e-Informatica Software Engineering Journal, Volume 5, Issue 1, 2011, pages: 39–49, DOI 10.2478/v10233-011-0029-x
VERSITA

ARINC Specification 653 Based Real-Time Software Engineering

Sławomir Samolej*
*Faculty of Electrical and Computer Engineering, Rzeszow University of Technology
ssamolej@prz.edu.pl

Abstract

This paper reports successive steps of a real-time avionic pitch control application creation. The application structure follows a new real-time systems development profile published in ARINC specification 653. The paper mentions some main ARINC specification 653 features and shows the subsequent application creation levels: control system units distribution, timing requirements definition, application implementation and tests. It describes the author's experience gained during an avionic hard real-time system development and focuses on real-time software engineering details of the application creation.

1. Introduction

Real-time applications gradually evolve form simple one-task embedded programs to large multi-task and distributed systems. Modern large real-time applications are usually based on real-time operating systems (such as VxWorks [1], PikeOS [2], LynksOS [3], WindowsCE [4], and Linux RTAI [5]), or are written in real-time languages (Ada 2005 [6] or Spakr [7]). These applications should be developed according to well defined practical rules expressed e.g. in [8, 9]. Real-time tasks are given priorities according to a predictable policy (such as Rate Monotonic or Deadline Monotonic Policies). Inter-task communication protocols prevent the system from deadlocks and unpredictable delays during execution (by Priority Inheritance or, if possible, by Priority Ceiling Resource Access Protocol application). The data exchange between distributed applications should be predictable. Naturally, such rules can be applied in modern real-time operating systems APIs or in Ada language, but less advanced developers often encounter problems, especially in case of complex software.

To make the real-time system development less cumbersome, several groups of experts have developed some practically applicable real-time design patterns. Among the set of well defined code-generation oriented design patterns the POSIX standard [10], HOOD [11] and HRT-HOOD [12] methods and Ada Ravenscar Profile [13] can be indicated. They define good real-time system programming techniques and are partly supported by some automatic real-time software code generators. Subsequently, the real-time design patterns have been integrated with dedicated software toolkits such as Matlab-Simulink [14], SCADE SUITE [15] or IBM Rational Rose RealTime [16]. These tools allow to design graphically the real-time software, giving as the output a real-time system structure (IBM Rational Rose), both structure and selected algorithms implementation (SCADE SUITE), or complete real-time application code and environment generated for a specific hardware (Matlab-Simulink). Moreover, SCADE SUITE enables to develop a complete hard real-time system source code that conforms to international safety standards DO-178B (up to level A for

Military and Aerospace Industries), IEC 61508 (at SIL 3 by TÜV for Heavy Equipment, and Energy), EN 50128 (at SIL 3/4 by TÜV for Rail Transportation), and IEC 60880 (for Nuclear Energy).

It is also worth noting that international standardization organizations have recently published some new standards or specifications regarding modern real-time systems development. The ARINC (AERONAUTICAL RADIO, INC) specification 653 [17] or the SAE (Society of Automotive Engineers) Standard AS5506 [18] are examples of such documents. To the author's opinion the new documents bring a new quality in real-time systems development. They provide a new abstraction layer in the real-time software design process which makes possible to create complete large distributed real-time systems. This paper deals with the ARINC specification 653 (ARINC 653) based hard real-time systems development.

The ARINC 653 was developed by aviation experts to provide "the baseline environment for application software used within Integrated Modular Avionics (IMA) and traditional ARINC 700-series avionics". It is closely connected with the Integrated Modular Avionics (IMA) concept [19, 20, 21]. Primary objective is to define a general purpose APEX (APplication/EXecutive) interface between Operating System (O/S) of the avionics computer and application software.The specification includes interface requirements between application software and O/S and list of services which allow the application software to control scheduling, communication, and status of internal processing elements. Over a period of 6 years from the specification announcement:

- the ARINC 653 based software has been implemented in A380, A400M and B787 airliners,
- at least three commercial real-time operating systems (WxWorks 653, PikeOS, LynksOS-178 RTOS) have been updated to offer the APEX,
- four European-founded and focused on the ARINC 653 – based software development research projects (PAMELA, NEVADA, VICTORIA and SCARLETT) have been created.

This paper describes some details concerning real-time programming rules included in ARINC 653. A Pitch Control Application created within SCARLETT [22] project by Research Group from Rzeszów University of Technology (RGRUT) illustrates the ARINC 653 based development process. The following sections are organized as follows. At first, the ARINC 653 is introduced. Secondly, the Pitch Control Application development is presented. The final part describes the future RGRUT's research and implementation plans.

2. ARINC specification 653

Airborne real-time systems have been evolving from the so-called "federated" structure to Integrated Module Avionics (IMA) [19, 20, 21]. The IMA concept has been introduced in European funded research projects, PAMELA, NEVADA and VICTORIA. The result of the projects was the first generation of IMA (IMA1G), currently on-board of A380, A400M and B787 aircraft. Following the IMA concept, modern on-board avionic subsystems (software applications) are grouped in a limited set of standard microprocessor units. The units and other electronic devices communicate via standard network interface – Avionics Full Duplex Switched Ethernet (AFDX) [23, 24]. The group of federated applications executed until now on separate microprocessor units (and communicating by means of ARINC standard 429 based devices [25], for example) must become a set of real-time processes executed on one hardware module. This module will be managed by a dedicated real-time operating system. Provided that the operating system offers a standard API and fulfills safety requirements, such solution significantly broadens portability of avionic applications and allows to develop and certify hardware and software independently. Current implementation of IMA covers limited range of aircraft functions but shows some significant benefits, i.e. aircraft weight reduction and lower maintenance costs.

2.1. Partitions

The IMA assumes that a set of time-critical and safety-critical real-time applications (avionics units) may be executed on one microprocessor module. To cope with this level of criticality, new real-time operating system architecture has been suggested. ARINC 653 [17] defines generic structure of the system. Figure 1 shows the logical structure of RTOS suggested in it.

The key concept introduced in the specification is the partition. It creates a kind of container for an application and guarantees that execution of the application is both spatially and temporally isolated. The partitions are divided into two categories, application partition and system partition. The application partitions execute avionics applications. They exchange data with the environment by means of specific interface – APEX (APplication/EXecutive). The system partitions are optional and their main role is to provide services not available in APEX, such as device drivers or fault management.

2.2. Hardware-Software Module Architecture

The ARINC 653 also includes some recommendations on microprocessor module architecture for the dedicated real-time operating system. General diagram of the architecture is presented in figure 2. Each module includes one or more microprocessors. The hardware structure may require some modification of core operating system but not the APEX interface. All processes that belong to one application partition (real-time tasks) must be executed on one microprocessor. It is forbidden to allocate them to different microprocessors within the module or split them between modules. The application program should be portable between processors within the module and between modules without any modifications of the interface to operating system core. Processes that belong to one partition may be executed concurrently. A separate partition-level scheduling algorithm is responsible for this. Inter-application (partition) communication is based on the concept of ports and channels. The applications do not have the information at which partition the receiver of data is executed. They send and receive data via ports. The ports are virtually connected by channels defined at separate level of system development.

Temporal isolation of each partition has been defined as follows. A major time frame, activated periodically, is defined for each module. Each partition receives one or more time partition windows to be executed within this major time frame. Generally, time partition windows constitute a static cyclic executive [9]. Real-time tasks executed within the partition can be scheduled locally according to priority-based policy. The order of the partition windows is defined in a separate configuration record of the system.

Health Monitor (HM) is an important element of the module. HM is an operating system component that monitors hardware, operating system and application faults and failures. Its main task is to isolate faults and prevent failure propagation. For example, the HM can restart a partition when detects application fault.

By assumption, the applications (or partitions) may be developed by different providers. Therefore an integrator of the IMA system development process is necessary. This person collects data regarding resources, timing constraints, communication ports and exceptions defined in each partition. The collected data are transferred into configuration records. The configuration record for each module is an XML document interpreted during compilation and consolidation of software.

2.3. APEX Interface

APEX (APplication/EXecutive) interface definition is the main part of ARINC 653. The APEX specifies how to create platform-independent software that fulfills ARINC 653 requirements. Main components of the interface are:

- partition management,
- process management,
- time management,
- memory management,
- interpartition communication,
- intrapartition communication,

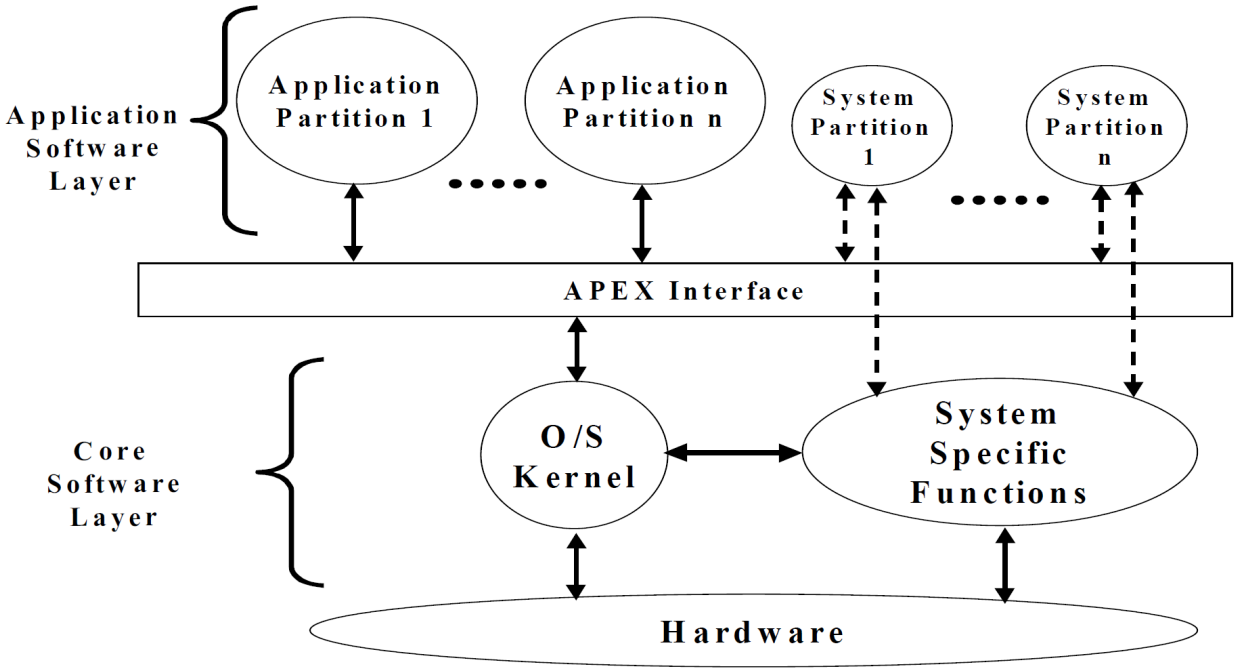

Figure 1. Logical real-time operating system structure created according to ARINC specification 653 ([17], pp. 4).

– health monitoring.

The APEX interface provides separate set of functions enabling the user to determine actual partition mode and change it. The application may start the partition after creation of all application components. It is also able to obtain the current partition execution status. Interpartition health monitoring procedures can stop or restart the partition.

The application may be constructed as a set of (soft or hard) real-time processes, scheduled according to priorities. APEX process management services can:

– create process and collect process status or ID,
– start, stop, suspend or resume process,
– prevent from process preemption,
– change the process priority.

The APEX manages both aperiodic and periodic processes. Periodic processes are activated regularly. Additionally, a separate parameter called "time capacity" is attached to each of them. It defines time frame (deadline) within which single instance of task must finish computations. When a process is started the deadline is set to current time plus time capacity. The operating system periodically checks whether the process completes processing within the allotted time. Each process has a priority. During any rescheduling event the O/S always selects for the execution the highest priority process in "ready" state.

There are no memory management services in APEX because partitions and associated memory spaces are defined during system configuration. The ARINC 653 assumes, for safety reasons, that the whole memory is statically allocated to partitions and processes before the partition or application starts. It is expected that memory space is checked either at build time or before running the first application.

Interpartition (inter-application) communication is based on queuing port and sampling port communication units. The queuing port provides interpartition message queue, whereas the sampling port shares variables between the ports. During system integration, the ports are connected by means of channels defined in system configuration tables. The ports communicate with other partitions or device drivers within the

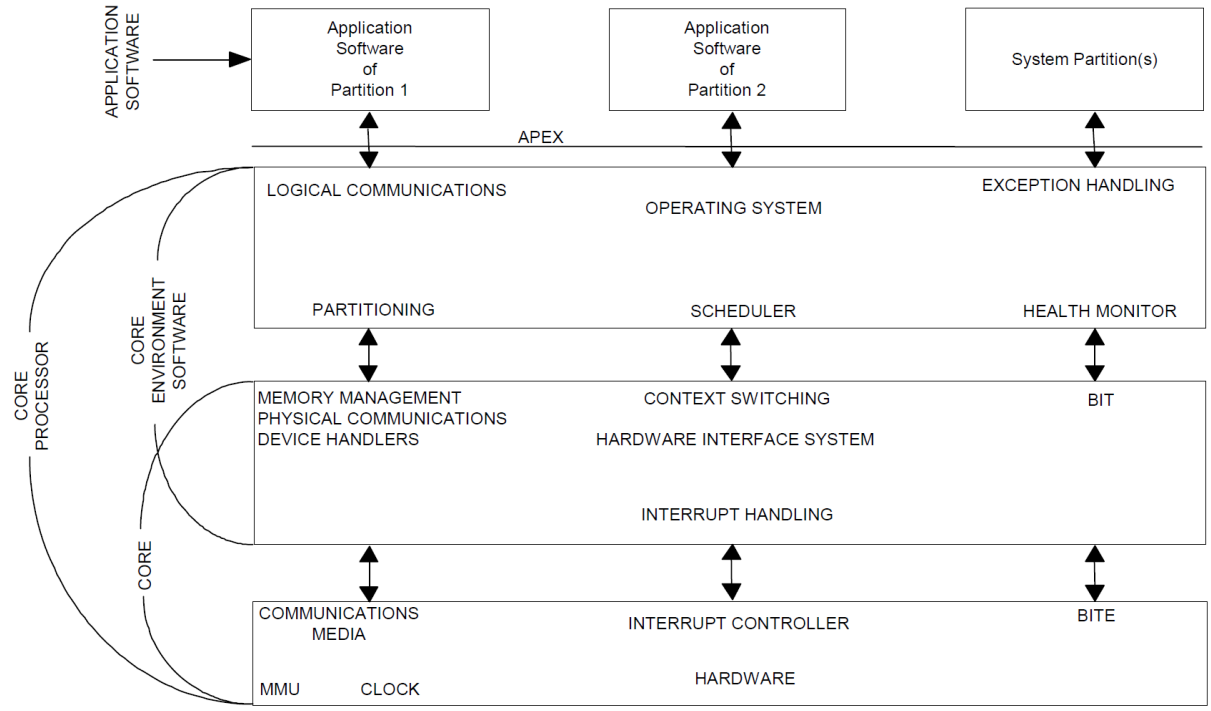

Figure 2. Hardware-software architecture of typical module according to ARINC 653 specification ([17], pp. 11).

module, or exchange data between modules (by means of AFDX network interfaces).

The synchronization of processes belonging to one partition may be achieved by appropriate application of counting semaphores and events. The inter-process communication within the partition (intrapartition communication) is implemented by means of APEX buffers (shared message queues) and APEX blackboards (shared variables). It is possible to define a time-out within which process waits for the data, if not available immediately. The process may be blocked for the specified time only.

The ARINC 653 Health Monitor is an advanced exception handing engine. Three error levels are defined:

- Process Level which affects one or more processes in the partition,
- Partition Level with only one partition affected,
- Module Level which affects all partitions within the module.

Both Partition Level and Module Level errors are handled by a set of procedures installed by the system integrator. The Process Level errors can be handled by the programmer. Separate sporadic task called "error handler" can be registered for each of the partitions. When the Health Monitor detects an error at the process level it calls the error handler. The handler recognizes the error and, depending on the error, can:

- log it,
- stop or restart the failed process,
- stop or restart the entire partition,
- invoke the registered error handler process for the specific error code.

To the author's knowledge four real-time operating systems meet ARINC 653 requirements, i.e. Wind River VxWorks 653 [1], Sysgo PikeOS [2], LynuxWorks LynxOS-178 RTOS, and LynuxWorks LynxOS-SE RTOS [3].

3. ARINC 653 based Pitch Control System Development

This section describes an ARINC 653 based sample avionics subsystem development. The objec-

tive is to create a distributed hard real-time application to control a pitch angle of typical airliner (Pitch Control Application – PCA). Subsequent development steps are: control system definition, control procedures allocation to hardware units, timing requirements assessment, application structure design, system programming, testing. Preliminary version of the PCA has been proposed in [26]. Papers [27] and [28] report on stages of PCA development and some extensions to detect control system malfunctions.

3.1. Control System Definition

The system controls two actuators (brushless motors) connected to a load (elevator). Each actuator is controlled by separate cascade of controllers shown in figure 3. The single actuator control system includes internal current control loop, velocity control loop (PID2, PID4), and position control loop (PID1, PID3). The Flight Control Algorithm (FCA) is a supervisory module that generates position demand signal for both actuator control subsystems. It collects signals from the pilot, aircraft, and actuators. The reference signal is a force or shift of control side-stick moved by the pilot. The FCA module corrects the reference signal using the actual aircraft pitch angle. The PCA includes also Error Estimator that collects some control signals and estimates quality of control system during runtime. It also produces separate output for system operator. The entire Pitch Control Application has been modeled in Matlab-Simulink [14] software toolkit. All control procedures and settings have been designed following control engineering rules.

3.2. Distribution of Control Procedures

The Pitch Control Application has been developed according to a set of restrictions formulated by the system integrator. One of the restrictions requires selected control procedures to be allocated on different hardware modules. Figure 4 illustrates the desired PCA control modules allocation. Hardware Module 1 (HM1) includes the FCA, Error Estimator and position controllers (PID1 and PID3, compare fig. 3). Hardware Modules 2 and 3 (HM2 and HM3) include velocity controllers (PID2 and PID4, compare fig. 3). The next restriction is that the FCA, Error Estimator and all the PID controllers should be software modules whereas the current controllers must be included into motor drives hardware. The hardware modules are connected via AFDX network. The network structure has also been imposed by the system integrator.

According to the requirements the hardware-software environment follows the IMA philosophy. Therefore the FCA and Error Estimator control blocks belong to one ARINC 653 based application partition, as two separate real-time tasks. All PID controllers are separate real-time tasks each of which belongs to separate application partition. The partition structure physically and temporally isolates the main control application subsystems. It also enables reallocation of PID control procedures between hardware modules, what is another application requirement.

3.3. Timing Requirements

The Pitch Control Application has been developed to fulfil the ARINC 653 real-time parameters shown in figure 5 and table 1. The time capacity (deadlines) and task periods have been chosen taking into account both actuator dynamics and computing power of hardware units. The partition including the FCA and Error Estimator periodic real-time task acquires 6 [ms] time slot and 20 [ms] deadline. It is executed in two 3-millisecond time frames (fig. 5). The partitions with PID1 and PID3 control procedures are activated every 5 [ms] and executed within 1 [ms] time frame. Similarly, PID2 and PID4 real-time tasks are activated every 5 [ms], but their deadlines are extended to 2[ms] since Hardware Modules 2 and 3 provide lower computing power than Hardware Module 1. The major time frames in figure 5 include some “System” slots. These slots may be used by other software modules loaded on hardware. The HARD attribute attached to each of the real-time tasks instructs the Health Monitor (built into the operating system) that if any task misses its deadline, the core operat-

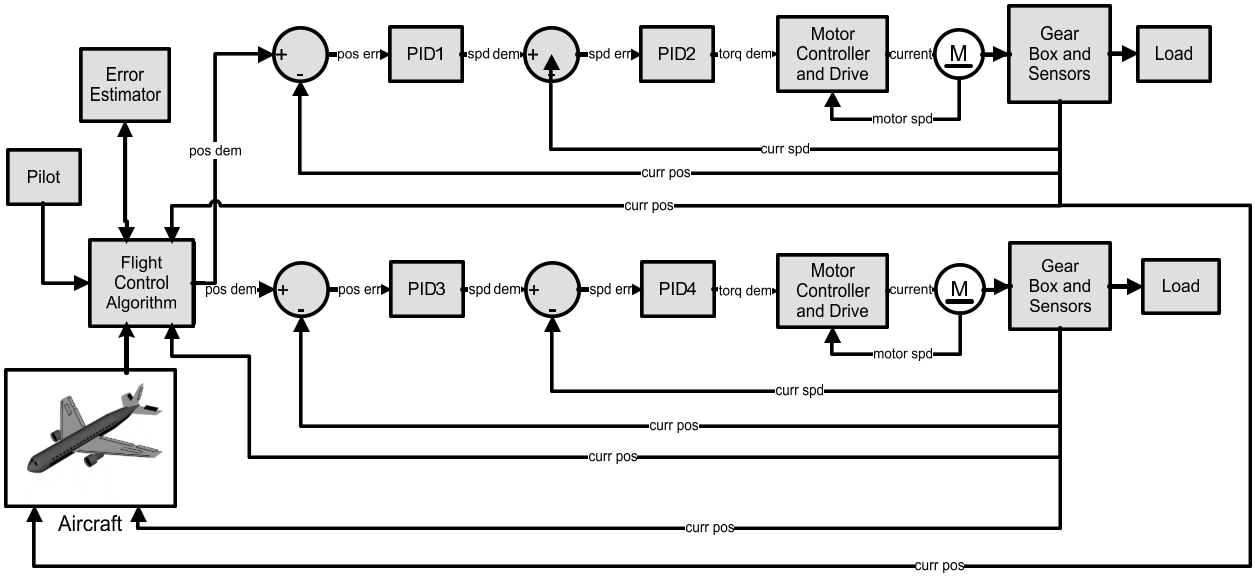

Figure 3. Pitch control system architecture

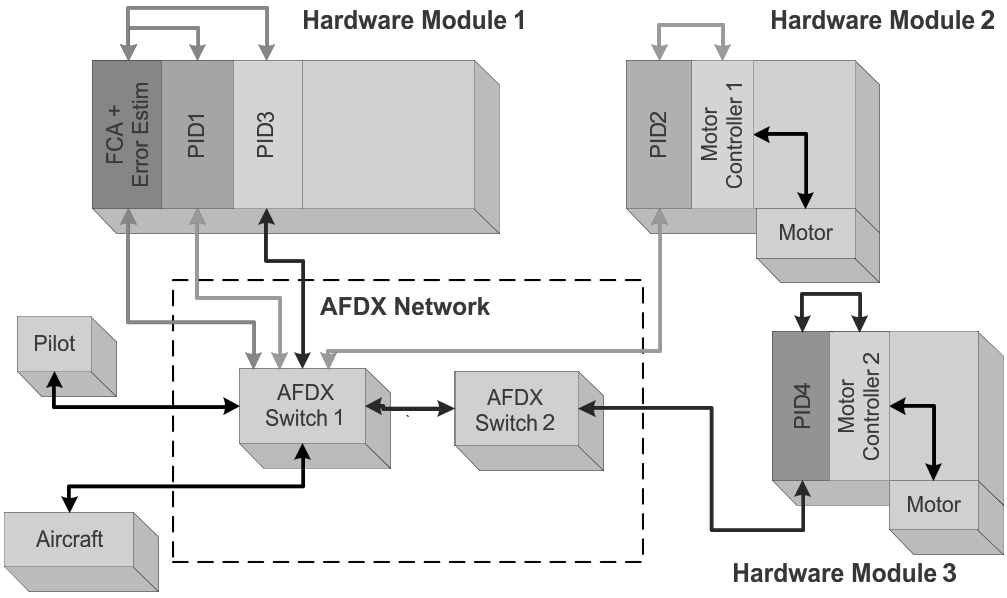

Figure 4. Allocation of Pitch Control System procedures on the Hardware Units

ing system must be informed. In consequence, this forces the operating system to take appropriate action. The Health Monitor procedures may even reload the whole partition that misses timing constraints. It has been assumed that the maximum communication delay in the AFDX network should not exceed 7 [ms].

All of the algorithms applied in the PCA are either controllers or simplified numerical procedures which solve some differential equations. During the development the worst case computing time for each of the algorithm has been evaluated. Experimental checks have proved that the algorithms meet their timing constraints.

As mentioned before, the PCA timing restrictions have been provided by control engineers who specified the system. P2 and P3 partitions acquire 1 [ms] time frames for computations and are activated every 5 [ms]. This guarantees sufficient frequency (200Hz) of PID algorithm repetition, so sufficient quality of control. P1 partition is 4 times "slower" than others without adverse effect

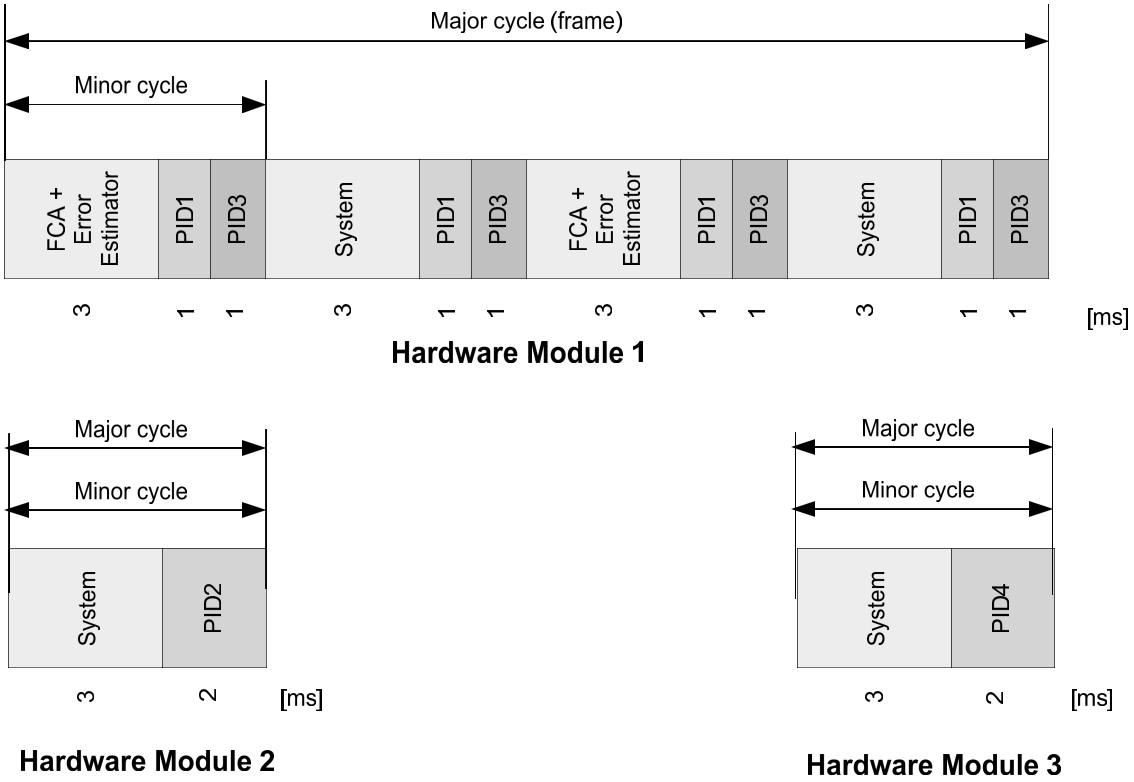

Figure 5. PCA timing requirements

Table 1. PCA real-time tasks parameters

Control Procedure	Stack Size	Base Priority	Period	Time Capacity	Deadline
FCA	4096	18	20	20	HARD
Error Estimator	4096	17	20	20	HARD
PID1	4096	20	5	1	HARD
PID3	4096	20	5	1	HARD
PID2	4096	20	5	2	HARD
PID4	4096	20	5	2	HARD

on quality of control. This preserves some computational time for other applications installed on the same hardware module.

3.4. Application Structure

The Pith Control Application structure is shown in figure 6. Apart from control procedures allocation strategy of figure 4, figure 6 shows the partitions and the intra- and interpartition communication structure. The first (P1) partition loaded into the Hardware Module 1 includes two real-time tasks, the Flight Control Algorithm (FCA) and Error Estimator. The FCA task collects signals from Pilot, Aircraft and actuator modules and produces the desired pitch angle signals for position controllers (PID1, PID3). The Error Estimator monitors both communication channels and quality of control during runtime. The algorithms applied in the Error Estimator have been presented in [27, 28]. The second (P2) partition includes the first position controller algorithm (PID1), running as separate real-time task. Identically, the third (P3) partition includes the second position control algorithm (PID3). The fourth (System Partition) collects all signals exchanged between hardware modules and transfers them to the AFDX network. The Hardware modules 2 and 3 have the same hardware-software structure.

They include one system partition and one application partition. The application partitions (P4 and P5) involve single real-time tasks with speed control PID algorithm. The system partitions provide inter-hardware module communication.

For the intrapartition communication, ARINC 653 blackboards have been applied, whereas the interpartition communication is based on ARINC 653 sampling ports and channels. Blackboards and sampling ports seem most suitable communication units for the system, since they are in fact shared and protected data regions. They always provide the latest acquired data. It is possible to set their properties in such a manner that they do not block the real-time tasks, even if they are empty. It is assumed that some of data packages produced by "fast" control blocks may be lost due to the "slow" ones. From real-time software engineering point of view all communication mechanisms applied in the PCA are shared variables and monitors [9]. This solves the mutual exclusion problem. The shared variables are accessed (at the operating system level) according to Priority Inheritance Protocol [8, 9]. One can check that the PCA communication structure does not include deadlocks.

Due to "external" partition scheduling and simple application structure, the priorities of local task are used mostly for the precedence constraints definition. The tasks priorities (defined within the partitions – compare tab. 1) reflect the order of the computations, the tasks should follow. This approach is essential especially in P1 partition. It is expected that the FCA real-time task is terminated before the Error Estimator task begins.

4. System Programming and Testing

The PCA has been finally implemented in C language for VxWorks 653 [29, 30, 31] and PikeOS [32, 33, 34] operating systems, with APEX interface applied for PCA structure generation. From the programmer's point of view, each partition defined during the application development is a separate program. The program consists of a set of real-time tasks. The tasks exchange data by means of APEX blackboards or send and receive data via sampling ports. Timing constraints are attached to the tasks. Each task involves main function which collects data from the input communication objects (blackboards or sampling ports), calls appropriate control block algorithm, sends computed data to output objects, and finally suspends execution. The function is periodically activated by core operating system.

The PCA is a distributed real-time control application, so implementation tests have involved three main areas, i.e. communication, timing constraints and control algorithms. Firstly, the communication structure of the application has been assessed and all channels and data structures checked. Secondly, Worst Case Execution Time (WCET) for each of the real-time task has been measured and the meeting of timing constraints both at the process and the partition level evaluated. The measured time has been compared with partition time slots defined at the beginning. Thirdly, the control procedures applied in the application have been tested, both as separate function blocks and as complete set of cooperating software modules.

To the author's experience, good practice for the ARINC 653 programmer is to prepare working document that explains:

- ports introduced in the application,
- channels' description (how to connect the ports with other ports),
- time budget (partition window) allocated to the partition,
- time period within which the application (partition) should run again,
- memory requirements.

Some extra records with internal structure of the application will also help. Typically the application developer should provide:

- parameters of real-time tasks (IDs, stacks, priorities, deadlines, periods, criticalness <HARD/SOFT>),
- internal communication structure (blackboards and buffers, their IDs, capacities, tasks attached),

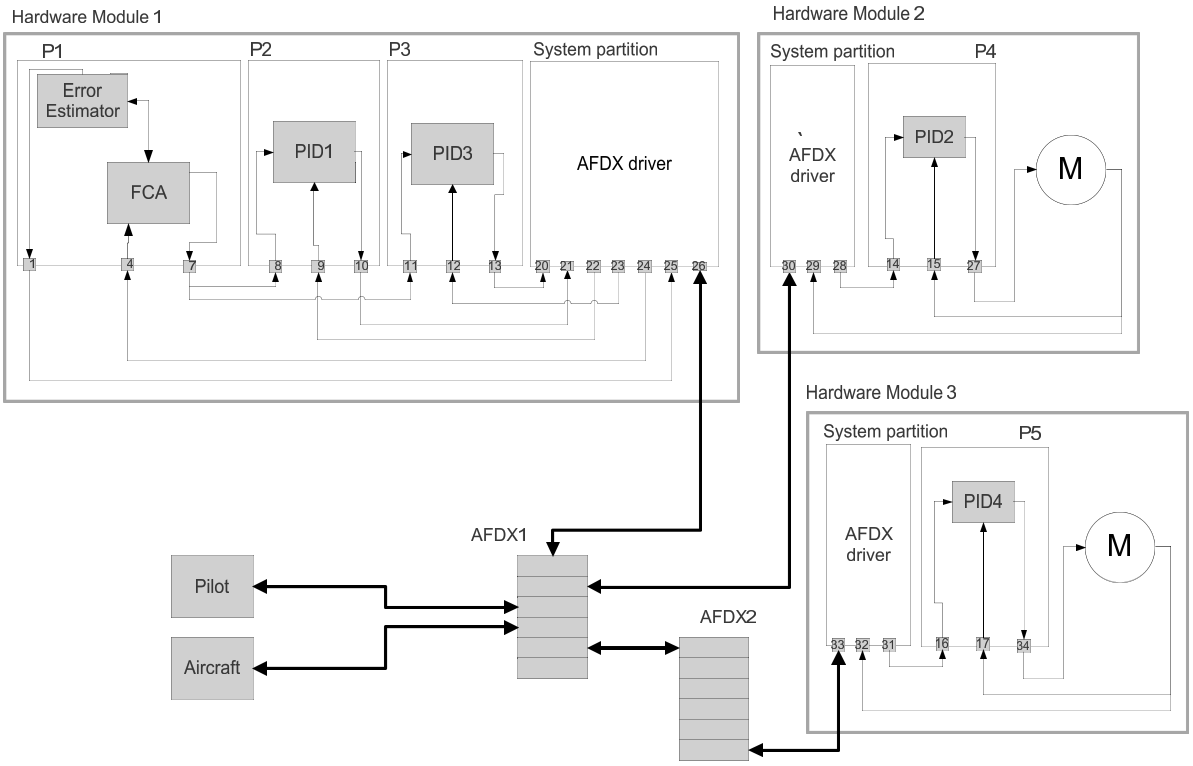

Figure 6. PCA structure

– communication timeouts (attached to APEX function calls to read data from blackboards or buffers),
– internal error handler procedure actions.

According to IMA philosophy the prepared applications (partitions) in a form of binary files equipped with such documents are sent to the system integrator.

5. Conclusions and Future Research

The paper reports emerging trends and design patterns in real-time systems development. It is focused on ARINC 653, where a new standard for real-time systems design is introduced. Successive steps of a typical ARINC 653 based application (Pitch Control Application PCA) development are described. The application has been designed as contribution of Rzeszów University of Technology to European SCARLETT [22] project. In the current state of the PCA development, the application is being integrated with other hardware and software modules delivered by SCARLETT partners. It is expected that the final PCA application test will reveal whether the current hardware-software environment conforming with ARINC 653 is able to execute some distributed hard real-time applications successfully.

6. Acknowledgments

Research reported in the paper is funded by SCARLETT 7th European Framework Project, Grant Agreement No. FP7-AAT-2007-RTD-1-211439. Some of hardware components used in the research published within this paper were financed by the European Union Operational Program – Development of Eastern Poland, Project No. POPW.01.03.00-18-012/09.

References

[1] Wind River WWW Site. [Online]. Available: http://www.windriver.com/
[2] SYSGO WWW Site. [Online]. Available: http://www.sysgo.com/

[3] Lynux Works WWW Site. [Online]. Available: http://www.lynuxworks.com/
[4] WindowsCE WWW Site. [Online]. Available: http://www.microsoft.com/
[5] Linux RTAI WWW Site. [Online]. Available: https://www.rtai.org/
[6] J. Barnes, *Programming in Ada 2005*. Addison-Wesley, 2006.
[7] ——, *High Integrity Software, The SPARK Approach to Safety and Security*. Addison-Wesley, 2003.
[8] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, 1997.
[9] A. Burns and A. Wellings, *Real-Time Systems and Programming Languages*. Pearson Education Limited, 2001.
[10] "IEEE Std 1003.1, 2004 Edition," IEEE, Tech. Rep., 2004.
[11] J.-P. Rosen, *HOOD an Industial Approach for Software Design*. Elsevier, 1997.
[12] A. Burns and A. Wellings, *HRT-HOOD: A structured design Method for hard Real-Time Ada Systems*. Elsevier, 1995.
[13] A. Burns, B. Dobbing, and T. Vardanega, "Guide for the use of the ada ravenscar profile in high integrity systems," University of York, Technical Report YCS-2003-348, Jan 2003.
[14] Matlab-Simulink WWW Site. [Online]. Available: http://www.mathworks.com/
[15] Scade Suite WWW Site. [Online]. Available: http://www.esterel-technologies.com/products/scade-suite/
[16] IBM Rational Rose RealTime WWW Site. [Online]. Available: http://www-01.ibm.com/software/awdtools/developer/technical/
[17] *Avionics Application Software Standard Interface Part 1-2, ARINC Specification 653P1-2*, 2005.
[18] *SAE AS5506 Standard: Architecture Analysis and Design Language (AADL)*, 2006.
[19] P. Bieber, E. Noulard, C. Pagetti, T. Planche, and F. Vialard, "Preliminary design of future reconfigurable ima platforms," in *ACM SIGBED Review - Special Issue on the 2nd International Workshop on Adaptive and Reconfigurable Embedded Systems*. ACM, Oct 2009.
[20] P. Parkinson and L. Kinnan, "Safety-critical software development for integrated modular avionics," Wind River, Wind River White Paper, 2007.
[21] J. W. Ramsey, "Integrated Modular Avionics: Less is More Approaches to IMA will save weight, improve reliability of A380 and B787 avionics," *Avionics Magazine*, 2007. [Online]. Available: http://www.aviationtoday.com/av/categories/commercial/8420.html
[22] SCARLETT Project WWW Site. [Online]. Available: http://www.scarlettproject.eu
[23] "AFDX: The Next Generation Interconnect for Avionics Subsystems," Avionics Magazine Tech. Report, Tech. Rep., 2008.
[24] *Aircraft Data Network Part 7 - Avionics Full Duplex Switched Ethernet (AFDX) Network, ARINC Specification 664 P7*, 2005.
[25] *ARINC 429: Mark 33 Digital Information Transfer Systems (DITS)*, 1996.
[26] S. Samolej, A. Tomczyk, J. Pieniążek, G. Kopecki, T.Rogalski, and T. Rolka, "VxWorks 653 based Pitch Control System Prototype," in *Development Methods and Applications of Real-Time Systems*, L. Trybus and S. Samolej, Eds. WKL, 2010, ch. Chapter 36, pp. 411–420, (in Polish).
[27] T. Rogalski, S. Samolej, and A. Tomczyk, "ARINC 653 Based Time-Critical Application for European SCARLETT Project," Aug 2011, accepted for presentation at the AIAA Guidance, Navigation, and Control Conference, 8-11 Aug 2011 Portland, Oregon, USA.
[28] S. Samolej, A. Tomczyk, and T. Rogalski, "Fault Detection in a ARINC 653 and ARINC 644 Pitch Control Prototype System," Sep 2011, accepted for publication in: Development, Analysis and Implementation of Real-Time Systems, L. Trybus and S. Samolej eds., WKL, 2011, (in Polish).
[29] *VxWorks 653 Configuration and Build Guide 2.2*, Wind River, 2007.
[30] *VxWorks 653 Configuration and Build Reference, 2.2*, Wind River, 2007.
[31] *VxWorks 653 Progmer's Guide 2.2*, Wind River, 2007.
[32] *PikeOS Fundamentals*, Sysgo AG, 2009.
[33] *PikeOS Tutorials*, Sysgo AG, 2009.
[34] *PikeOS Personality Manual: APEX*, Sysgo AG, 2009.