which processes must be executed, according to their sensitivity list. The simulator then executes these processes. When this is done, the second step, ***Update***, updates the values of ports/signals according to the previous processes executions. If signal or ports updates trigger a process sensitivity list again, then we go back to the evaluation step. When no process is triggered anymore, the simulation timestamp is updated in the ***Time Elapse*** step.

The ASI simulator, as it is implemented, features memory dynamicity, which avionic constraints don't allow. Furthermore, process scheduling at each timestamp is dynamic and non-deterministic [5]. This doesn't affect the result of the simulation, but can be an issue in an avionic context, considering execution time.

2.3 SystemCASS

SystemCASS (SystemC Accurate System Simulator) [6] is a SystemC simulator that establishes a static scheduling of processes, which is made at the start of simulation, To do so, SystemCASS requires describing all component models as CFSM (Communicating Finite State Machine) using a CABA (Cycle Accurate Bit Accurate) abstraction level. Furthermore, a single clock must drive all modules.

SystemCASS modules can include three types of processes:

Transition: triggered by the clock rising edge, it sets the new values of registers, depending on their actual values as well as input port values.

Moore/Mealy Generation: triggered by the clock falling edge, these processes set the new values of output ports, depending on register values only (Moore) or register and input port values (Mealy).

When calling the *sc_start* method, SystemCASS creates depending graphs that generate the static scheduling of processes, which will be used throughout the simulation phase. This implementation ensures a deterministic behavior of the simulation.

As a result, SystemCASS is more suitable to avionic constraints than a dynamic event driven simulator. As we use gcc compiler, SystemCASS original implementation featured dynamic memory allocation during the creation of the depending graph after the elaboration phase, and right before the simulation phase. So we worked to remove these dynamic allocations. To do so, we first used a static version of gcc compiler and second we identified in run-time all the encountered dynamic memory allocations and replaced it with static memory allocations.

To identify dynamic allocation we used gdb debug tool and a script that put breakpoints on malloc call, this script is:

```
set logging file trace.txt
set logging on
break malloc
Command
Bt
Continue
End
Break main
Command
Continue
End
Run
Set logging off
Quit
```

2.4 Implementation

To validate our system, we designed a demonstrator based on two QorIQ T2080 design boards each featuring a PowerPC E6500 processor (**Erreur ! Source du renvoi introuvable.**). The first board assumes the role of a CPM module, running test applications which are supposed to transmit data to other CPM modules. The second board assumes the role of an ASM module. It performs data reception and runs the embedded SystemCASS simulator.

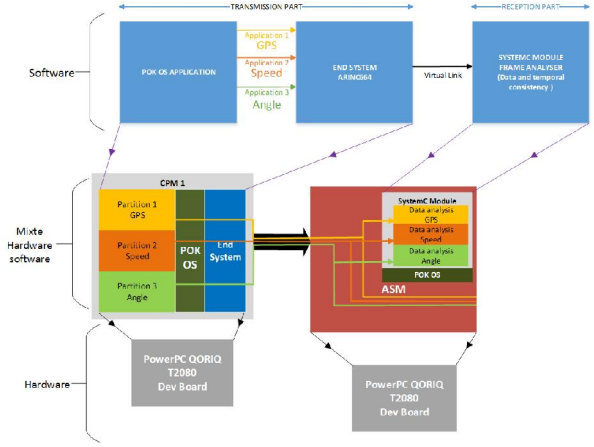

Figure 3: T2080 demonstrator

Each QorIQ T2080 board hosts the PolyORB Kernel (POK) operating system. POK is a partitioned operating system compliant

with ARINC653 avionic standard [7]. POK ensures enforcement of safety and security requirements at run-time. It also provides some example of avionics applications. One of these applications is the Flight Management (see Figure 4)

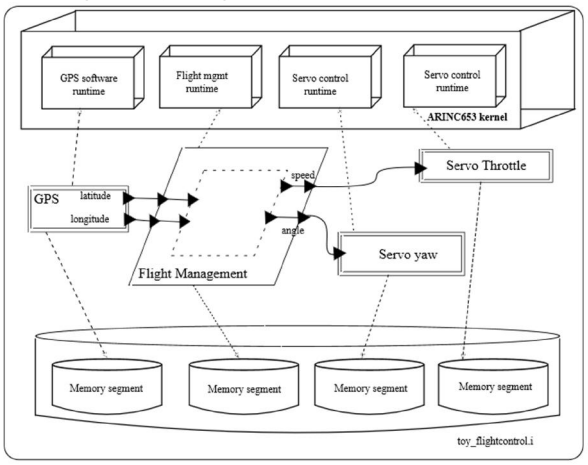

Figure 4: POK Flight Management Application

We used this application to run on the CPM QorIQ board. POK OS handles the flight management application (GPS, Speed and Angle) and at the same time handles the ARINC 664 End System module. On the ASM QorIQ board, POK handles the SystemCASS simulator to perform data monitoring or architecture exploration.

3 Use cases

3.1 Data monitoring

Considering the predictability and determinism of applications software ruled by the protocol ARINC653 and their windows of communication in ARINC664, one can predict part of the aircraft data traffic. Some verification within the communication protocol already exist concerning the integrity of the data transport but none can analyze the content itself to determine whether one or another application is really supposed to send a value, or if a communication disappeared or if a value is simply incoherent. Obviously simulating the whole communication flow to determine if it is coherent would be too much time expensive in simulation. The idea is to target specific applications, or specific suspect behaviors (missing material, erroneous values) we could watch over during the flight. Knowing what we're looking for, we can then create a simplified functional timed model of applications as communication providers. On the basis of ARINC664 and ARINC653 configurations values (major frame, bandwidth allocation gap ..), we can predict communication by simulation and compare it with the real traffic to verify temporal as well as data consistency.

The application is implemented as follows: on the CPM QorIQ board, POK runs the Flight Management application, which features three partitions (speed, angle and GPS) and generates the application data (Figure 5). POK's ARINC653 properties guarantee space partitioning (meaning that memory of partition is protected) and also guarantees time partitioning (meaning that only one partition at a time is executed).

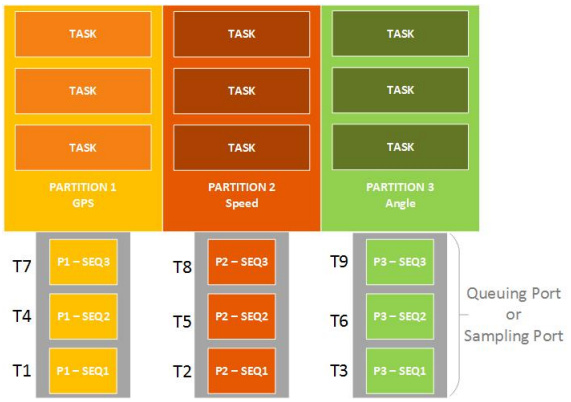

Figure 5: Data Generation and Space Partitionning

The execution of each partition is handled by a static scheduler (as we can see in Figure 6) and is defined by the system integrator. Each partition (P1, P2 and P3) has a set of execution windows (T1, T2, T3) and this set of windows is repeated in time (T4, T5, T6 and so on…) and at the same order, which guarantees that each partition has access to the system resources once in a MAF (Major Frame).

Figure 6: Partitionning Scheduling

Once that data is generated by POK, they are put in the Queuing Port or Sampling Port and are then sent to the End System with the order defined by the scheduler. Queuing Port can be seen as a buffer and the Sampling Port as a FIFO. The End System then encapsulates the data in an ARINC664 frame with the specification of the Virtual Link (BAG, Frame Size, Jitters) that has been defined by the system integrator (see Figure 7). A Virtual Link defines an unidirectional logical connection from one source End-system to one or several destination End-System(s). Each partition has a dedicated Virtual Link (VLi is dedicated to the data of the Partition i).

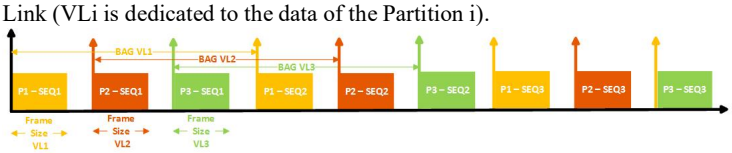

Figure 7: ARINC 664 Frame at the Output of the End System

On the ASM QorIQ board, POK runs the embedded simulator. SystemCASS runs a SystemC module that analyzes the ARINC664 frames coming from the CPM board. It performs data and temporal consistency.

The data consistency consists in analyzing the payload of the ARINC664 frame that contains data of each application (GPS, speed, angle). In order to do so, a verification of the physical variation law between two data values T and T+1 for each application is performed. For example, the verification of the value of the partition P1-SEQ1 and P1-SEQ2 is performed as shown in Figure 8.

Figure 8: Data Consistency

On the other hand, temporal consistency consists in verifying that the execution order of each partition is consistent with the scheduling defined by the transmitter part Figure 9 shows an example of the temporal consistency verification.

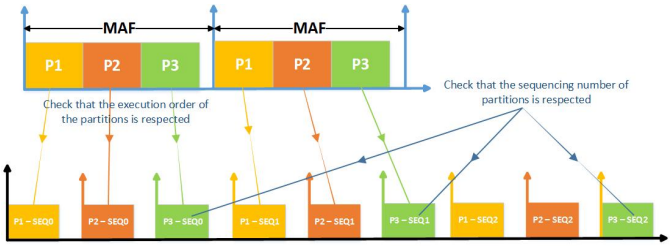

Figure 9: Temporal Consistency Verification

3.2 Architecture exploration

Embedded simulation can also be used to help decide in real time how the system should be configured (i.e. what is the most efficient application mapping configuration) when critical situations occur and processing resources should only focus on the most essential applications.

To do so, a predefined set of application mapping configurations should be stored in a library. When the system detects some incoherent execution or some major misbehavior (based on the data monitoring simulation, or other verification mechanisms), a reconfiguration procedure can be started (see Figure 10). The embedded simulator then runs the stored configurations to get performance profiles. A decision motor then selects the most appropriate configuration (whether it's the one who reaches the best performance, or simply the first configuration who meets a predefined performance requirement)

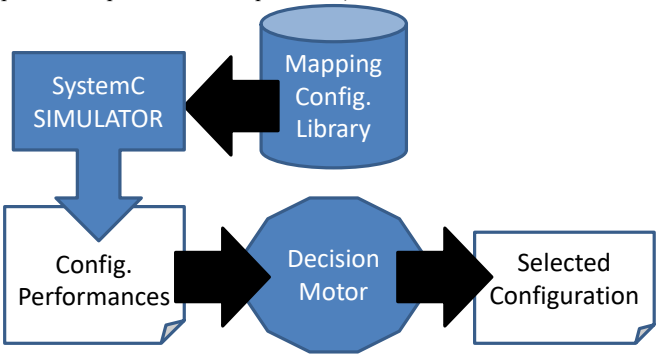

Figure 10: Architecture exploration use case

The system can then be dynamically reconfigured to remap the application according to the simulated scheme.

4 Conclusion

In this article we have presented a new method featuring embedded simulation. We have implemented a SystemC-based simulator compliant with avionic requirements. The simulator allows the real time monitoring of ARINC664 communications. The goal of this monitoring is to check whether communication occurs at the expected time, according to the system scheduling, and to validate data consistency. The simulator also allows architecture exploration to determine the most efficient mapping of applications in order to handle specific situations We show how this simulator could be used to adapt the in-flight processing in real-time to react to critical events.

References

[1] C. R. Spitzer, U. Ferrell, T. Ferrell, and P. J. Prisaznuk, "ARINC Specification 653, Avionics Application Software Standard Interface," in *Digital Avionics Handbook, Third Edition*, CRC Press, 2014, pp. 625–632.

[2] J. P. Paul, "ARINC 653 role in integrated modular avionics (IMA)," in *27th Digital Avionics System Conference Proceedings*, 2008, vol. 1.

[3] "CORAC," *COnseil pour la Recherche Aéronautique Civile.* .

[4] O. S. Initiative, "IEEE standard SystemC language reference manual," *IEEE Comput. Soc.*, pp. 1666–2005, 2006.

[5] C. Schumacher, J. H. Weinstock, R. Leupers, and G. Ascheid, "SCandal: SystemC analysis for nondeterminism anomalies," in *Specification and Design Languages (FDL), 2012 Forum on*, 2012, pp. 112–119.

[6] R. Buchmann, F. Petrot, and A. Greiner, "Fast cycle accurate simulator to simulate event-driven behavior," in *Electrical, Electronic and Computer Engineering, 2004. ICEEC'04. 2004 International Conference on*, 2004, pp. 35–38.

[7] J. Delange and L. Lec, "POK, an ARINC653-compliant operating system released under the BSD license," in *13th Real-Time Linux Workshop*, 2011, vol. 10.

Contact Information

Bertrand Granado
Laboratoire LIP6 UMR7606
Université Pierre et Marie Curie
BC 167, Tour 24/25 - 5ieme Etage
4 place jussieu
75252 Paris Cedex 05

Tél : 33 (0)1 44 27 96 33
Email: bertrand.granado@lip6.fr
Website: http://www.lip6.fr