# IMA Platform Computing Module based on Partial Reconfigurable FPGA

R. F. Romero[1], O. Saotome[1], D. S. Loubach[2], E. G. O. Nóbrega[2], I. Sander[3] and I. Söderquist[4]

[1]Electronics Engineering Division, Aeronautics Institute of Technology-ITA, São José dos Campos, Brazil
(Email: r_fromero@hotmail.com, *osaotome@ita.br*)
[2]Advanced Computing, Control & Embedded Systems Lab / FEM, University of Campinas – UNICAMP, Campinas, Brazil
(Email: *dloubach@fem.unicamp.br, egon@fem.unicamp.br*)
[3]Electronics and Embedded Systems Department, KTH Royal Institute of Technology, Stockholm, Sweden
(Email: *ingo@kth.se*)
[4]HMI & Avionics Department, Saab AB, Linköping, Sweden
(Email: *ingemar.soderquist@saabgroup.com*)

## ABSTRACT

**Integrated Modular Avionics (IMA) is a real-time network of computing modules sharing a same computing environment and some basic services. These modules host several applications of different criticality levels separated by a partitioning mechanism. This partitioning ensures robust separation and logical independence among functions and applications. While federated architectures are based on dedicated modules and distributed processing units for each functionality, IMA architecture provides reduced size, weight, power and cost by having a number of different functionalities in a same computer. Big aircraft companies such as Embraer, Airbus, Bombardier, Dassault, Boeing and Saab already have adopted IMA architecture in their aircrafts design. The trend of modular avionics systems, especially considering IMA second generation, points out new requirements such as reconfiguration. Therefore, the use of Field Programmable Gate Arrays (FPGAs) as a computing module in an IMA architecture is very promising. Modern FPGA technology including runtime partial reconfiguration along with advanced software design suites enables complex avionics systems design and on-the-fly adaptation. Considering this scenario, this paper proposes an IMA module design approach based on partial reconfigurable FPGA. The module can be able to exchange its internal behavior according to unexpected environmental changes or new application functionalities. The purposed approach allows a highly efficient hardware redundancy inside one chip itself, besides offering low-cost runtime reconfiguration feature.**

## INTRODUCTION

Avionic architectures are real-time embedded systems (RTES) composed of digital processing modules and communication buses which supports applications such as communication, navigation, flight control, stability, guidance, aircraft health monitoring, passenger entertainment and weather measure. Avionics systems represent about 40 to 50% of aircraft cost (Bieber, et. al., 2007).

In the past, avionics systems were based on federated architecture where each function was performed by line-replaceable unit (LRU) connected to its dedicated sensors and actuators (Wolfig & Jakovljevic, 2008). The dependencies between standalone subsystems are well understood and there are limited resources sharing between the processing units.

Federated systems concept met its limit in 1990s (Segvik, Krajcek and Ivanjko, 2016) when advances on digital systems applied to aviation electronics implied in limits regarding cost, weight and space availability.

Considering this scenario, the first generation of Integrated Modular Avionics (IMA1G, or just, IMA) concept, based on centralizing and reutilization of communication and computation resources is adopted in order to keep the costs, weight and volume within reasonable limits.

The IMA approach comprises two principles, (i) integrate multiple software functions, with possible different criticality levels on a single avionic computing resource; and (ii) strict and robust partitioning. Allocating to each function its own non-shared virtual resources prevents functions interference on each other.

The two above mentioned principles intended to provide integration, flexibility, interoperability, weight and power consumption reduction when compared with federated architectures. By the other hand, computation sharing adds new indirect dependencies between subsystems and communication sharing might causes limitation of the communication flows.

Besides dependencies between applications and communication problems, fault tolerance, containment and reconfigurability leads to a natural trend for the second generation of IMA (IMA2G) (Adrillon & Aviation, 2013).

Also present in the literature as Distributed Integrated Modular Avionics (DIMA) the second generation of IMA is now concentrating researches and development efforts worldwide both in academy (KTH, Zagreb) and industry (Airbus, Saab, Boeing).

The IMA2G aims to significantly improve the balance between processing power and power consumption thru dynamic adaptive real-time embedded systems with mixed criticality. The adaptive capability can be reached using hardware reconfiguration techniques, based on the redistribution of functions according to criticality maps, and optimizing the use of chip resources by design space exploration. Therefore, a catastrophic failure may be avoided and the avionic system is expected to deliver its functionality even in degraded mode.

Dynamic partial reconfiguration (DPR) is present in modern Field Programmable Gate Array (FPGA), such as Xilinx 7-series, along with advanced Integrated Development Environment (IDE) design by FPGA vendors, such as Altera Quartus and Xilinx Vivado. In our context, DPR is a run-time partial reconfiguration made by an on-chip controller.

In this context, we purpose in this paper a practical IMA module design approach based on reconfigurable FPGA aiming cost, area and power reduction. The concept introduces on-chip redundancy for mixed criticality modules synthesized in FPGA.

The design flow is based on Xilinx 7-series FPGA devices along with Xilinx Vivado design suite, but can be extended to other FPGA vendors and models.

Next section presents an overview of partial reconfigurable FPGA features and design process. Subsequently, the practical design approach is presented with an application example. The conclusions are exposed in the final section.

## PARTIAL RECONFIGURABLE FPGA

FPGAs are integrated circuits widely used to prototyping different complexity levels of digital circuits in the same die. The circuits are described by the designer through a hardware description language, such as Verilog and VHDL. The set of circuits are loaded in vendor IDE and then synthesized and implemented for the target FPGA model. The IDE generates a bit stream which can be downloaded in an internal or external memory connected to the FPGA. When power-up the device reads the memory and generate the described circuits using an array of combinational logic blocks and reconfigurable interconnects. Input/output blocks allow external signals to be connected with the internal signals of the FPGA.

In this way, the FPGA technology provides the flexibility of on-site programming and re-programming through memory content uploading the full configuration file. The flexibility can be increased allowing the modification of an operating FPGA design by uploading a partial

configuration file. Partial bitstream can modify reconfigurable regions in the FPGA without compromising the integrity of the applications running on those parts of the device that are not being reconfigured, i.e. the static logic, also known as top-level logic.

Partial reconfiguration provides design update in runtime, increases flexibility in the choices of algorithms or protocols available to an application, besides improving FPGA fault tolerance, reducing size, weigh, power and cost.

On the other hand, partial reconfiguration is not trivial due to possible device fragmentation and communication between static and newly implemented partition (Bobda, 2007).

As a conventional FGPA design it is divided in three main steps, synthesis, implementation and bitstream generation. A partially reconfigurable design requires bottom-up synthesis, i.e. each module has its own synthesis project. There is no optimizations across module boundaries.

The top-level checkpoint must have one black box for each reconfigurable partition, where black boxes have entity/module (consider VHDL/Verilog description languages) but no logic. Each black box has its own partition, a logical section of the design, user-defined at a hierarchical boundary, to be considered for design reuse. A partition can be assigned to a specific device physical area, also known as pblock (physical block) in Xilinx devices. A pblock must be specified by the designer before the implementation step.

The synthesis tool infers or instantiates I/O buffers on all top level ports of entities/modules and considers a black box as a Reconfigurable Partition (RP). A RP is the level of hierarchy within which different Reconfigurable Modules (RM) are implemented. Therefore, a single design may have many reconfigurable partitions, each one with a set of reconfigurable modules within must have the same entity/module.

To avoid wrong connections between the ports of RM which are designed to fit with the respective black box, all I/O buffer connections must be turned off. The designer have to create as many configurations as necessary to implement all reconfigurable module at least once. Then, the design tool creates one checkpoint for each synthesized configuration.

Each entity/module of the design can be implemented. The static module implementation must be consistent for each configuration. RM implementation depends on both static module checkpoint and RM checkpoint.

The implementation process has two essential steps, place and route design. Once all configurations have been placed and routed through implementation process an algorithm verifies consistency between the configurations.

In the generate bitstream step, the design tool creates a full standard configuration file plus one partial bit file for each RM within that configuration. In order to store partial bit files in an external flash memory is needed to convert these files from *bit* to *bin* extension.

## PRACTICAL DESIGN APPROACH

The purposed design flow can be applied for any FPGA design suite. In our example we used the Xilinx Vivado Suite. We divided our design approach in five steps, (i) define superset entity/module for all functions; (ii) design finite state machine (FSM) for mux control; (iii) generate out-of-contect IPs; (iv) create top-level logic, run synthesis and implementation; and (v) generate bitstreams and update PRC. In order to clarify the present contents we introduced a case study.

### *A. Case study*

The IMA design approach purposed in this paper is applied in a hypothetical aircraft, composed of three functions: avionics navigation, flight control and passenger entertainment, illustrated in Figure 1. Navigation and flight control are vital for aircraft, so they have high criticality, while passenger entertainment are not essential, it is low critical.

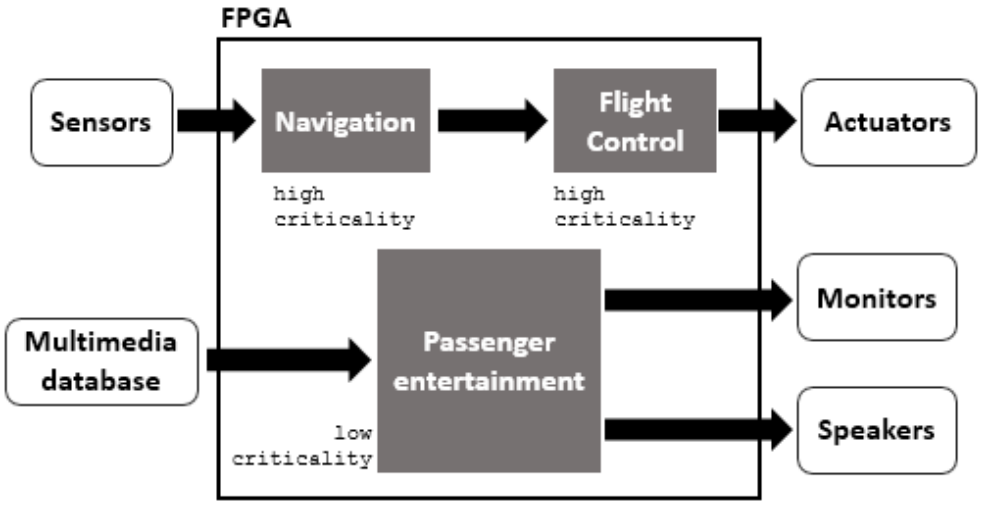


Fig. 1. Avionic system example

*B. Define superset entity/module for all functions*

The first step is recognize the IMA functions according to its criticality level. In our example, as mentioned above, navigation and flight control have high criticality, while passenger entertainment has low criticality.

Each function must be described as a component/module. In a failure situation a high critical function represented by its RM can be implemented in a RP which had a low critical function implemented originally, Figure 2. For instance, if there is a fault in the flight control the entertainment must be substituted by a copy of the flight control system. I/O mapping and partition scheduling are managed by FPGA reconfigurable controller which will be explained below.

The reconfiguration is possible if both low and high critical functions can be implemented in the same RP, so they must have exactly the same entity/module signals. Therefore, all functions must have the same entity/module, i.e. same input and output signals types and names. This entity/module is named superset and it contains all the input and output signals present in all the functions.

In fact, in the architecture of each functions there will be non-used input signal and open output signals. For instance, entertainment architecture do not use any signal connected to navigation system, although navigation inputs and outputs are in the superset.

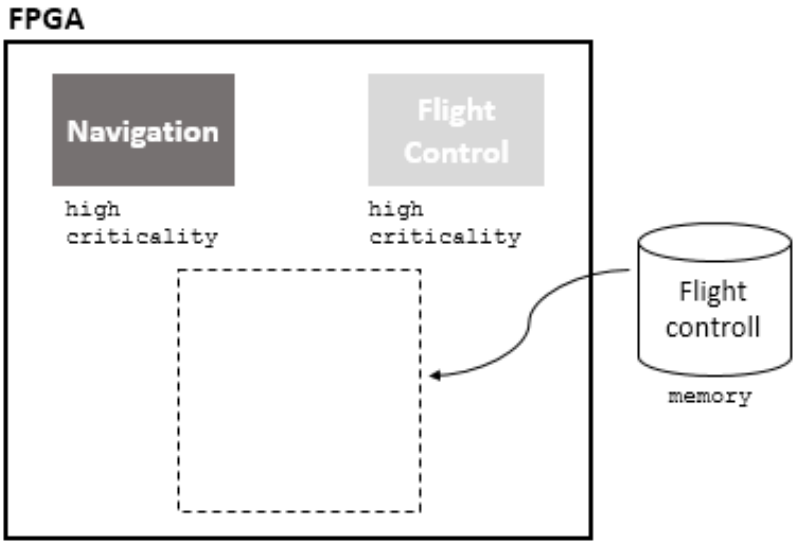


Fig. 2. Module substitution through reconfiguration

*C. Design finite state machine for mux control*

As seen before, same inputs and outputs signals will not be used in each function. For inputs there is no problems, once an input signal can be simply left in high impedance. However, two or more outputs must not drive a same line due to the conflict situation between low and high digital levels. The synthesis tool is not able to detect mutual exclusion between two module outputs, so the mux ensures module outputs isolation. Front of this problem, Figure 3 shows our purpose of using a finite state machine in order to control the buses through multiplexers (mux).

When the fault is detected the finite state machine changes the mux selection for swap the bus connection from the low to the higher criticality function. Therefore, this concept spends one mux per RP.

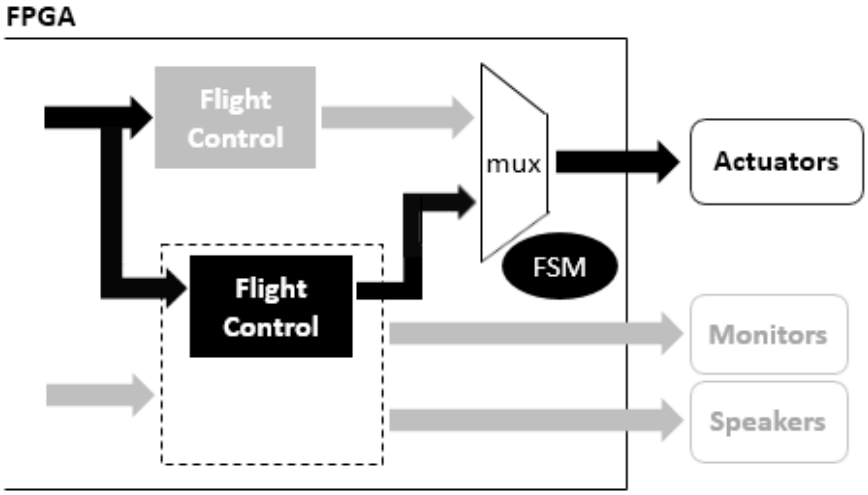


Fig. 3. FSM with mux for bus control

*D. Generate out-of-context IPs*

Xilinx Vivado IDE provides a set of IPs in order to facility the complex dynamic partial reconfiguration process. These IPs should be generated out-of-context to be after included in the design. They include: Partial Reconfiguration Controller (PRC), Internal Configuration Access Port (ICAP), External Memory Controller (EMC) and JTAG to AXI interface.

The PRC receives hardware or software trigger events and pulls partial bitstreams from memory and delivers them to ICAP. Besides, PRC assists with logical decoupling and customizable startup events. The ICAP does not need to be generated because is a primitive.

The PRC is configured according to the number *N* of reconfigurable partitions, i.e. one RP for each function, and *N* reconfigurable modules for each RP. The amount of hardware triggers must be *N* as well. In the beginning of design, addresses and sizes of partial bit streams should be left as zero, because the bitstream sizes are defined only after the implementation, once it depends on the space occupied in FPGA and varies among different Vivado versions.

Finally, the EMC receives commands from PRC and searches for the correct partial bitstream stored in the flash memory for be delivered to ICAP. The EMC interface is connected with flash memory through FPGA general purpose I/Os.

*E. Create top-level logic, run synthesis and implementation*

The last step before the synthesis is to create a top-level logic containing all functions, FSM and IPs. At this point the I/O and timing constraints have to be defined and the synthesis can be executed.

Opening the Vivado Graphical User Interface (GUI), it is possible to draw the pblocks for each RP. This step can be pointed out as a hurdle into design. Pblocks top and bottom edges must match the clock region boundaries and left or right edges must not split interconnect columns inside the FPGA. The first condition is fitted changing SNAPPING_MODE value to

ON. However, the pblock left and right edges are not so easy to define. After ensuring that there is no design rules violation, the final Xilinx Design Constraint (XDC) must contain the pblocks constraints to perform the correct implementation.

*F. Generate bitstreams and update the PRC*

When implementation is completed, it is possible to generate the full and partial bitstreams. These files have *bit* extension and must be converted to *bin* extension to be stored in an external flash memory. The bytes size of each *bin* file is delivered by *size* command from Tool Command Language (TCL).

The partial bitstreams addresses should be calculated according to bit file sizes and memory architecture. Finally, a configuration memory file (MCS) merges bit files through TCL command. In the end, the PRC can be updated with full configuration and partial bitstreams addresses and sizes.

## CONCLUSION

We purposed a practical approach for IMA design based on partial reconfigurable FPGA and illustrated it with a simple avionic system.

Design steps were pointed out and the use of finite state machine along with multiplexers seems to be a good solution to deal with several signals from different modules driving the same bus.

The proposed design approach leads to on-chip run-time module reconfiguration, resulting in low cost, low volume and power reduction when compared with a conventional redundancy using hardware replication.

Finally, our design approach can be adapted for other FPGA vendors, such as Altera, Lattice or Microsemi.

As future work, a real avionic system can be implemented through the purposed technique and performance indicators, such as power and reconfiguration time may be measured and compared with equivalent non-reconfigurable IMA.

## REFERENCES

P. Bieber, F. Boniol, M. Boyer, E. Noulard, and C. Pagetti. *New challenges for future avionics architectures. Journal AerospaceLab*, vol. 4, 2012.

R. Wolfig and M. Jakovljevic. *Distribuited IMA and do-297: Architectural, communication and certification attributes*. Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th, pp. 1.E.4-1-1-1.E.4-10, Oct 2008.

M. Segvic, K. Krajcek and E. Ivanjko. A Proporsal for a Fully Distributed Flight Control System Design. MIPRO/CTS. 2016.

B. Andrillon and D. Aviation. *Contribution of integrated modular avionics of second generation for business aviation*. 2013.

C. Bobda. *Introduction to Reconfigurable Computing: Architectures, Tools and Applications*. Springer, 2007.

*Vivado Design Suite User Guide: Partial Reconfiguration*. UG909 (v2015.1) April 1, 2015.

*PRC: Partial Reconfiguration Controller V1.0. LogiCORE IP Product Guide. Vivado Design Suite*. PG193 April 6, 2016.

*AXI HWICAP v3.0. LogiCORE IP Product Guide. Vivado Design Suite*. PG134 November 18, 2015.

*JTAG to AXI Master v1.1. LogicCORE IP Product Guide. Vivado Design Suite*. PG174 June 8, 2016.

AXI EMC v3.0. LogiCORE IP Product Guide. Vivado Design Suite. PG100 November 18, 2015.