Aerospace Technology Congress, 8-9 October 2019, Stockholm, Sweden
Swedish Society of Aeronautics and Astronautics (FTF)

Considerations on Domain-Specific Architectures Applicability in Future Avionics Systems

Denis S. Loubach[1], Johnny Cardoso Marques[2], and Adilson Marques da Cunha[2]

[1]Department of Computer Systems
[2]Department of Software and Information Systems
Aeronautics Institute of Technology – ITA, São José dos Campos, São Paulo/Brazil - 12228-900
E-mail: dloubach@ita.br; johnny@ita.br; cunha@ita.br

Abstract

The community related to the avionics systems domain tends to be conservative mainly due to the safety-critical requirements this kind of system has to comply with. With respect to the presented context, this paper addresses and discusses the next challenges of future avionics systems always considering the safety-critical aspects and the possibility of formal models of computation application together with domain-specific architectures. Runtime partial reconfiguration, for example, is one promising way to mitigate the increasing failure rate and thereby fulfill safety requirements. N-modular redundancy implementation with different hardware architectures is also analyzed considering reconfigurable computing.

Keywords: domain-specific architectures, avionics systems, safety-critical systems, runtime reconfiguration, reconfigurable computing, formal models of computation.

1 Introduction

The industry community related to the avionics systems domain tends to be conservative mainly due to the safety-critical requirements this kind of system has to comply with. In that case, the majority of development follows the DO-178 [1] for the software, and the DO-254 [2] for the hardware guidance.

On the other hand, high-volume commercial electronics usually dictates some trends such as environmental concerns like the restriction of hazardous substances and packing more and more functionality with enhanced performance into smaller packages also taking into account lower power-consumption. Along the time, this led to chip geometries in the order of nanometers.

Nowadays, computer architecture improvement to deliver performance enhancements is getting harder to achieve, compared to the renascence period from about 50 years ago. The more relevant facts are the end of Dennard scaling and the slowdown of Moore's Law. Chien [3] points out the end of Dennard's scaling, shift to multicore, and slowing in the clock rate growth as *disruption from bellow*. That author also mentions the considerable architectures customization in mobile and embedded devices. Those facts have impact on the avionics systems domain as the system complexity increases exponentially from one generation to the next.

In view of this, one current trend in high-performance computer architecture is the use of *domain-specific architectures* (DSA) instead of the general-purpose ones [4]. DSA is able to provide performance and power benefits to face general-purpose architectures that are not having significant performance improvements in the last years. Potential research areas include high-quality implementations of *open source architectures* and *reconfigurable computing* which are nowadays supported by system-on-chip (SoC) composed by a hard processor and FPGA with runtime full and partial reconfiguration enabled [5].

With respect to the presented context, this paper addresses and discusses the next challenges of future avionics systems always considering the safety-critical aspects and the possibility of formal models of computation application together with DSA. Runtime partial reconfiguration, for example, is one promising way to mitigate the increasing failure rate and thereby fulfill safety requirements. N-modular redundancy implementation with different hardware architectures is also analyzed considering reconfigurable computing.

2 Software & Hardware Working Together

Notice that software and hardware always needed to work together, especially with respect to embedded systems.

One of the basic concepts of embedded systems is hardware and software tightly coupled to address an specific function. However, the modern concept may be slightly different in the sense embedded systems are becoming more "general-purpose" concerning its hardware architecture and computing power.

In this scenario, the following sections introduce the four standards used to develop a safety-critical class of real-time

DOI
10.3384/ecp19162018

embedded systems, *i.e.* avionics. Next, it is presented a general overview of domain-specific architectures, reconfigurable computing, and models of computation.

2.1 System, Software & Hardware Avionics Standards

Typically, safety-critical systems are developed in regulated environments by norms and standards. Examples are found in domains such as: aviation, automotive, medical, railway, space, and nuclear [6].

In avionics systems development, there are four standards typically required to be used as part of the aircraft certification process: SAE ARP-4754A [7], RTCA DO-178C [1], RTCA DO-254 [2], and RTCA DO-297 [8]. The relationship among these standards is presented in Fig. 1.

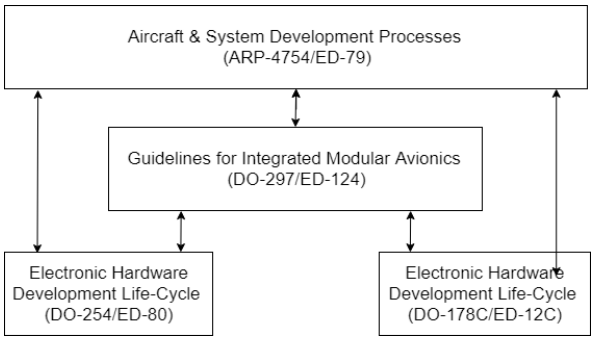


Figure 1: Relationship among certification standards

All those standards use *safety* classification. The system failures are classified within five categories: catastrophic, hazardous, major, minor, and no safety impact. The products (systems, software, or hardware) which malfunction, cause or contribute to a system failure occurrence have an attributed design assurance level (DAL) according to the most critical system failure condition.

2.1.1 SAE ARP-4754A

According to Xiaoxuna *et al.* [9], the SAE ARP-4754A [7] has been prepared primarily for electronic systems which, by their nature, may be complex and are readily adaptable to high levels of integration. However, the guidance is also applicable to engine systems and related equipments. It provides updated and expanded guidelines for processes used to develop civil aircrafts and systems that implement aircraft functions.

The SAE ARP-4754A has 47 objectives presented in Table A-1, published in Appendix A of the standard. Those objectives are organized in 8 processes: Planning; Aircraft and system development process and requirements capture; Safety assessment; Requirements validation; Implementation verification; Configuration management; Assurance; and Certification authority coordination. These processes are organized in process model as presented in Fig. 2.

2.1.2 RTCA DO-178C

The first version of RTCA DO-178 was released in 1982. The European Organization for Civil Aviation Equipment

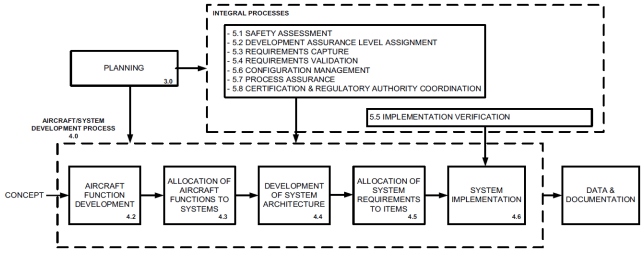


Figure 2: General processes available at SAE ARP-4754A

(EUROCAE) has an Equivalent Document ED-12. In 2011, the current RTCA DO-178C [1] was released.

The RTCA DO-178C has five software levels broken down into objectives to be satisfied. The satisfaction of applicable objectives enables the software approval as part of the aircraft certification process. Among the five existing software levels (A, B, C, D, and E), level A is the most rigorous and requires compliance with all objectives of the standard. The level E refers to software products which malfunction does not result in loss of safety margins.

As presented in Table 1, the classification of the failure condition is associated with the defined software levels. For each software level, a set of objectives are required for compliance demonstration.

Table 1: RTCA DO-178C software levels

Failure Condition	Software Level	Objectives
Catastrophic	A	71
Hazardous	B	69
Major	C	62
Minor	D	24
No Safety Impact	E	None

The 71 RTCA DO-178C objectives are presented in 10 tables, published in Annex A of the standard. The tables identify software process objectives with the following characteristics: Planning (Table A-1); Development (Table A-2); Verification of the high and low-level requirements and software architecture (Tables A-3, A-4, and A-5); Verification of source and executable codes (Tables A-5 and A-6); Testing and analysis (Table A-7); Configuration control (Table A-8); Quality assurance (Table A-9); and Certification (Table A-10). These processes are organized and presented in Fig. 3.

As part of the effort of the DO-178C release, other supplementary standards were also developed, including special recommendations, regarding tool qualification (RTCA DO-330 [11]), *model-based development and verification* (RTCA DO-331 [12]), object-oriented technology (RTCA DO-332 [13]), and formal methods (RTCA DO-333 [14]).

2.1.3 RTCA DO-254

According to Kounish *et al.* [15], the RTCA DO-254 [2] is the standard which is used for the airborne and safety-critical application providing a proper guidance to assure the design

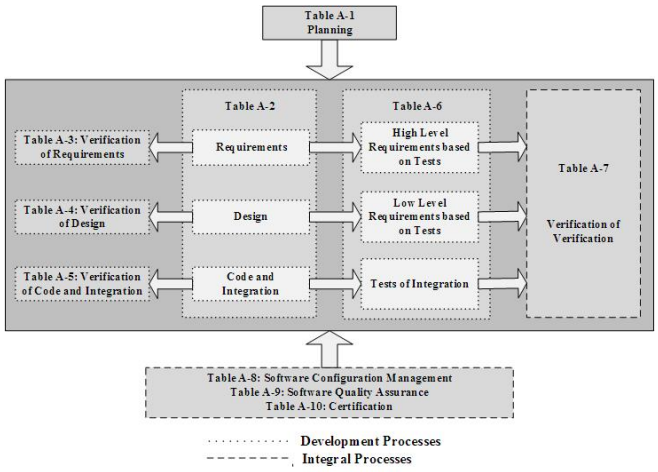


Figure 3: RTCA DO-178C processes organization [10]

of airborne electronic hardware. It has many similarities with RTCA DO-178C. As an example, the RTCA DO-254 has also five design assurance levels broken down into objectives to be satisfied.

The RTCA DO-254 objectives are presented in Table A-1 and published in Appendix A of the standard. It has 6 processes: Planning; Requirements; Conceptual design; Detailed design; Implementation; Validation; Verification; Configuration management; Assurance; and Certification. These processes are organized and presented in Fig. 4.

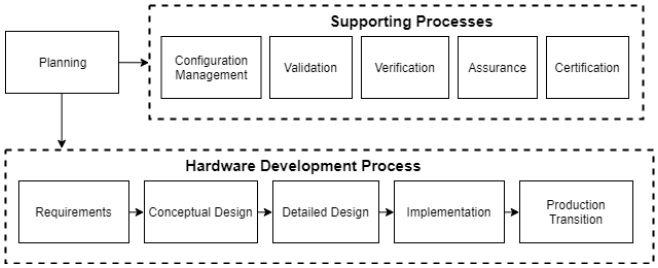


Figure 4: RTCA DO-254 processes organization [2]

2.1.4 RTCA DO-297

According to RTCA DO-297 [8] glossary, the integrated modular avionics (IMA) is

> "a shared set of flexible, reusable, and interoperable hardware and software resources that, when integrated, form a platform that provides services designed and verified to a defined set of safety and performance requirements, to host applications performing aircraft functions".

An IMA system architecture is composed of one or more platforms and includes interfaces to other aircraft systems and users.

The IMA typical approach consists of the following levels of acceptance: component; module; application; platform; IMA system; and IMA aircraft. These levels of acceptance are organized as illustrated in Fig. 5.

A *component* is a self-contained hardware part, software part, database, or combination thereof that is configuration controlled. A component does not provide an aircraft function by itself.

A *module* may be software, hardware, or a combination of hardware and software, which provides resources to hosted applications. Modules may be distributed across the aircraft or may be co-located.

An *application* is a collection of software and/or hardware modules with a defined set of interfaces that, when integrated with a platform, performs a function.

At component, module, and application levels, the developments of such parts should follow the RTCA DO-178C and/or the RTCA DO-254.

A *platform* is a group of modules that establishes a computing environment, support services, and platform-related capabilities, such as health monitoring and fault management.

An *IMA system-level* consists of platform(s) and a defined set of hosted applications.

The IMA aircraft-level should demonstrate that each aircraft function and hosted application functions as intended, supports the aircraft safety objectives, and complies with the applicable regulations. However, during the installation activities, the interactions between hosted applications relative to the provided aircraft functions should be verified and validated during aircraft ground and flight testing.

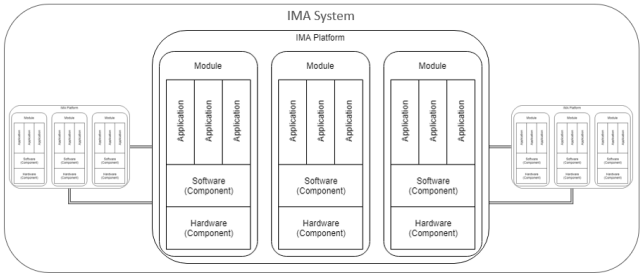


Figure 5: Levels of IMA approval

2.1.5 Wrap-Up

Together, these four standards are closely pursued in the development of avionics systems. From the system perspective, passing through modules, down to the software and the hardware.

The conservatism from the avionics system community is mainly due to the levels of impact a failure may cause, for example, level A in DO-178. Therefore, new trends are almost always seen as "dangerous" to be included or considered in avionics.

However, if those trends are backed-up by sound basis, they can be step by step introduced in future generation systems. We argue that it is the case regarding domain-specific architectures supported by reconfigurable computing and models

of computation.

2.2 Domain-Specific Architectures

During the semiconductor boom, supported by the Moore's Law, general-purpose code was accelerated in general-purpose cores, including techniques such as cache hierarchy; 512-bit single instruction, multiple data (SIMD) floating-point units; various pipeline stages; dynamic branch prediction; out-of-order execution; speculative execution; multithreading; and multiprocessing. Nowadays, it is needed a paradigm shift in computer architecture to achieve a new level of efficiency. This shift concerns moving from general-purpose hardware to domain-specific architectures [4].

Gupta [16] mentions three different architectures, as shown in Table 2.

Table 2: Architecture paradigm and domain [16]

Architecture	Domain
Scalar (*e.g.* CPU)	Complex algorithms with diverse decision tree. Limited in performance scaling.
Vector (*e.g.* GPU)	Efficient in reduced set of parallelizable functions. Suffers latency and penalties related to memory hierarchy.
Programmable logic (*e.g.* FPGA)	Customized to computer a particular function. May take hours to compile and synthetize.

Central Processing Unit (CPU); Graphics Processing Unit (GPU); Field Programmable Gate Array (FPGA).

Different from general-purpose architectures, the hardware can be specific and optimized for a particular domain in DSA. Moreover, this can be done in runtime, as discussed in the next section.

In this sense, computers will be more and more *heterogeneous*.

2.3 Reconfigurable Computing and Formal Models of Computation

Reconfigurable computing includes computation research towards the use of reconfigurable devices. In this case, for a specific application requirement, *i.e. domain-specific*, and at given time frame, the reconfigurable devices' spatial structure is changed to comply with a given objective [17, 18].

Reconfigurability is the "art of engineering degrees of freedom into embedded systems", as highlighted by [19]. Notice, that this concept is different from the *adaptiveness*. The latter stands for a possible behavioral ability a system can have that aids to decide how the system should be reconfigured in a optimized fashion. Adaptive computing is stated as one of the reconfigurable computing research areas.

A reconfiguration design is presented in [5]. The author introduces a hardware and software composition which shows the application of both *partial* and *full* runtime hardware reconfiguration based on a heterogeneous computer. In other words, a SoC with *scalar* and *programmable logic* elements integrated in a single package. Then, that reconfiguration design, composed by hardware and software, can be classified as a domain-specific architecture.

Architectures like that can be applied to the avionics systems domains, as far as they are able to provide the safety inherent from this domain.

In this sense we can use formal *models of computation* (MoC) together with runtime reconfiguration to provide flexibility and performance along with safety to future avionics systems.

Different models types exist to address different purposes. *Functional modeling* is used to address the functional behavior of a system. On the other hand, design and synthesis refine into implementation details. A *model* is defined as an abstraction of an entity, *e.g.* a physical system or even another model, according to [20]. That author also states that abstraction is a method for choosing which aspects will be taken into account when modeling a system.

A MoC is an abstraction of a physical computing device, and so, different MoCs serve to different objectives.

MoCs are based on three main points: *processes*, *events*, and *signals*. The following definitions are derived from [20, 21].

Definition 1. *Event. Elementary information unit exchanged between processes.*

Definition 2. *Signal. Processes communicate to each other by writing to and reading from signals, which are a sequence of events. Signals preserve the order that events are entered. Each event has a tag and a value. Tags can be used to model physical time, events order, and other key properties of a MoC.*

Definition 3. *Process. Receives and send events. The process activity is comprised of evaluation cycles, i.e. an application of a function which maps inputs to outputs. Then, in each evaluation cycle the process receives inputs, computes, and sends outputs.*

Definition 4. *Process Constructor. Parameterizable templates for instantiating processes, i.e. a higher-order function.*

Processes constructors are used to create processes. New processes can be created to form a hierarchical concurrent process network.

Definition 5. *Model of Computation. Set of processes and process networks implemented by a well-defined set of processes constructors.*

The modeling task can be supported by a tool or framework. Horita *et al.* [22] introduced a framework comparison method with a number of desirable characteristics the tools should have to aid in the formal modeling and simulation of systems. ForSyDe was one of the frameworks analyzed.

Formal System Design (ForSyDe) [23] is a transformational design methodology based on the *functional programming paradigm*. It targets heterogeneous embedded systems [20].

A system is modeled as a hierarchical concurrent process network in ForSyDe. Processes communicate with each other by signals.

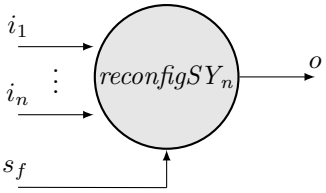
In ForSyDe, we model a *signal* as a list of *events*, where the event *tag* is implicitly given by the event position in the list. The semantics of a tag is defined by the currently in-use MoC, *i.e.*, an identical tag of two events in different signals does not imply these events happened at the same time. All events in a signal must have values of the same type [21].

The synchronous (SY) MoC splits the time domain into slots. Everything inside a slot occurs at the same time. The evaluation cycle of processes lasts exactly one time slot in synchronous MoC [20], according to the *perfect synchrony hypothesis*, *i.e.* neither computation nor communication takes time.

Synchronous processes consume and produce exactly one event on each input and output in each evaluation cycle. This implies the *total order* of all events in any signal in the SY MoC. Events with the same tag appear at the same time instant.

Taking these fundamental basis explored so far, the runtime reconfiguration can be modeled as higher-order function [24], as follows.

The adaptive process *reconfigSY* definition is illustrated in Fig. 6.



$$ap_f = reconfigSY_n$$

where:

$$o = ap_f(s_f, (i_1, \ldots, i_n))$$
$$o[k] = s_f[k](i_1[k], \ldots, i_n[k])$$

Figure 6: Process constructor for reconfigSY [21]

reconfigSY synchronous reconfigurable process can be modeled by using the *mapSY* ForSyDe process constructor.

$$reconfigSY = mapSY(\$) \quad (1)$$

Let's assume an N-modular redundancy [25]. Typically, this is applied to safety-critical system to provide dependability. Different modules may be implemented differently even using distinct hardware architectures, as long as their final output is the same. This is totally possible to be implemented using runtime reconfiguration, as shown in [26].

Concerning the aeronautics trend towards the second generation of integrated modular avionics adoption, DSA is certainly a very good candidate to be used in future avionics systems.

Examples of applications combining DSA and runtime reconfiguration are reduced instruction set computers (RISC) designed for a specific computation, and software-defined radio (SDR). In the first case, it is possible to have many different RISC bitstreams in the memory and load or unload them in runtime using partial reconfiguration, according to the current need. Regarding SDR, it is possible to have different implementations and use them depending on the present context to address challenges such as size, weight, and power (SWaP).

Some benefits of this scheme are hardware reuse resulting in a reduced circuit area thus contributing to minimizing single event upset (SEU), and the full control of the trade-off involving power consumption and performance.

3 Next Challenges and Trends

Hennessy and Patterson [4] mention that a current DSA challenge is *how to port software*, generally written in programming languages such as C/C++ and Ada, to different architectures. This challenge is similar to the one faced back in the 80's: how to port code written in Assembly to different microcontrollers without the need to rewrite almost all of it. The answer for the latter was high level languages and hardware room. *i.e.* memory, to fit more and more code.

Another challenge to the integrated modular avionics is the *runtime hardware reconfiguration* application. The question is how to ensure the reconfiguration is safely performed in a deterministic way.

Confidently, DSA together with formal models of computation figure as a promising trend for the next years.

4 Summary

This paper presented some considerations, supported in the literature, on domain-specific architectures applicability to future avionics systems.

A research path is open concerning more and more application of formal models in real-time embedded safety-critical systems such as avionics. Besides, reconfigurable computing figure as one promising underlying element to achieve power-consumption and performance efficiency in domain-specific architectures.

5 Acknowledgments

This research work is supported by the Research Development Foundation (*Fundacao de Desenvolvimento da Pesquisa)* - FUNDEP/MCTIC/MDIC, the *Ecossistema Negocios Digitais Ltda*, the *Casimiro Montenegro Filho Foundation* (FCMF), and the Brazilian Aeronautics Institute of Technology - ITA.

References

[1] Radio Technical Commission for Aeronautics - RTCA. *DO-178C - Software Considerations in Airborne Systems and Equipment Certification*, 2012.

[2] Radio Technical Commission for Aeronautics - RTCA. *DO-254 - Design Assurance Guidance for Airborne Electronic Hardware*, 2000.

[3] Andrew A. Chien. Computer architecture: Disruption from above. *Commun. ACM*, 61(9):5–5, August 2018.

[4] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 6th edition, 2017.

[5] Denis S. Loubach. A runtime reconfiguration design targeting avionics systems. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, pages 1–8, Sacramento, USA, September 2016. IEEE.

[6] Johnny Marques and Adilson Cunha. Verification scenarios of onboard databases under the rtca do-178c and the rtca do-200b. *36th IEEE/AIAA Digital Avionics Systems Conference*, 2017.

[7] Society of Automotive Engineers-SAE. *ARP-4754A Guidelines for Development of Civil Aircraft and Systems*, 2011.

[8] Radio Technical Commission for Aeronautics - RTCA. *DO-297 Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations*, 2011.

[9] L.. Xiaoxun, Z. Yuanzhenb, F. Yichenb, and S. Duoa. A comparison of sae arp 4754a and arp 4754. *Procedia Engineering*, 17:400–416, 2011.

[10] Leanna Rierson. *Developing Safety-Critical Software: A Practical Guide for Aviation Software and DO-178C Compliance*. CRC Press, 2013.

[11] Radio Technical Commission for Aeronautics - RTCA. *DO-330 Software Tool Qualification Considerations*, 2011.

[12] Radio Technical Commission for Aeronautics - RTCA. *DO-331 Model Based Development and Verification Supplement to DO-178C and DO-278A*. RTCA, 2011.

[13] Radio Technical Commission for Aeronautics - RTCA. *DO-332 Object-oriented and Related Technologies Supplement to DO-178C and DO-278A*. RTCA, 2011.

[14] Radio Technical Commission for Aeronautics - RTCA. *DO-333 Formal Methods Supplement to DO-178C and DO-278A*. RTCA, 2011.

[15] R. Koushik, M. Anushree, B. J. Sowmya, and N. Geethanjali. Design of spi protocol with do-254 compliance for low power applications. *2017 International Conference on Recent Advances in Electronics and Communication Technology (ICRAECT)*, 2017.

[16] Amit Gupta. Advances in adaptable computing. In *Proceedings of the 2019 International Symposium on Physical Design*, ISPD '19, pages 37–38, New York, NY, USA, 2019. ACM.

[17] Christophe Bobda. *Introduction to Reconfigurable Computing: Architectures, Algorithms, and Applications*. Springer, 2007.

[18] Dirk Koch. *Partial Reconfiguration on FPGAs: Architectures, Tools and Applications*. Springer-Verlag New York, 1 edition, 2013.

[19] J. Lyke, C. G. Christodoulou, A. Vera, and A. H. Edwards. Reconfigurable systems: Advanced applications and technologies [scanning the issue]. *Proceedings of the IEEE*, 103(7):1000–1003, July 2015.

[20] Axel Jantsch. *Modeling Embedded Systems and SoC's: Concurrency and Time in Models of Computation*. Morgan Kaufmann, San Francisco, USA, 1st edition, 2003.

[21] Ingo Sander and Axel Jantsch. Modelling Adaptive Systems in ForSyDe. *Electronic Notes in Theoretical Computer Science*, 200(2):39 – 54, 2008. Proceedings of the First Workshop on Verification of Adaptive Systems (VerAS 2007).

[22] Augusto Y. Horita, Ricardo Bonna, and Denis S. Loubach. Analysis and comparison of frameworks supporting formal system development based on models of computation. *Advances in Intelligent Systems and Computing*, 800:161–167, 2019.

[23] I. Sander and A. Jantsch. System modeling and transformational design refinement in ForSyDe [formal system design]. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 23(1):17–32, Jan 2004.

[24] Denis S. Loubach, Eurípedes G. O. Nóbrega, Ingo Sander, Ingemar Söderquist, and Osamu Saotome. Towards runtime adaptivity by using models of computation for real-time embedded systems design. In *9th Aerospace Technology Congress*, Solna, Stockholm, October 2016. FTF - Swedish Society of Aeronautics and Astronautics.

[25] H. . Lo, L. . Ju, and C. . Su. General version of reconfiguration n modular redundancy system. *IEE Proceedings G - Circuits, Devices and Systems*, 137(1):1–4, Feb 1990.

[26] Ricardo Bonna, Denis S. Loubach, Ingo Sander, and Ingemar Söderquist. Triple modular redundancy based on runtime reconfiguration and formal models of computation. In *10th Aerospace Technology Congress*. Swedish Society of Aeronautics and Astronautics (FTF), 2019.