

Hardware/Software Co-Design of an Avionics Communication Protocol Interface System: an Industrial Case Study

François Clouté

Laboratoire d'Electronique LEN7
ENSEEIH, 2 rue Camichel
31071 Toulouse Cedex 07, France
33.5.61.58.84.36

cloute@len7.enseeiht.fr

Jean-Noël Contensou

Laboratoire d'Electronique LEN7
ENSEEIH, 2 rue Camichel
31071 Toulouse Cedex 07, France
33.5.61.58.82.84

contenso@len7.enseeiht.fr

Daniel Esteve

Laboratoire d'Electronique LEN7
ENSEEIH, 2 rue Camichel
31071 Toulouse Cedex 07, France
33.5.61.58.84.11

esteve@len7.enseeiht.fr

Pascal Pampagnin

AEROSPATIALE Aéronautique
Direction Systèmes et Services
31060 Toulouse Cedex 03, France
33.5.61.18.38.40

pascal.pampagnin@avions.aerospatiale.fr

Philippe Pons

AEROSPATIALE Aéronautique
Direction Systèmes et Services
31060 Toulouse Cedex 03, France
33.5.61.93.01.70

philippe.pons@avions.aerospatiale.fr

Yves Favard

AEROSPATIALE Aéronautique
Direction Systèmes et Services
31060 Toulouse Cedex 03, France
33.5.61.93.55.55

yves.favard@avions.aerospatiale.fr

ABSTRACT

Hardware/Software co-design is not a new idea, since designers have been used to mixing programmable and specific hardware components for algorithms implementation. However, with the growing complexity of systems, a computer-aided co-design methodology becomes essential.

This paper presents an application of the avionics domain: the ARINC communication protocol interface system. The co-design approach is based on the POLIS framework, coupled with the Esterel specification language.

Keywords

Co-design, avionics, ARINC, Esterel, POLIS.

1 INTRODUCTION

The major constantly growing factor that limits the development of complex systems is not the silicon technology manufacture, but the lack of a system-level design methodology. The increasing widespread of embedded systems in the domains of vehicle, avionics, communication, *etc.*, emphasizes that need.

Unlike a general-purpose computer, an embedded system has to realize a well-defined set of specific tasks. The required specialization should alter minimally its flexibility, to get a maximum design re-use [6].

Thus an embedded system typically consists of some VLSI hardware components, like ASIC or FPGA, and software supported by standard programmable components, like RISC or DSP.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CODES '99 Rome Italy

Copyright ACM 1999 1-58113-132-1/99/05...\$5.00

In a modern commercial aircraft, the avionics (*i.e.* the set of on-board hardware and software electronics equipment) consists of about a hundred of computers communicating between them and the environment. Each of these computers is dedicated to a specific avionics function. Such critical systems require a certified development.

Currently, the design of an embedded system is not optimal:

1. the system specification is written in a natural language, eventually without abstraction of architectural details;
2. the architectural decisions are made a priori, following the architect's experience or/and the past product versions;
3. the hardware and software parts are developed too separately;
4. the software is tested only after the hardware/software integration on a real prototype;

An hardware/software co-design methodology aims at solving all those issues [5, 7, and 10]. Co-design is defined as a methodology for designing software and hardware concurrently, thus reducing the design time and time-to-market. Hardware/software co-design of embedded systems includes co-specification, hardware/software partitioning, architecture selection, co-synthesis and co-verification.

There are different approaches of co-design, related to the type of the target applications. The taxonomy of embedded systems distinguishes two main domains: control-oriented and data-dominated applications.

In data-flow applications, *e.g.* digital signal processing, the behavior of the system is scheduled at a fixed rate, and the main complexity of the design comes from the mathematical operations on data. In control-oriented reactive applications, the system reacts continuously to the environment. Then, the monitoring of

the different tasks is crucial, especially as there are real-time constraints [2, 12]. The distinction is trivial, since very complex systems deal with both. But designing them requires a separate point of view.

This paper describes the hardware/software co-design of an avionics embedded control-dominated system: the ARINC protocol interface system. The next section provides some background about the ARINC protocol interface. Section 3 considers the system specification with an Esterel overview. Section 4 presents the POLIS co-design approach. Section 5 highlights some experimental results about the hardware/software design space exploration. Section 6 concludes and discusses future work.

2 THE AVIONICS ARINC PROTOCOL INTERFACE

In a modern commercial aircraft, the avionics consists of about a hundred of computers communicating between them and the environment.

The ARINC (Aeronautical Radio Inc.) is an international standard which specifies the communication protocol between the different embedded systems on board. Thus embedded systems designed by different manufacturers can communicate in the same aircraft. The standard protocol defines the type of the data frames and the exchange format of those data. However, the requirements do not force the implementation.

The ARINC protocol is a serial communication protocol with a rate of 100 Kbits per second. Data packets are 32 bits, added to 4 bits for the synchronization. A data packet consists of an 8-bit identification field, a 23-bit data field, and one parity bit. An ARINC bus is a set of channels, each carrying data packets.

The ARINC interface system is in charge of the acquisition of the data packets received on several parallel input channels. For each channel, after the synchronization bits, the recognition and the parity checking, the message is received. For each message true to the ARINC pattern, an address is computed from the identification field to store the data field and dating information. A pre-programmed memory is used for the addressing. Concurrently, the environment asynchronously requests the ARINC interface to return the available data.

The ARINC interface system represents a critical real-time embedded system, both with a complex control based on data values and soft/hard timing constraints.

The use of a co-design methodology aims at providing a rigorous design, ended in a final prototype with an adequate hardware/software architecture. The methodology we used is supported by POLIS [1], a co-design framework developed at the Berkeley University. The specification language is Esterel [3], a reactive synchronous programming language from the INRIA Institute of Sophia-Antipolis, France.

3 THE ESTEREL SPECIFICATION

Esterel is a textual, imperative, synchronous language, oriented towards the specification of control-dominated reactive systems.

Programming in Esterel is facilitated by a concurrent and modular decomposition, and an explicit definition of the control by the use of program constructs for concurrency, pre-emption, and exception. Unlike other synchronous languages [8], like Lustre [9] or Signal [11] dedicated to the computational systems, Esterel

is restricted to the addition of a header file in C for the definition of procedures or functions.

In Esterel, the basic element is the signal, valued or not, emitted by the system or the environment, and at the same time received, according to the synchronous semantics. Time is a multiform concept, based upon the nature of the signals (time, distance, temperature, etc).

The functionality of the ARINC interface was decomposed into 9 different modules, with some eventually instantiated more than once (*i.e.* PACKET CONTROL and RAF). The first step was to write each module in Esterel and to verify it by simulation with a graphical debugger.

The figure 1 presents the functional decomposition of our model of the ARINC interface system.

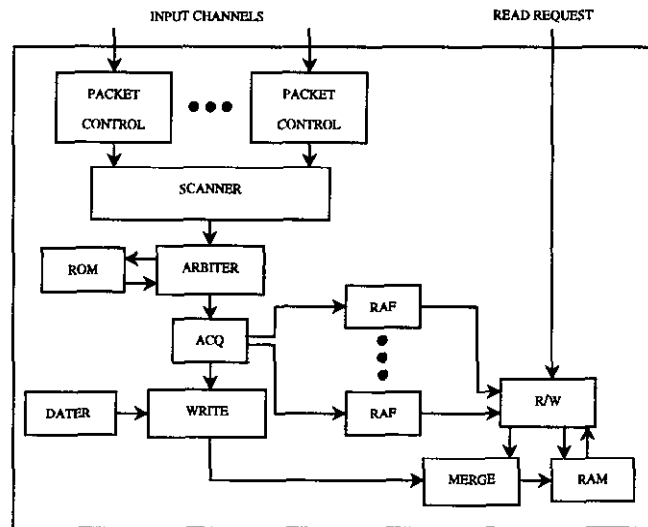


Figure 1. Model of the ARINC interface system

The data packets coming from each channel are detected, tested with respect to the ARINC pattern in each concurrent module PACKET CONTROL. Any accepted packet awakes the process of the module SCANNER, which under some conditions, enables the access to the pre-programmed memory with a priority mechanism fixed by the module ARBITRER. The module ACQ performs the storage addressing, annotated with both dating coming from DATER, and other information which commands the modules RAF, WRITE, R/W and MERGE to make the relevant processing to the data in RAM. The module R/W reacts also to any read request from the environment.

For example, the code shown in figure 2 gives a part of an Esterel module. The reactive behavior is an infinite loop that tests the presence of the signal START_RAF. Inside the loop, a down counting is performed from the last value of the signal START_RAF, with an exception handling if the internal variable VALEUR is equal to zero. The corresponding trap maintains a signal BIT_DEFAULT_DE_RAF emitted, unless a new signal START_RAF is received.

```

...
%loop
every START_RAF do
  var VALEUR : integer in
    %initialization
    VALEUR := ?START_RAF;
    %exception handling
    trap DEFAULT_DE_RAF in
      every CLK_20MS do
        VALEUR:=VALEUR-1;
        if VALEUR=0 then exit
DEFAULT_DE_RAF; end if
      end every
    handle DEFAULT_DE_RAF do
      sustain BIT_DEFAULT_DE_RAF;
    end trap
  end var
end every
end module

```

Figure 2. Esterel code

We can compile an Esterel program into a deterministic and sequential finite-state-machine. The table 1 shows the complexity of the ARINC interface, by analogy with the finite-state-machine parameters.

Modules	Instances	States	Functions	Signals	Variables	Actions	Halts	Calls
ctrl packet	4	5	6	10	16	35	4	17
scanner	1	34	0	11	7	13	10	6817
arbiter	1	3	7	26	46	85	2	546
acq	1	3	0	11	12	6	2	28
date	1	3	0	2	3	7	2	14
raf	2	5	0	5	4	7	4	24
i/w	1	10	1	10	13	21	6	972
write	1	3	1	6	10	10	2	11
merge	1	3	0	6	10	10	2	15

Table 1. Functional decomposition of the complexity of the ARINC interface system

A top-level description was specified in Esterel, with the acquisition of four input channels, and a simple model of pre-programmed memory which infers two storage addresses. The Esterel synchronous specification was verified by simulation. For giving an approximate idea of the system complexity, the generation of one single equivalent finite-state-machine could not normally terminate under a SPARC IPX Station with 32 Mbytes of ROM. Nevertheless, the generation of a sorted circuit code in C is the right alternative for an all software implementation.

As the final ARINC interface system has to cope with up to 80 input channels concurrently, hardware should be essential to meet the timing constraints. Thus our Esterel modules were entered in the POLIS co-design flow.

4 THE POLIS CO-DESIGN ENVIRONMENT

The POLIS co-design framework is oriented towards control-dominated reactive embedded systems, with a generic target architecture composed of one microcontroller and some hardware coprocessors.

The POLIS environment is based on a formal model called Co-design Finite State Machine (CFSM). A network of CFSMs, with an asynchronous communication model between CFSMs

represents a system. Each CFSM is specified by an Esterel module, locally synchronous.

The POLIS design flow is illustrated by the figure 3, and the main steps are described below:

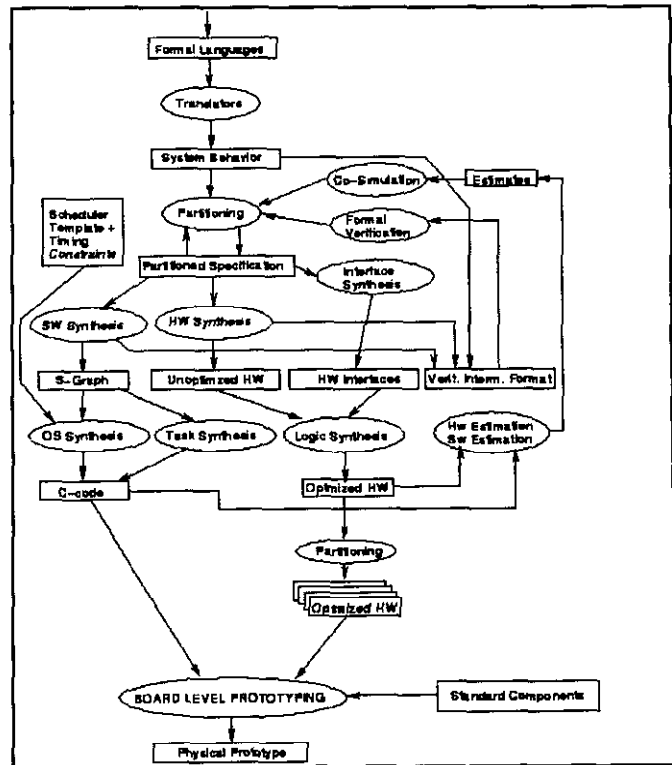


Figure 3. The POLIS system

1. Translation of the system-level language like Esterel into the CFSM model;
2. Formal verification of the specification after the translation of a CFSM into a finite-state-machine formalism;
3. Manual hardware/software partitioning. The granularity level is the CFSM;
4. Hardware/software co-simulation based on the Ptolemy simulation framework [4]. Related to the hardware/software partitioning, the microprocessor selection, and the scheduler selection, the architectural trade-offs are explored and evaluated, relying on code size and performance estimates of the processor;
5. Hardware synthesis of the CFSM sub-network by mapping into the BLIF (Berkeley Logic Intermediate Format) format. Each transition function is a combinational circuit, optimized by logic synthesis techniques, and states variables are implemented by registers. An XNF netlist can be generated to get a FPGA Xilinx prototype;
6. Software synthesis of the CFSMs sub-network into a C code structure which includes one procedure for each CFSM and a real-time operating system;
7. Synthesis of the interfaces between the different implementation domains: hardware, software, and the environment.

5 THE EXPLORATION OF THE HARDWARE/SOFTWARE DESIGN SPACE

The Esterel specification of the ARINC interface was modified to be used as a front-end language in the POLIS system, as regards to its asynchronous communication model. We altered the code of some CFSMs that react to events coming from at least two distinct CFSMs, in order to get a correct scheduling. Moreover, the MERGE module was added. The Ptolemy graphical user interface enables to connect the CFSMs, before functional simulation.

POLIS/Ptolemy provides to the user a rich library of components for the test bench. Many simulation scenarii were applied in the Ptolemy debug mode. The ARINC interface system with four input channels and two-storage index was verified by functional simulation.

The performance analysis of the system is the key to select an architecture that meets the timing constraints. The performance simulation relies on the C generated models of each CFSM, the hardware/software partitioning, the scheduling policy chosen for the operating system, and the timing and cost model of the processor. The user can choose a scheduler between a static round robin, a static priority with preemptive, or a static priority non-preemptive mechanism.

We present in the table 2, for each module and for the whole system, the estimated results of the code size in bytes, and the minimum and maximum number of execution cycles of the selected processor. The two 32-bit microcontrollers are the MIPS R3000, and the Motorola 68332.

Module	MIPS R3000			Motorola 68332		
	min time	max time	code size	min time	max time	code size
ctrl packet (x4)	27	436	1613	43	907	1648
scanner	30	279	10102	55	1100	8790
arbiter	39	495	2237	124	983	2419
acq	38	162	326	112	496	243
dater	28	125	222	66	290	163
raf (x2)	27	148	452	43	378	367
r/w	27	251	1780	43	832	1568
write	27	83	207	64	277	181
merge	25	79	227	54	296	162
whole system	254	1227	15849	/	/	/

Table 2. Performance and cost estimated results.

The system must both perform the data acquisition of each input channel at a speed of 10 μ s, and respond to the asynchronous read request at a minimum interval of 6 μ s. The hard timing limit for the return of a data is 3 μ s. We performed worst case simulations, *i.e.* with a maximum rate of true ARINC packets, concurrently, over the four channels.

We showed that an architecture with an all-software implementation on a MIPS R3000 would require a clock frequency of 230 MHz, so as not to miss deadlines. Such a microcontroller do not exist.

A more realistic architecture with the four modules (called CONTROL PACKET, controlling the acquisition of ARINC data packets over each channel) mapped to hardware, a MIPS R3000 at a clock frequency of 133 MHz, and a static priority preemptive scheduler, the timing constraints were met.

6 CONCLUSION AND FUTURE WORK

This paper presented the hardware/software co-design of an industrial example: the avionics ARINC interface system. The first step of our work was to get a system-level executable specification, abstracting the implementation details.

Programming in Esterel requires a conceptual approach different from a traditional mono-thread sequential language. The synchronous Esterel modules were modified with respect to the POLIS model used for distributed systems.

The POLIS system is convenient for control-dominated systems. The future work will consist of extending the POLIS library with another microcontroller model with better performance. The design space exploration at the system-level could also include the memory. A double port RAM was used by now, without studying the effect of other types of memory. Finally, the co-synthesis and the prototyping of a software/hardware architecture of the ARINC interface system with the POLIS co-design flow represent another work axis.

7 ACKNOWLEDGEMENTS

We would like to thank all the members of the Internet groups of Esterel and POLIS for their precious help.

8 REFERENCES

- [1] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, et B. Tabbara. "Hardware-Software Co-Design of Embedded Systems, The POLIS approach" Kluwer Academic Publishers, 1997.
- [2] F. Balarin, L. Lavagno, P. Murthy, and A. Sangiovanni-Vincentelli. Scheduling for Embedded Real-Time Systems, IEEE Design & Test of Computers, pp. 71-82, January 1998
- [3] G. Berry, G. Gonthier. The Esterel Synchronous Programming Language: Design, Semantics, Implementation, Science of Computer Programming Vol. 19, N°2, pp. 87-152, 1992.
- [4] J. Buck, S. Ha, E.A. Lee, and D.G. Masserschmitt. Ptolemy: a framework for simulating and prototyping heterogeneous systems. International Journal of Computer Simulation, special issue on Simulation Software Development, January 1990.
- [5] R. Ernst. Codesign of Embedded Systems: Status and Trends, IEEE Design & Test of Computers, pp. 45-54, Avril 1998
- [6] R. Ernst. Target Architectures, in W. Wolf and J. Staunstrup Hardware/Software Co-Design: Principles and Practice, Kluwer Academic Publishers, 1997.
- [7] D. Gajski, F. Vahid, S. Narayan, et J. Gong. Specification and Design of Embedded Systems, Prentice Hall, 1994.
- [8] N. Halbwachs. Synchronous Programming of Reactive Systems, Kluwer Academic Publishers, 1993.
- [9] N. Halbwachs, P. Caspi, and D. Pilaud. The Synchronous Dataflow Programming Language Lustre. Another look at Real Time Programming, Proceedings of the IEEE, Special Issue, September 1991.
- [10] J.-M. Laporte. Hardware/software Codesign Study Report, Annual Conference of European Multimedia, Microprocessor System and Electronic Commerce MSEC'97, November 1997.

- [11]P. Le Guernic, M. Le Borgne, T. Gauthier, and C. Le Maire. Programming Real-Time Applications with Signal. Another look at Real Time Programming, Proceedings of the IEEE, Special Issue, September 1991.
- [12]C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM, Vol.20, N°1, pp. 46-61, January 1973.