Modelling and Control of Magnetic Fusion Plants Based on ARM CPU and STM32 MCU Platforms

Giuseppe Avon

PhD Supervisor: Prof. Arturo Buscarino

ENI Supervisors: Eng. Eliana De Marchi, Eng. Fabio Zanon

Dipartimento di Ingegneria Elettrica Elettronica ed Informatica

Universita' degli Studi di Catania

Doctor of Philosophy in Ingegneria dei Sistemi, Energetica, Informatica e delle Telecomunicazioni

ACKNOWLEDGEMENTS

This thesis was developed under the ENI – CNR – University of Catania Joint Research Agreement. Parts of this work were developed in collaboration with the RFX Consortium. Contributions to the MARTe2 (Multithreaded Application Real-Time Executor) framework were developed as part of a traineeship as developer at Fusion for Energy European Agency for the ITER (International Thermonuclear Experimental Reactor) development, in the CODAC (Control, Data Access and Communication) group.

I would like to express my deepest appreciation to Prof. Arturo Buscarino and Prof. Luigi Fortuna as this endeavor would not have been possible without their support, together with Eng. Fabio Zanon and Eng. Eliana De Marchi.

Words cannot express my gratitude for André Neto and its extraordinary development team at Fusion for Energy, for their constant support, help and contribution to my growth as researcher, developer and human.

Lastly, I'd like to acknowledge Marina and my whole family for being always on my side during the whole period.

ABSTRACT

This thesis is devoted to the modelling, design and implementation of systems that can be integrated into magnetic fusion plant controllers. These systems, usually referred to as *control and data acquisition systems*, supply the interface to all the instrumentation required for the nuclear fusion plant. At an extremely high level of description, control and data acquisition systems can be divided into two main subsystems: *plasma control* and *integrated control* systems. The plasma control system (PCS) controls the fusion reactor runtime operation, it runs on heavily customised hardware platforms real-time, high frequency, data acquisition and control loops. The integrated control system (ICS), instead, controls the auxiliary plants like electric distributors, water, and cryogenic cooling pumps, running on off-the-shelf hardware (PLC/PAC/PXI).

The presented work result falls under the PCS subsystem categorisation and follows a path starting with the study of the model for the vertical stabilisation of the plasma position, with an insight on the RFX experiment. The study also evaluated the possibility to implement a *circuital analogous* of the plasma model, to enable the rapid development and validation of real-time control strategies using *hardware-in-the-loop* (HIL) and *software-in-the-loop* (SIL) platforms.

The second part of the thesis will present the contribution to the development of MARTe2, a framework deployed in several fusion real-time control systems PCS. The contribution was aimed (1) at the integration of the MARTe2 framework with ICS field peripherals (PROFINET®) and (2) at the porting of the framework on ARM-based platform, both as CPU (Central Processing Unit) and MCU (Micro Controller Unit).

Purpose of the porting is to bring the standardisation, modularity and reusability offered by the MARTe2 framework on devices able to directly work with the field, which would otherwise require ad-hoc firmware solutions and approaches.

The PROFINET® component, as part of the MARTe2 components suite, adds several opportunities related to the interaction of the framework with the ICS field, which will be discussed thoroughly in the designated chapter.

While the ARM porting of the MARTe2 framework will be employed as part of the magnetic sensor diagnostic system, the PROFINET® component will serve the electron cyclotron resonance heating (ECRH) factory acceptance testing (FAT) tools suite. Aside main research

and development activities, several side quests enrich the contribution given to one of the International Thermonuclear Experimental Reactor – ITER software frameworks, contributing to the development of real-time diagnostics of the PCS subsystem.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

Development in the nuclear fusion field is one of the most challenging tasks nowadays. From the research point-of-view, nuclear fusion covers an extremely wide area, ranging in every field, from plasma physics, control engineering, computer science, landing to socio-economic and cultural impact of the availability of an unlimited power source. In this chapter a quick overview on the fusion and on the currently running experiments will be given. Afterwards the context of this work will be examined, to have a better understanding of the topics that will be debated. The last part of the chapter will present a summary of the activities that will be discussed in the operational core of the thesis work. Information on this introductory chapter will be organised as a flow which will naturally lead to the core chapters of the thesis with the necessary knowledge to deal with the presented contents.

1.1 CHAPTER KEY POINTS

- The energy powering the stars, as the Sun, comes from the fusion of hydrogen atoms together to form helium. Gravitational forces on the stars create the ideal conditions from fusion, on Earth, instead, the ideal condition must be recreated by confining the fuel inside a reaction vessel.
- Current readily feasible reaction is the fusion of Deuterium and Tritium isotopes of the Hydrogen. Fusion technologies operate the confinement of the plasma to obtain its *ignition.*
- Several approaches are pursued in control systems to contain the plasma, recreating the temperature, density and confinement conditions needed to obtain the ignition.
- Magnetic confinement inside toroidal-shaped vessels is one of the most common approaches. The ITER project vessel will be the largest operating reactor in the world with this configuration. Inertial confinement is another approach, which relies on a different principle.
- Mathematical models, supported by heavy field experimentations, help understanding, predicting, and formulating strategies to control the plasma behaviour on the fusion reactor.

- Fusion reaction and all the auxiliary components in the plant are controlled by a complex network of systems which run a combination of off-the-shelf and custom hardware and software solutions.
- MARTe2, a software framework used in some systems that belong to the ITER plasma control system (PCS), is used in many fusion experiments around the world. Its layered and modular architecture create a standardised approach to the solution of many common real-time control systems problems.
- Main contribution of the thesis work is related to the MARTe2 software framework and to its component's suite development. A model driven approach, called *circuital analogous* technique is also explored, to model, simulate and control a specific aspect of the plasma behaviour.

1.2 BACKGROUND: NUCLEAR FUSION

The following introductive chapters outline the context where the research and development work are located. Their main purpose is to introduce the reader to the themes, terms, problems, and solutions commonly used in the nuclear fusion field. As many technical solutions are available in the field, only relevant to the discussion are treated, in order to keep this introduction as lean as possible. Moreover, as the work is extremely specific, it also serves to the reader to locate the scopes and circumscribe the operational field of the presented solutions.

Nuclear fusion starts by bringing together two light elements so that they can fuse and form a heavier element. The resulting element will have a slightly lower mass than the sum of the starting two, the mass difference results in the release of energy. The released kinetic energy of the product elements is then trapped to be used for power generation. The most practical fusion reaction for power generation is the Deuterium-Tritium reaction (D + T), due to the lower energy requirements.

To accomplish the fusion, particles must overcome the electric repulsion barrier. Beyond the repulsion barrier, called *Coulomb Barrier*, the particles will be close enough for the *strong nuclear force* to act. Three parameters: (1) temperature, (2) density and (3) confinement time, often referred to as *triple product* represents a useful figure of merit to threshold the *ignition* conditions for the plasma. Once the *ignition* condition is reached, the fusion reaction becomes self-sustaining and external heating sources can be de-activated. Parameters of the *triple product* are strictly interconnected among them.

To obtain the required temperature, plasma in the reactor must be heated. Initial heating is obtained by *ohmic heating*, which is an effect that can be assimilated to the heating process that heats up a wire where current is flowing. *Ohmic heating* is commonly used in the plasma pre-heating, as it does not allow the plasma to reach the required temperature for the fusion. In fact, as plasma temperature arises, the collision frequency and the resistivity both decrease Due to the described effect, *Ohmic heating* allows only the plasma to reach temperatures up to a few keV, of the around 15 keV needed. To reach the needed temperatures, two main methods are available: the first technique is based on injecting a beam of high-energy neutral particles to the plasma (Neutral Beam Injection – NBI), the second one is based on the radiofrequency irradiation (RF). Radiofrequency irradiation is obtained injecting waves with frequencies resonating with the natural plasma frequencies into the plasma vessel, some techniques used are the electron cyclotron (ECRH) and ion cyclotron resonance heating (ICRH).

In the heated plasma, the high kinetic energy would render collisions extremely unlikely. To bring the plasma to the ideal density and have it circulating in the reaction vessel, a system based on vessel-positioned electromagnets, driven by a complex control system, controls plasma position and thus the density. Also, density must be carefully controlled: if it goes over the optimal range, collisions happen between nuclei and electrons, creating large amount of radiation, called *bremsstrahlung*, which impairs fusion taking away energy from the plasma.

The third parameter, confinement time (often denoted with the Greek letter τ - tau), denotes the time duration during which newly created Helium atoms are confined within the plasma, allowing them to transfer their energy to the unburnt fuel and keep the ignition condition.

The ignited plasma must be *confined* into the reactor vessel, also keeping it away from the surrounding walls. The two main approaches to achieve the confinement are (1) magnetic and (2) inertial.

Magnetic confinement relies on the electromagnetic properties of the plasma, controlling it through interaction with magnetic fields. The objective is to restrict the motion of the plasma particles *across* the magnetic field lines of force, conversely allowing it to move freely *along* them. The most efficient vessel configuration to achieve the magnetic confinement is the tokamak (**то**роидальная **ка**мера с **ма**гнитными **к**атушками – **to**roidal **cha**mber with **ma**gnetic **c**oils) concept.

Inertial confinement, instead, bases its operating principle on the heating and on the compression to extremely high densities of a solid fuel pellet. The compression is obtained by

heating a spherical hollow target uniformly to trigger an implosion of the fuel. Energy is provided by pulsed lasers or ion beams from an accelerator.

In a tokamak, the plasma is shaped in a torus, by using a *toroidal* and a *poloidal* magnetic field. The toroidal field is produced by ring coils around the vessel, the poloidal field is instead produced by passing a current in the plasma itself. Plasma, ideally, can be seen as a secondary winding of a transformer, where the primary winding is an inductor placed in the hollow centre of the tokamak torus.

The resulting helical shaped field has a characteristic pitch which is often referred to as *safety factor* or *q*, where safety refers to the resulting plasma stability. An additional magnetic field, called *vertical* is needed to avoid the plasma outwards expansion. The vertical field is obtained with torus-concentric positioned coils.

Amongst technical advances in tokamaks, in an aim to obtain a higher efficiency in plasma confinement, it has been found that the optimal configuration is not a circular but a D-shaped torus. The D shape is regulated by two parameters called *triangularity* and *elongation* and are characteristics of the tokamak vessel.

Currently, tokamaks are the most advanced fusion reactors based on magnetic confinement. The tokamak configuration in fact, as most promising candidate for the first generation of reactors, was chosen for the ITER experiment. [1] [2] [3] [4]

1.3 TOWARDS THE CONTROL

The confinement of the plasma is an extremely complex control task, as it is unstable. Arising instabilities are linear, non-linear, with different wavelengths and frequencies. The plasma itself shows self-organizing behaviours [5]. When the plasma collides with the walls, it is terminated: it is called *plasma disruption*. Disruption is the result of a large-scale *magnetohydrodynamic* (MHD) instability. Small-scale instabilities also exist due to gradient in plasma parameters and cause anomalous transport: their collective effect can also drive fluctuations which result in large-scale instabilities. [6] [7]

Plasma control is essential for the practical power generation and for the reactor machine integrity. In fact, discharges are also detrimental to the life of the vessel wall materials, other than the immediate termination of the fusion reaction. Extremely complex models exist to model the plasma behaviour, often called *codes*. Modelling activity is essential to the

development of the control laws to keep the fusion steady state. They are used in offline computations, to derive control laws and in online computations, in the plasma control system, to estimate plasma parameters that cannot be directly measured. There are many models, each one for a particular aspect of the fusion reaction, their development is supported by the ongoing measurements in all the fusion experiments around the world and their employment drives further discoveries and refinements to them, in a positive feedback scheme.

Plasma modelling and control is based on the possibility to measure the plasma parameters. A branch of the research, called *plasma diagnostics*, oversees finding ways and supporting systems to measure, directly or indirectly, the plasma parameters. Plasma diagnostics is a complex field with roots in an extremely high number of fields. Purpose of the plasma diagnostic systems is to measure operating parameters for the control, safety and for the physics, like temperatures, relative positions, magnetic fields, currents. Diagnostics must also be an extremely flexible and expandable system, which must evolve faster than the other components in the system. [2]

1.4 THE PLANT CONTROL SYSTEM

All the different plant systems that constitute a fusion device are organised in a hierarchical tree going from the field sensors and actuators up to the central supervisory, monitoring and data handling facilities. ITER facility designates the entire system as *Plant Instrumentation & Control (Plant I&C)*. In these systems there is a high degree of integration between standardized and custom (non-standardized) controllers. A standardized system means an off-the-shelf device which can be a programmable logic controller (PLC, standalone) or a reconfigurable I/O (RIO, integrated in rack mounted PCs). A non-standardized system, instead is based on a custom board which runs a custom software, specifically designed for the controlled environment. The hierarchical tree components are networked by using several networks (physical or logical): relying on optical fibre or on copper media, they guarantee the data exchange keeping network traffic segregated by category (Data Archiving, Time Communication, Audio/Video, Synchronous Communication, Safety and Interlocks).

Another common categorization amongst plant controller systems is the *speed*. *Fast controllers*, usually employed in diagnostics, plasma control and closed-loop plant control systems, run on high-performance network with hard real-time constraints at high sampling rates (up to tens of kHz). Hardware platforms for fast controllers is usually a dedicated system with RIOs or custom

hardware boards. *Slow controllers* instead are employed in auxiliary systems control, are based on PLCs, and operate at low sampling rates (up to hundreds of Hz). [8]

1.5 QUICK OVERVIEW OF SOFTWARE IN REAL-TIME PLASMA CONTROL

Drilling down through the plasma control and diagnostics systems, where *fast controllers* run high sampling rates control loops, software is key to succeed in the operation. This is especially true since significant portions of the PCS run on custom hardware solutions. One common software framework for the development of real-time control applications, in the fusion field is MARTe2.

MARTe2 is a multiplatform software framework, deployed in many nuclear fusion real-time control systems (e.g., ITER or JET tokamaks). The framework offers a comprehensive suite of tools to develop an application, both for developers and for users. Developers can benefit from the MARTe2 architecture to build drivers to interact with the field or processing algorithms by defining the "construction blocks" of the control applications. Users, instead, develop their control applications by leveraging already existing "blocks" and facilities using a descriptive configuration language which helps connecting block and defining the control algorithm. [9]

1.6 QUICK OVERVIEW ON THE FUSION REACTORS OF CONCERN

For the mentioned experiments in the presented thesis, a quick reference on their main specifications is given.

1.6.1 ITER

ITER (International Thermonuclear Experimental Reactor) also metaphorically recalling the Latin word *iter* meaning "the way" or "the path", is an undergoing research and development project aimed at obtaining the nuclear fusion. ITER, currently under construction at the moment of writing, will be the world's largest magnetic confinement tokamak-based experimental nuclear reactor. It is in the southern France, at Saint-Paul-lés-Durance, near the Cadarache technological research and development centre for energy. ITER is considered the most complicated engineering project in human history, seeing thousands of engineers and scientists involved in a 35-year collaboration. ITER has a series of objectives to reach during its roadmap:

- Create a milestone for the integration of nuclear fusion technologies and allow and assess studies on plasma for the current and future generations of nuclear fusion machines.
- Obtain ignition, by the D-T fusion reaction and sustain it for a prolonged period, as a starting point for production power plants, also demonstrating the safety of the technology.
- Demonstrate the tritium breeding, by producing tritium in the vessel, as current tritium availability will not suffice the estimated need of the future fusion plants.
- Produce 500 MW of power from the fusion and operate on a ten-fold return ratio (Q>10), by inputting 50 MW of heating power. Produced energy however will not be converted and exported to the grid. [10] [11]

Table 1 ITER main relevant parameters and dimensions

Description	**Value**
Total Fusion Power	1.5 GW
Neutron Wall Loading	1 MW/m2
Plasma Radiuses (Major/Minor)	8.1/2.8 m
Plasma Current	21 MA
Toroidal Field @ 8.1 m Radius / Toroidal Field Coil	5.7 T / 12.5 T
Auxiliary Heating Power	100 MW

1.6.2 REVERSED FIELD EXPERIMENT - RFX

Situated in Padua, in the RFX consortium facility, it was built in 1991, saw first plasma in 1992 and underwent a first package of upgrades (RFX-mod) in 2004. It was operative until 2016 and currently undergoes another major upgrade (RFX-mod2). Its configuration is slightly different from the tokamak, and is called *reversed field pinch*, where the largest part of the controlling magnetic field is sourced directly from the plasma itself, instead of being supplied with external coil sources. RFX is the world's largest reversed field pinch machine and has one of the most advanced plasma control systems, based on 192 coils distributed on the toroidal chamber. The coils can be independently or tandem controlled. Its main contribution was in the field of the active plasma control techniques, based on the saddle coils on the reactor vessel, running real-time control loops at kHz sampling rate. Results from the RFX experiments contributed to the development of several other nuclear fusion experiments, including ITER itself. Due to its ability to induce 2 MA of current in the plasma, by applying 20 V of voltage potential to the

plasma ring, it can produce 40 MW of heating power, causing the typical tokamak heating system to be useless. [12] [13]

Table 2 RFX parameters and dimensions

Description	**Value**
Torus radiuses (inner/outer)	0.459 / 2 m
Maximum plasma current	2 MA
Toroidal field	0.7 T

1.6.3 DIVERTOR TOKAMAK TEST - DTT

DTT project is a tokamak-based experiment, specifically designed to evaluate different divertor performances. A divertor is a device that is used to extract heat and by-products ashes from the in-vessel plasma. The DTT experiment, started in 2018, will help solving the divertor design challenge for the DEMO project, the ITER successor. [14]

Table 3 DTT parameters and dimensions

Description	**Value**
Torus radiuses (inner/outer)	0.65 / 2.10 m
Maximum plasma current	5.5 MA
Auxiliary Heating Power	45 MW
Toroidal field	6 T

1.6.4 FUSION FOR ENERGY

European Joint Undertaking for ITER and the Development of Fusion Energy, shortly named Fusion for Energy and abbreviated in F4E is the EU body responsible for the contribution to the ITER project. It is based in Barcelona (Spain) with offices also in Cadarache (France, ITER plant) and Garching (Germany). It was established in 2007 for a period of 35 years. Its mission is to contribute to the development of demonstration fusion reactors by offering technical knowledge and expertise. [15] [16] [17]

1.6.5 ENI – CNR JOINT RESEARCH AGREEMENT

The Italian multinational oil and gas company signed a Joint Research Agreement for the development of strategic assets, as the magnetic confinement fusion, for the energy transition. The ENI-CNR partnership has led to innovative solutions in various fields and is actively

contributing to the development of magnetic fusion, also through the Commonwealth Fusion Systems (CFS).

1.6.6 COMMONWEALTH FUSION SYSTEMS (CFS)

Commonwealth Fusion Systems is a Massachusetts Institute of Technology (MIT) spin-out actively working in the development of fusion for industrial applications. CFS aims at creating a synergy between the scientific and industry world and has a collaboration with the Plasma Science and Fusion Center of the MIT.

1.7 CONTRIBUTION AND ORGANISATION OF THE THESIS WORK

The thesis contribution can be divided into two main achieved goals. One is related to the modelling and simulation of a specific aspect of the plasma, recurring to a hybrid theoretical-practical approach based on electronic components, using RFX experiment models.

The other part, developed under a traineeship agreement with Fusion for Energy in the CODAC Group, is based on contributions to the development of the MARTe2 framework: in particular, the porting of the framework on a custom hardware board, which is the main digital component of the magnetic diagnostics system of the ITER tokamak and a driver to interface MARTE2 with the PROFINET field bus, to complete the ITER ECRH heating system factory acceptance test tools suite.

The core of this work is organised into three chapters:

- Chapter 2 presents the study of plasma behaviours that are due to the nature of the containing vessel: the Resistive Wall Modes (RWM). After an explanation on the RWM, one of the mathematical models describing this behaviour is presented, in order to try to reduce its complexity. The reduction is aimed at obtaining a *circuital analogous* implementation. A circuital analogous implementation is a physical circuit built upon a model set of differential equations. Although a circuital analogous implementation has several advantages, its implementation becomes unfeasible as the model grows in size. An attempt to reduce the model size is presented, preserving the unstable plasma states which are subject of the stabilisation control.
- Chapter 3 presents the development of an interface driver between the real-time control framework MARTe2 and the PROFINET® field network, called *ProfinetDataSource*. The chapter goes through all the implementation steps details of both the protocol layer

and the framework interfacing, ending with a presentation of the main usage scenario of the driver and a complete series of implications that originated from the implementation strategy. The chapter ends with a presentation of the achievements, as the *ProfinetDataSource* became part of the ITER Electron Cyclotron Resonance Heating, Factory Acceptance Test Tools (ECRH FAT-Tools) software suite.

- Chapter 4 presents the work of porting MARTe2 real-time control application framework on ARM platform, both for bare metal (without operating system) and FreeRTOS (real-time operating system). The work also goes through a complex approach, called Asymmetrical Multi-Processing (AMP), where all the cores in a multicore equipped processor are used independently, as totally different units. The chapter ends with a presentation of the achievements, as the porting became part of the ITER Magnetics Diagnostics project, where ARM Systems-On-Module (SOMs) acquire, process and stream data coming from the magnetics sensor on the vessel to compute plasma parameters.

CHAPTER 2. RESISTIVE WALL MODES MODELLING AND CONTROL

Resistive wall modes (RWM) are global magnetohydrodynamic (MHD) instabilities that are common to many toroidal confinement devices like tokamaks, spherical tokamaks and reverse field pinches (RFP). In a tokamak configuration they are one of the most severe limits in achieving the advanced tokamak regime. In the reverse field pinch configuration, RWMs are found as current driven instabilities. During the last years, significant efforts have been spent to develop efficient control strategies [18].

2.1 CHAPTER KEY POINTS

- RWM instabilities are common in tokamaks experiments, their mitigation and control are key to the achievement of plasma regimes.
- Several models (*codes*) exist to describe the RWM instabilities, helping the development of control strategies, also to be applied in real-time.
- Codes rely on large dynamic models, with great numbers of inputs, outputs and states.
- An approach, called *circuital analogous*, allows the implementation of the equations of a dynamic system into a physical circuit.
- As circuits rely on components which are not perfect, nor ideal, a circuital analogous offers some advantages over pure mathematical approach. The control strategy in this scenario is nearer to the real-world implementation.
- Circuital analogous suffer from a heavy drawback, due to the growing size and complexity of implementation as the number of I/O's and states of the dynamic model grows.
- Integrating the RWM model into a circuital analogous requires an extreme model simplification, which shall not disregard the essence of these kind of instabilities while keeping a feasible approach.

2.2 RESISTIVE WALL MODES

In many "*reversed field pinch"* (RFP) experiments the plasma is surrounded by a highly conducting, close-fitting wall that stabilizes the plasma to all ideal, free-boundary instabilities.

A perfectly conduction wall forces the radial component of the magnetic field at the boundary to vanish. For an ideal plasma, in which the plasma is frozen to the field, the radial velocity is also zero at the boundary. Hence, the boundary is fixed. Instabilities that require the boundary to move are stable with a conducting boundary. As a result, existing experiments that operate with a highly conducting shell avoid free-boundary instabilities. However, if the plasma duration is sufficiently long, the effect of any wall with finite electrical conductivity will eventually disappear, as the plasma duration exceeds the electrical penetration time of the shell. [2]

2.3 RWM CODES

Codes dedicated to the study of RWMs can be divided roughly into two groups:

- 2D thin wall approximation, codes which investigate thoroughly the physics of RWMs including kinetic effects into MHD equations but using a simplified description of the boundary.
- 3D geometry on the mode dynamics for a much more realistic description of the structures surrounding the plasma.

The control system of a plant is usually developed around these models. [19]

This work of thesis starts from the RFX CarMa code, literature, and bootstrap scripts to study the model itself and understand its internals and responses in an attempt to find a possible reduction. The attempt to reduce the model was aimed at a circuital analogous implementation, for the dominant instabilities.

The CarMa code provides a linearized model of plasma response as regards the dynamics of RWMs in the presence of active and passive conducting structures. The version, which was made available from the RFX Consortium, the code is derived from the coupling of MARS-F and CARIDDI. [19]

The CarMa works on the mesh, represented with its toroidal and poloidal gaps, with 2550 degrees of freedom describing the 3D current density in the shell. All the 192 independently fed active coils are represented in the mesh, with their actual geometry. Similarly, the 192 saddle loops provide the linked flux measurement (mean radial magnetic flux over the saddle area). [19]

2.4 CIRCUITAL ANALOGOUS APPROACH

Electronics circuit can be used to implement a physical representation of a dynamic model. The approach consists in writing the system in canonical control form, to have the most explicit input/output relationship and, once obtained the equation set, off-the-shelf components are used to implement it. This technique has advantages and drawbacks which derive both from the physical nature of the mathematical representation. The behaviour of the system becomes a physical system with electrically measurable quantities. On the immediate advantages some derive from the imperfection of the implementation, as electronic circuits are subject to noise, non-linearities, deviations from the nominal value, variations of the value due to the operating conditions (e.g., temperature). A system subject to these imperfections leads to a more robust implementation of the controllers. A physical implementation of a dynamic model can run real-time, compared to its mathematical representation, although the size limitation makes this advantage negligible. Circuital analogues become more and more difficult to implement as the number of inputs/outputs/states grows, thus hard limiting its employment in complex system representation.

This is the specific case of the CarMa model, which consists in a linear MIMO model with 192 inputs, 192 outputs and 2550 state variables. The size of the system and the internal dimension of its dynamics makes it hard and nearly impossible to be implemented practically with a circuital analogous, either considering analog, digital, or hybrid approaches. In order to propose a more compact model, the possibility to reduce the internal order of the dynamics has been investigated.

Usually, model order reduction is performed on the basis of some internal quantities of the linear system, which are invariant under state transformation, that is they are independent on the specific state space representation adopted and contain signature information on the input-output dynamics of the system. Model order reduction techniques are often based on balanced representations, that is representations for which a given signature information, related to controllability and/or observability or to passivity, is put in evidence. Open-loop balanced representations allow to determine the degree of controllability and observability of each state variables, thus providing information on their importance in the model. However, open-loop balanced representations, which are based on the diagonalization of the systems Gramian equations, assume a significance for model order reduction under the hypothesis of asymptotic stability of the dynamics. The singular values, which are the elements of the diagonal Gramians,

measure therefore the importance for the input-output relationship. In this case, the state variables associated to the lower degree of controllability and observability are less important from an input-output point of view and can be neglected, thus leading to a reduced order model of the dynamics. The error of the novel model is directly related to the sum of the singular values associated to the discarded variables.

The CarMa model is intrinsically unstable, with several modes associated to positive real-part poles, therefore a different model order reduction strategy must be exploited. Let us consider the closed-loop balanced representation which can be obtained also for unstable systems and is based on the diagonalization of the positive definite solutions of the Control Algebraic Riccati Equation and that of the Filtering Algebraic Riccati Equation. Thus, the closed-loop balanced representation highlights the importance of the state variables in a feedback control scheme. The characteristic values, which are the ordered quantities contained in the diagonal of the positive definite solutions in a closed-loop balanced representation, quantify this role. In this case, the order reduction, rather than be related to the model dynamics is referred to the dimension of the state which is actually fed back for control purposes.

Since the CarMa model is essentially oriented toward the control of the model behavior, the possibility of closing a feedback control loop which is based on reconstructing and feeding back a limited, reduced, number of state variables is of practical interest.

The characteristic values of the CarMa model have been computed and the highest 25 are reported ordered in the shown figure. As it is possible to observe, the first 14 characteristic values are of three orders of magnitude higher that the remaining 2536. Therefore, a control loop can be based on the feedback of only 14 variables, rather than the whole 2550 state dimension.

The number of state variables that must be fed back is therefore at least 14, in fact, if a lower number r of state variables is considered at least one eigenvalue assumes positive real part in the controlled system. This is according to a linear relationship, as shown above. Moreover, the compensator H(s) with r=14 fed back state variables is asymptotically stable.

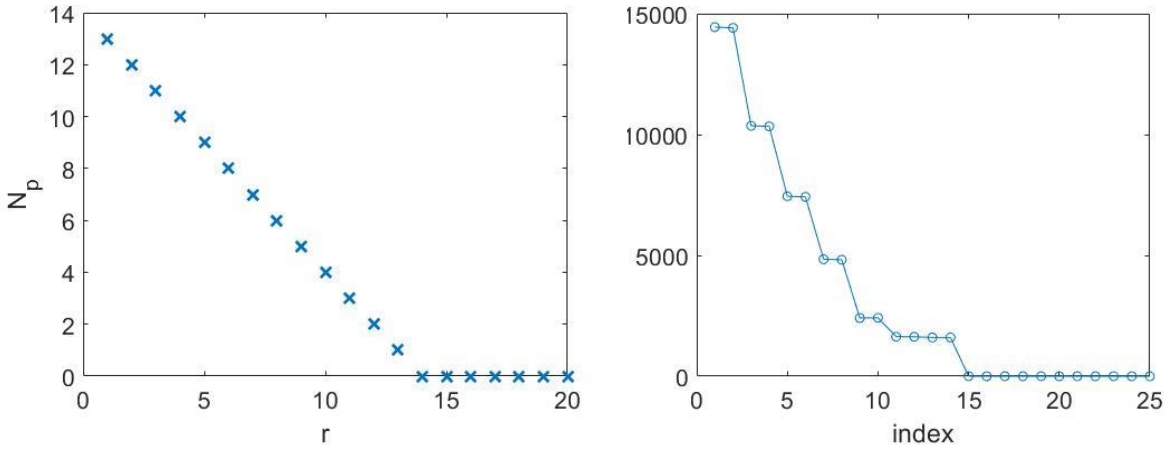


Figure 1 - Contribution of the first relevant group of singular values

CHAPTER 3. DEVELOPMENT FOR MARTE2 FRAMEWORK

Control systems in nuclear fusion experiments rely on a combination of custom and off-the-shelf hardware and software solutions. Having a standardised approach and a reference framework for the development of the control system software is an extremely important and desirable feature. Many nuclear fusion experiments rely on a software framework, MARTe2. This framework structure and implementation gives several advantages, both for the development of new components and for its porting on different architectures.

3.1 CHAPTER KEY POINTS

- MARTe2, a framework for the development of real-time control systems applications is used in many fusion experiments around the world.
- Its implementation philosophy allows developers, physicists and control systems experts to work together in team, with well-defined roles. Its Quality Assurance system renders it suitable for critical real-time control applications.
- Its code structure is specifically designed to allow the porting to other architectures and environments.
- A component to interface MARTe2 with PROFINET field I/O devices was implemented. It will become actively used and its implementation foresees larger employment scenarios.

3.2 INTRODUCTION TO THE FRAMEWORK

MARTe2 is a software framework, written in C++, specifically for the development of real-time control systems. One of the objectives of MARTe2 framework is to avoid the writing of highly specific (vertical) code, bound to an application and targeted to a platform and environment, which is a frequent practice when dealing with these scenarios. The verticalized approach, tightly bound to a hardware and software combination is not particularly suitable for growing project where multi-disciplinary teams are involved in the development. Moreover, reusability and maintainability factors play a key role, allowing the developers to establish a common code base.

MARTe2 offers two perspectives during the development of applications: developer and user. For the developer, MARTe2 establishes a clear separation of concerns between hardware interfaces, software algorithms, platform, and environment, allowing both the reuse of components and the ability to develop and test the control solutions in systems which may differ from the production environment. For the user, it offers a configuration system with a data driven approach, where sources, algorithms and sinks for the data follow a logical flow. Blocks expose data signals requiring to the user the specification of their interconnection scheme and real-time execution environment (CPU, thread).

MARTe2 is developed under strict quality assurance standards and processes and is MISRAC++ 2008 compliant. Moreover, the establishment and growth of the MARTe2 community has also exposed the existing code base to different applications, thus increasing its quality and robustness.

MARTe2 and its predecessor, MARTe, are deployed in many real-time fusion control systems, notably the JET tokamak and will be deployed in the ITER experiment plant instrumentation and control. [9] [20] [21] [22] [23] [24]

Its code can be divided into two main sets: hardware interfacing and user algorithms. Hardware interfaces are called *DataSources*, while user algorithms are called *GAMs* (Generic Application Modules). A MARTe2 application, called *RealTimeApplication*, is the result of the interconnection of *DataSources* and *GAMs.* [25] [26]

A generic application module, often referred to as GAM, is the MARTe2 component where the user algorithm resides. An extremely important concept in GAMs is that no interfaces with the hardware (direct or indirect) shall be implemented, except for the allocation of memory. A DataSource provides a real-time interface for the exchange of input and output signals to and from the hardware. Finally, the real-time application is the result of the interconnection of DataSources and GAMs together. GAMs are assigned to real-time threads, and states, which means specific execution units' selection and evolution of a global state machine controlled by the framework. [27] [28]

MARTe2 is engineered in tiers and layers: three tiers (BareMetal, FileSystem and Scheduler) and seven layers. Tiers are independent libraries, BareMetal is implemented to be independent from the other two, allowing the framework to be deployed also in resource-limited embedded systems. Each layer encapsulates functionalities and only depends on the facilities provided by levels below. [25]

3.3 MARTe2 QUALITY ASSURANCE

The development of MARTe2 follows a well-defined quality assurance process. This process is extremely important to ensure that the heterogeneous contributions from the community do not compromise the project overall quality. To increase the robustness of the code, MARTe2 uses only a controlled subset of C++, where all the aspects of the language that are considered dangerous for critical systems are removed. C++ version is ISO/IEC 14882:2003 (C++03) and coding rules are defined by the standard MISRA C++:2008. Unit and integration tests are implemented for each module of the framework, assuming black-box unit testing. Code coverage for accepted components must be above 80%. Source code is kept under versioning (git + GitLab) by following a specific flow, based on master/develop branches and user story (feature) branch. All feature branches are created from the develop branch and merged back only if quality checks are successfully passed. The agile workflow for the management of the resources and the lifecycle of the development is helper with Redmine tool. Code coverage is implemented with *gcov/lcov*, static analysis and compliance is performed using *Gimpel FlexeLint*. The Google test framework is used to perform the test routine, however a portable simplification was also developed as part of this thesis work, to suit the embedded system's needs. [29]

3.4 PROFINET® DATASOURCE

A PROFINET® DataSource was developed as part of this thesis work, to allow MARTe2 integration as slave periphery on the PROFINET® field communication bus. The DataSource, which is now included in the MARTe2 components official suite is also part of the ITER electron gyrotron heating (ECRH) factory acceptance test (FAT) tool suite [30] [31].

Purpose of the *ProfinetDataSource* is to abstract a PROFINET® slave in a PROFINET® bus. This slave appears on the bus as a real physical peripheral and can reflect its status (Input/Output) to and from MARTe2. The *ProfinetDataSource* was implemented primarily to suit the needs of the ITER ECRH FAT Tools, however the requirements were scaled to create a generic component which can be used also in different scenarios that will be discussed further in a specific section.

The *ProfinetDataSource*, as part of the MARTe2-components official bundle, was also put under QA and it is MISRA C++:2008 compliant for the relevant portions of it. It also includes

a comprehensive test unit, which runs within a soft-PLC environment the component validation logic.

3.5 PROFINET® STANDARD

3.5.1 INTRODUCTION

The description of the implementation work on the *ProfinetDataSource* relies heavily on concepts and practices of the PROFINET® protocol. For this reason, an introductory chapter on the argument will be given. For each described PROFINET® feature, the MARTe2 DataSource implemented counterpart is introduced, in order to have a clear spatial collocation of the work in the implementation description.

PROFINET® is a communication standard for automation, based on Ethernet IEEE 802 and described in IEC 61158 and IEC 61784. The cited standards form the basis for device or application-specific profiles, also creating planning, engineering and commissioning steps. Functions supported by PROFINET IO is divided into four *conformance classes* (CC), where each CC (CC-A, CC-B, CC-C, CC-D) provides a summary of the minimum properties. The standard follows the *producer*/*consumer* model to accomplish the data exchange. Three device classes are defined for PROFINET IO:

- **Controller**: the programmable logic controller on which the automation software runs (PLC). It *produces* the outputs and *consumes* input of the field IO devices.
- **Device**: a peripheral I/O device on the field. It *produces* the input data and *consumes* the output data.
- **Supervisor**: a field programming device used for commissioning (PD), a standard personal computer (PC) or a human-machine interface (HMI) used for diagnostic and supervisory purposes.

A plant must contain at least one controller and one or more IO devices. PD is usually integrated temporarily for commission and troubleshooting; PC and HMI can be integrated for the plant supervision and control. Automation devices can simultaneously fulfil the **controller** and **device** classes. [32]

The *ProfinetDataSource* behaves as a PROFINET IO **Device**, produces input and consumes output to and from MARTe2 real-time application.

3.5.2 IO DEVICE MODEL

The device access point (DAP) defines a device in terms of possible technical and functional features. The DAP is always placed in *slot 0 – subslot 0.* The *slot* and *subslot* structure designates a device as modular where the *slot* is the place where a module can be inserted. Each module occupies a *subslot.* Process inputs and outputs reside within a *subslot*. IO devices can be differentiated between *compact* and *modular*. Compact devices have a fixed slot/subslot structure while modular devices allow the user to change it at configuration time. [32]

Similarly, *ProfinetDataSource* is engineered as a *modular* IO device, where the slot/subslot configuration can be modified by intervening on the MARTe2 configuration file. The *ProfinetDataSource* allows the represented field IO device to assume an infinite set of input/output configurations, based on the description given at real-time application level. However, this description must be matched from the controller side, where a PROFINET counterpart description must reside. The PROFINET description of device characteristics is called General Station Description (GSD).

3.5.3 GENERAL STATION DESCRIPTION MARKUP LANGUAGE (GSDML)

Field device internal configuration resides into an XML-based description which is called GSDML (General Station Description Markup Language) [32]. The GSD file contains both relevant data for the engineering tool and for the IO controller expected data formats. It is usually supplied by the manufacturer. In the *ProfinetDataSource* there are two possible scenarios, which will be discussed in the dedicated chapter:

- MARTe2 configuration file is matched with the GSDML structure of an existing manufacturer device, in order to have a MARTe2 IO device behaving like the physical one (**emulation** mode).
- MARTe2 and GSDML files are built together to be matching, in order to build a custom MARTe2-PROFINET bridge (**bridging** mode).

3.5.4 APPLICATION AND COMMUNICATION RELATIONS (AR / CR)

During the start-up phase, a data exchange occurs between the IO controller and the IO device, to establish the communication paths. These paths, established during the setup allow the cyclic data exchange (IO), acyclic data exchange and alarms exchange. An IO controller can establish an application relation with an IO device, using one or several IOs for the data exchange. During the start-up, the AR is created specifying all the cyclic and acyclic I/O data, which is intended to be exchanged, plus the specification on expected modules/submodules layout. Matching

between the expected and actual configuration determines the initiation of the cyclic exchange, and eventually the arising of acyclic (alarms) condition. [32]

The *ProfinetDataSource* establishes one application relation and supports cyclic and acyclic data exchange with the IO controller. During the start-up phase the expected modules/submodules layout from the IO controller is received and matched with the MARTe2 configuration specified layout. The matching process (called modules/submodules plugging) determines the initiation of the cyclic exchange phase.

3.5.5 CYCLIC AND ACYCLIC DATA EXCHANGE

Data from cyclic exchange contains real-time representation of the inputs and outputs and is exchanged as real-time unacknowledged stream for a configurable period, called cycle time. Cycle times can be selected in a range from 250 µs up to 512 ms. Monitoring of the connection uses a multiple of the cycle time, where each consumer (caveat: a consumer is located on both communication sides) monitors the failure in arrival and sends a notification to the application. The cyclic exchange happens on layer 2 Ethernet by marking the packet with *Ether Type* = 0x8892. Acyclic exchange, instead, like diagnostic, identification and maintenance information (I&M) can be requested from any device at any time (notably, the engineering tool). The I&M data is subdivided in five blocks (IM0 to IM4) addressed separately by index. IM0 support is mandatory and contains the hardware and firmware data on the IO device. [32]

ProfinetDataSource supports the cyclic exchange at every PROFINET rate, given that the MARTe2 application is configured accordingly to be synchronised with the exchange. As the real-time application must maintain precise scheduling requirements, some expedients and adaptations were made at the interface between the stack and the DataSource. Acyclic exchange is also supported for I&M data for reading but writing is volatile, persistence is guaranteed only during the DataSource execution lifespan and is not applied on the matched MARTe2 configuration file.

3.5.6 PROFINET STACK

The *ProfinetDataSource* is based on the rt:labs p-net PROFINET device stack, that is a C stack supporting v2.4 level specification in conformance classes A and B. It provides a portability-ready layer and has a small footprint, suitable for embedded systems. C++ is also supported. The p-net stack is an IO device stack only, with support for cyclic and acyclic data exchange. It is open source with dual-licensing method (GPL v3 and commercial). The p-net library relies itself on an OSAL library (Operating System Abstraction Layer) which is also provided by

rt:labs [33] [34]. OSAL contains OS related features needed by the p-net stack (mutexes, semaphores, network access) [35]. As the OSAL library shares intentions with MARTe2 BareMetal and FileSystem tiers, it is also foreseen a deeper integration between the two, totally replacing the OSAL with MARTe2 itself.

3.6 PROFINETDATASOURCE IMPLEMENTATION

The *ProfinetDataSource* behaves as a PROFINET slave (IO device) connected to the PROFINET bus. It supports cyclic (real-time exchange) and acyclic (I&M exchange) reflecting MARTe2 inputs and outputs. The DataSource becomes an I/O for the PROFINET master (IO controller) thus reflecting its process image, having MARTe2 positioned on the field side of the IO device. The *ProfinetDataSource* can be generally subdivided in three layers:

- rt:labs p-net library: the stack upon which the *ProfinetDataSource* relies.
- *ProfinetDataSourceAdapter*: an adaptation layer between the MARTe2 and the stack world.
- *ProfinetDataSource*: the MARTe2 DataSource itself.

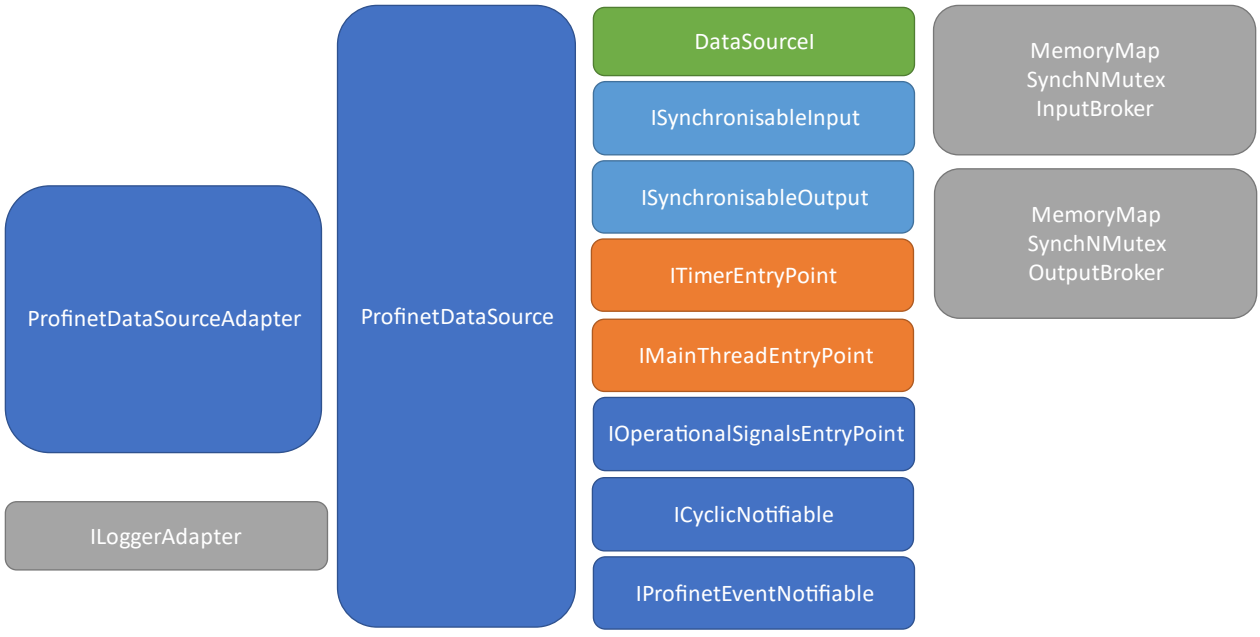


Figure 2 - ProfinetDataSource block diagram for the architecture

While the p-net layer was already described in the introductory paragraph, the core of the *ProfinetDataSource*, which represents one of the presented thesis contributions, will be described in their internals.

3.6.1 THE PROFINETDATASOURCEADAPTER

The *ProfinetDataSourceAdapter* is an adaptation layer between the p-net stack and the MARTe2 DataSource interface. It is not properly named after the Gang-Of-Four Adapter pattern [36], as it does not follow the prescribed implementation, however it shares the adaptation philosophy with it. The *ProfinetDataSourceAdapter* was designed with a standalone capability in mind, however its interface follows the MARTe2 DataSource philosophy thus its interface also suits the context interfacing needs.

The first adaptation layer provided is needed to map 1:1 the C stack implementation with the C++ OOP paradigm. The adapter defines a bank of callback mappers, defined as a first step into the p-net stack. These mappers are then both defined inside and outside the adapter, by using the free argument provided. With this strategy the p-net stack callbacks are moved inside OOP C++ logic to match the instance of the p-net library with the instance of the adapter class. In order to avoid disruption of the OOP data hiding principle, the callback bank is accepted into the adapter as C++ *friend* methods.

```
[…]

/* (A) */

profinetConfigurationHandle->connect_cb = pnetds_connect_ind;

[…]

/* (B) */

friend int pnetds_connect_ind(pnet_t * net, void * arg, uint32_t arep, pnet_result_t *
p_result);

[…]

/* (C) */

int pnetds_connect_ind(pnet_t *net, void *arg, uint32_t arep, pnet_result_t * p_result) {

return static_cast<ProfinetDataSourceAdapter*>(arg)->ConnectIndication(net, arep,
p_result);

}

[…]
```

In the above snippet, three relevant regions of the adapter code are marked (A, B, C) describing the overall adaptation strategy. The region (A) shows how the p-net handle expects the callback assignment for the specific connection indication. The connection indication expects a specific function with the signature shown in region (B), note that the method (function in this case) is marked as friend. In the region (C) the callback "`pnetds_connect_ind`" is mapped with the adapter "`ConnectIndication`" method with the instance of the adapter, which is passed to the callback as pass-through parameter `arg`. Note also that, to meet readability requirements of the MARTe2 QA process, functions belonging to the p-net stack callback (C structured context) follow *snake-case* convention while the inner MARTe2 calls (C++ OOP context) follow *Pascal Case* naming convention. The callback mapping involves fifteen internal p-net stack events which represents the evolution of the stack state machine in the PROFINET protocol implementation (e.g., connect/release, read/write, module/submodule plug/unplug, new data, alarm, reset and led). One important thing related to the stack is that, for each module and submodule, a portion of RAM memory must be allocated and mapped, for the callbacks to be able to read (produce) and write (consume) data. The memory management is overseen by the *ProfinetDataSource*, which allocates, based on the MARTe2 configuration file, memory for the signals represented by the IO device. This memory mapping process will be thoroughly described in the specific chapter. Allocated memory is shared between the p-net and the DataSource and its access is arbitrated using MARTe2 provided synchronisation and mutual exclusion primitives.

The *ProfinetDataSourceAdapter* exposes an interface to the MARTe2 *ProfinetDataSource* which is used to interact with the stack. Interaction is intended in terms of the definition of the behaviour of the stack as instructed by MARTe2 real-time application evolution. Exposed methods are related to the configuration of the device, with settings related to the a-cyclical (I&M) parameters (vendor, hardware and software revision, serial number), to the physical configuration in slots/subslots with the related modules/submodules ending with the configuration of the cyclical data exchange.

3.6.2 THE PROFINETDATASOURCE

The DataSource implements the MARTe2 aspects of the PROFINET exchange and can be reduced to a double-buffered heap of RAM memory. The heap is first split in two halves (inputs and outputs) and second in other two halves: one is exclusively accessed from MARTe2 while the other is shared with the *ProfinetDataSourceAdapter*. The latter is also passed to the p-net

stack to minimize the number of copies, thus positively impacting on the overall performances of the DataSource.

In MARTe2 terms, the ProfinetDataSource is a synchronised DataSource. Without going too deep in MARTe2 implementation details, a synchronised DataSource can constrain the real-time application task pace. When asking data to a synchronised DataSource, the entire application is locked waiting for the synchronisation point to happen. A synchronisation point usually corresponds to an event of data or peripheral readiness.

On the *ProfinetDataSource* synchronisation checkpoint two essential things are happening, at memory layout level:

- The MARTe2 input portion is copied in the adapter input portion.
- The adapter output portion is copied in the MARTe2 output portion.

In other words, the double-buffer mechanism moves data between the two portions of it, considering the data direction. The double buffer guarantees data independence and race avoidance. This mechanism was thought with the PROFINET stack in mind, as it is agnostic on the transported data at subslot unit (i.e., the variables contained into a subslot unit are moved as a whole binary-blob. MARTe2, on its side instead, knows the signals layout and uses the slot/subslot convention to group and place signals into the memory bank. Signals are wisely placed on a contiguous memory block and are assigned sequentially, from the MARTe2 side. Given the contiguous nature of the MARTe2 half of the buffer and the agnostic behaviour of the stack at signal level, the synchronisation operation is reduced to a single memory-copy step. This choice was fundamentally driven by the performance requirements of the DataSource and by the real-time nature of MARTe2 application.

Leveraging the dynamic signal handling capability of MARTe2, the ProfinetDataSource adds two further signals, LED and Ready, which give further information about the PROFINET stack status:

- The LED signal gives information about the peripheral status and can be controller both from the master and from the engineering tool. It is not particularly useful during normal runtime operations but can be used to virtually localise the peripheral. The LED presence is mandated by the PROFINET specifications.
- The Ready signal instead is extremely useful to inform the downstream MARTe2 GAMs about the data readiness, especially during start-up or problematic phases, as

during offline operations the DataSource keeps the last valid data. The ready signal thus is intended to be used from downstream GAMs to tell apart fresh from stale signal data.

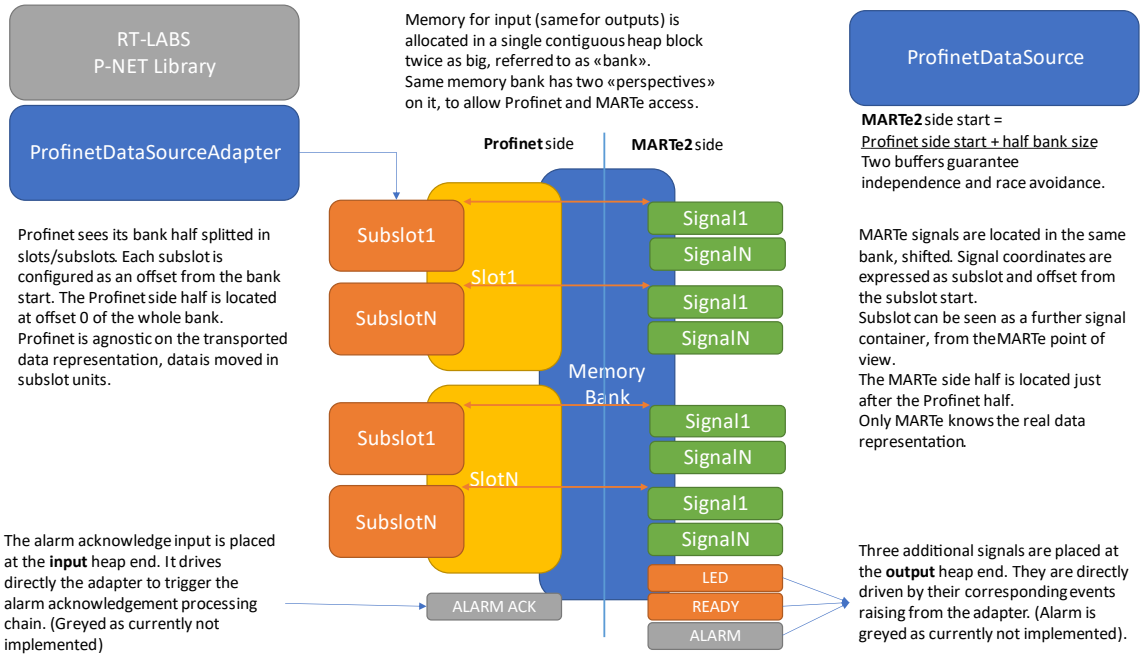


Figure 3 - ProfinetDataSource memory utilisation profile

The *ProfinetDataSource* relies on a complex interface inheritance layout. Aside from the MARTe2 framework mandated *DataSourceI* implementation, several interfaces were developed to accomplish the task. These interfaces are used mainly to realise decoupling between the *ProfinetDataSourceAdapter* and the *ProfinetDataSource*. The DataSource, adapter and brokers share a common philosophy: they all promote the tidiness and avoid over-coupling data structures and modules. For these reasons every coupling between the different aspects of the DataSource happens through the dedicated interface. Accelerator structures are also implemented, especially to quickly map between PROFINET and MARTe2 signals. Interfaces between the DataSource internals are:

- *ICyclicNotifiable*: offers an interface for the cyclic notification of an event. Relies on the *NotifyCycle* method entry point. It is used by the adapter in its cyclic update loop to notify the DataSource about the correct completion of an update loop where both the IOxS (producer - IOPS and consumer – IOCS) were updated. The cyclic update is regulated by a MARTe2 service thread which runs at the PROFINET peripheral scan rate.
- *ITimerEntryPoint*: provides the entry point for the timer service, exposing the *TimerTick* method. The *TimerTick* method is called at the PROFINET peripheral scan rate.

- *IMainThreadEntryPoint*: provides the entry point for the main thread executor, relying on the *MainThread* method, which uses a status flag as input and returns the processed status flags.
- *IProfinetEventNotifiable*: provides the interface for the listener, to allow underlying stack bubble events to the upper layer, bringing the event type.
- *ISynchronisableInput* / *ISynchronisableOutput*: these two interfaces provide the synchronisation entry point, as a specialisation of the MARTe2 Synchronise method, where differentiation between input and output synchronisation shall occur.
- *IOperationalSignalsEntryPoint*: provides the interface for the listener to bubble the LED and Ready status signal change. These two signals are referred to as operational or ancillary signals.

3.6.3 HELPERS

The ProfinetDataSource relies its operations on two helper classes, which are in turn relying on the MARTe2 thread executor service. The *ProfinetMainThreadHelper* and *ProfinetTimerHelper*:

- The *ProfinetMainThreadHelper* stays idle waiting for an event to occur. Once an event has occurred, the right handler is executed. The handle selection relies on the status flag, previously described, where input events are processed, and the output flag sees them cleared once they are served by the underlying system.
- The *ProfinetTimerHelper* is the vital event for the PROFINET stack. The timer ticks at the IO device declared rate, regulating the whole stack execution. Note that also the timer is an event, and its occurrence is handled in the *ProfinetMainThreadHelper*.

Regardless of the source or helper, the handling logic is always executed inside the *ProfinetDataSourceAdapter*.

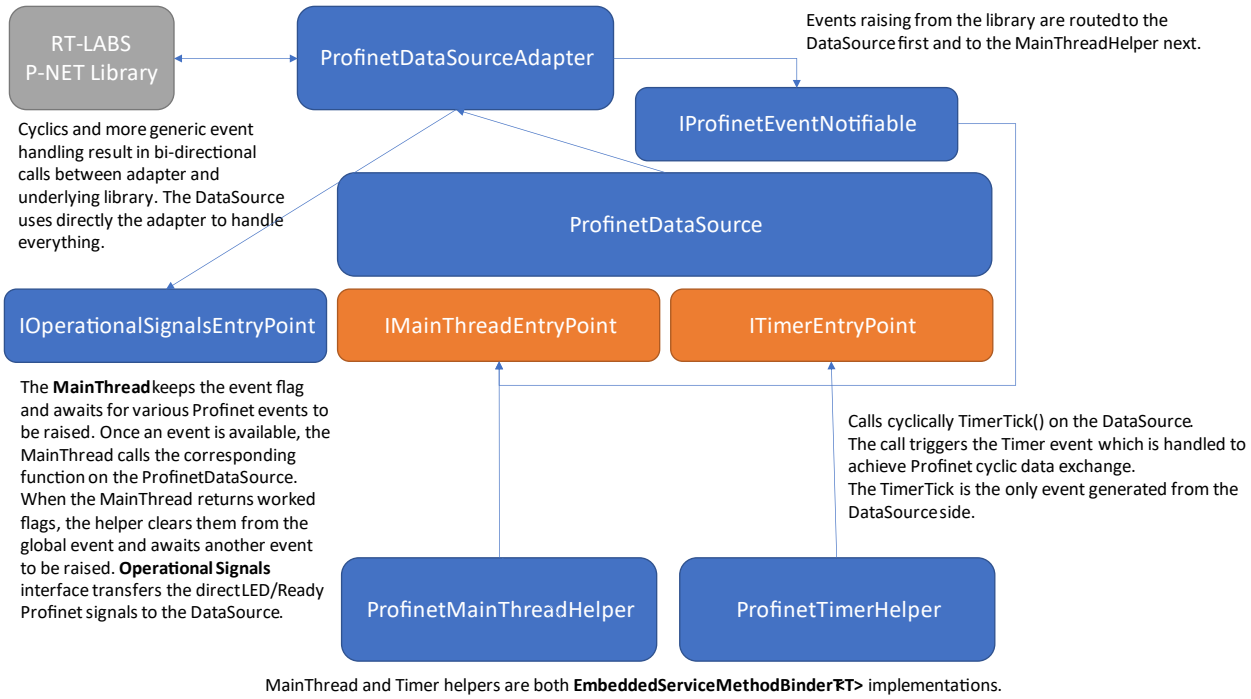


Figure 4 - ProfinetDataSource scheme of interactions among modules

3.6.4 INPUT AND OUTPUT BROKERS

In MARTe2 terminology, a broker is the interface between the GAM memory and the DataSource hardware data (memory in the specific case). The brokers are called before and after the GAM execution to actualise the required input (before) and outputs (after) the user algorithm is executed.

The *ProfinetDataSource* relies on two custom-implemented brokers, to suit the specific needs. These brokers (*MemoryMapSynchNMutex [Input, Output] Broker*) share the logic with the MARTe2 bundled *MemoryMapSynchronised* brokers. Considering the heap (I/O memory bank with process image) is shared between the DataSource and the adapter/stack, these two brokers implement a different synchronisation/termination phase. This customisation logic was a structural design choice, to avoid the p-net stack side locking and, consequent impairment, of the MARTe2 real-time application cycle time. In fact, MARTe2 side always uses a Try-Lock mechanism in the broker which, in the event of failure of the PROFINET stack (especially with an acquired lock on the heap), translates only into a stale data propagation. Obviously, to maintain general real-time application requirements, careful tuning between the PROFINET and MARTe2 tasks must take place. For this reason, a parameter called *Reduction Ratio*, was also implemented to avoid copying and updating data every cycle while keeping the IO device online for the IO controller. This can be also said in PROFINET stack terms equivalent, as only

an IOCS occurs (consuming) at the timer tick event occurrence. The IOPS (production) of data is skipped, for the copying event between the two sides of the double buffer to be avoided.

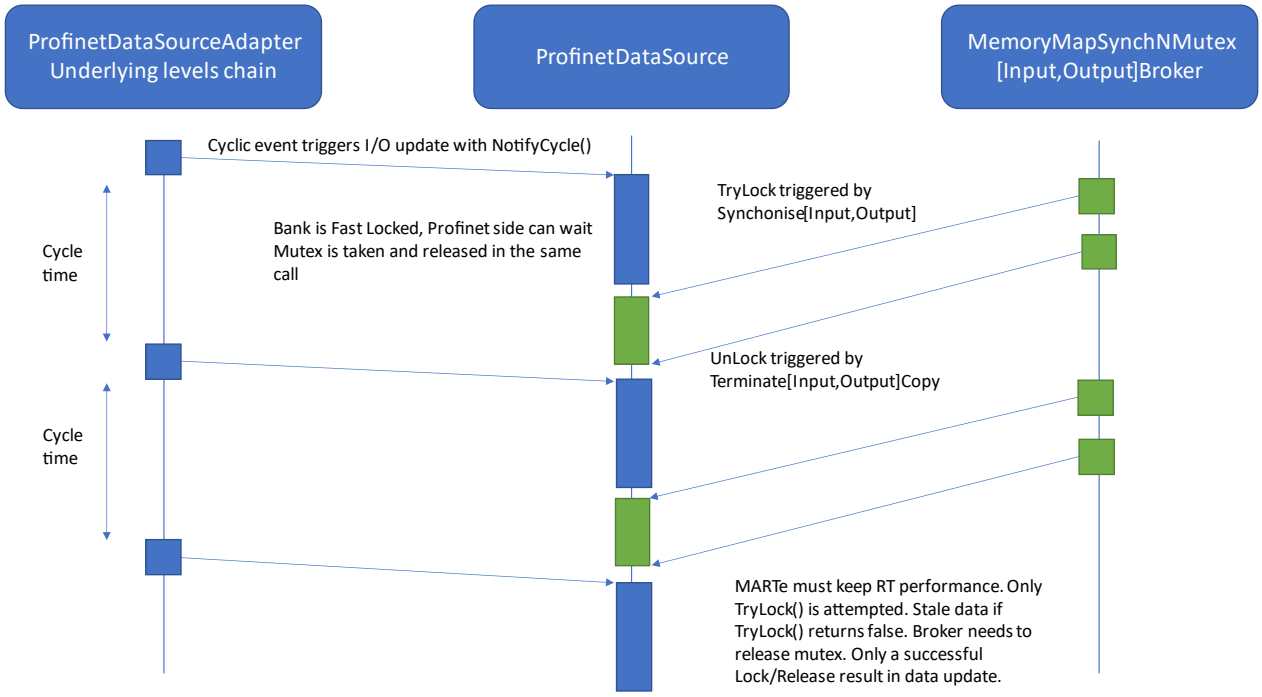


Figure 5 - Interaction between DataSource and custom brokers

3.6.5 Configuration and operation

As a MARTe2 component, the ProfinetDataSource requires a specific section in the MARTe2 configuration file. Aside from the MARTe2 requirements, this section in the configuration file must follow a particular scheme which reflects the PROFINET stack internals and the GSDML peripheral descriptor. The matching between the GSDML and MARTe2 configuration file is key to the correct operation of the DataSource and demonstrates its ability of to represent virtually any type of IO device.

The module which will be “represented” by MARTe2 must have a slots/subslots layout known in advance. As previously stated, slots and subslots are a sort of virtual bays which contain the I/O modules and submodules. An engineering tool is used to configure the IO controller (PLC) by importing the GSDML descriptor file. The engineering tool usually provides an interface which allows, given the supported peripheral slots/subslots, the insertion of modules and submodule. This step *materialises* the IO device configuration scheme.

When the ProfinetDataSource starts, using the slot/subslot layout contained in the MARTe2 configuration file, the stack *expects* a configuration which essentially informs it that a particular subslot has a specific data direction (input/output) and is represented with a certain number of

bytes. Key to the understanding of the whole process is the usage of the term *expects*. The expectation resides on the fact that, once started, the IO controller, based on the GSDML configuration, *sends* a specific layout. The matching step with the *expectations*, occurring in the PROFINET stack, is called *plugging* and happens during the start-up phase. The plugging event triggers the generation of the IO controller *sent* layout which is then matched internally with the DataSource *expected* layout. These terms are used consistently across the configuration files, respecting those used across the PROFINET standard and, consequently on the stack side. The ProfinetDataSource is specifically built to match 1:1 the GSDML description file.

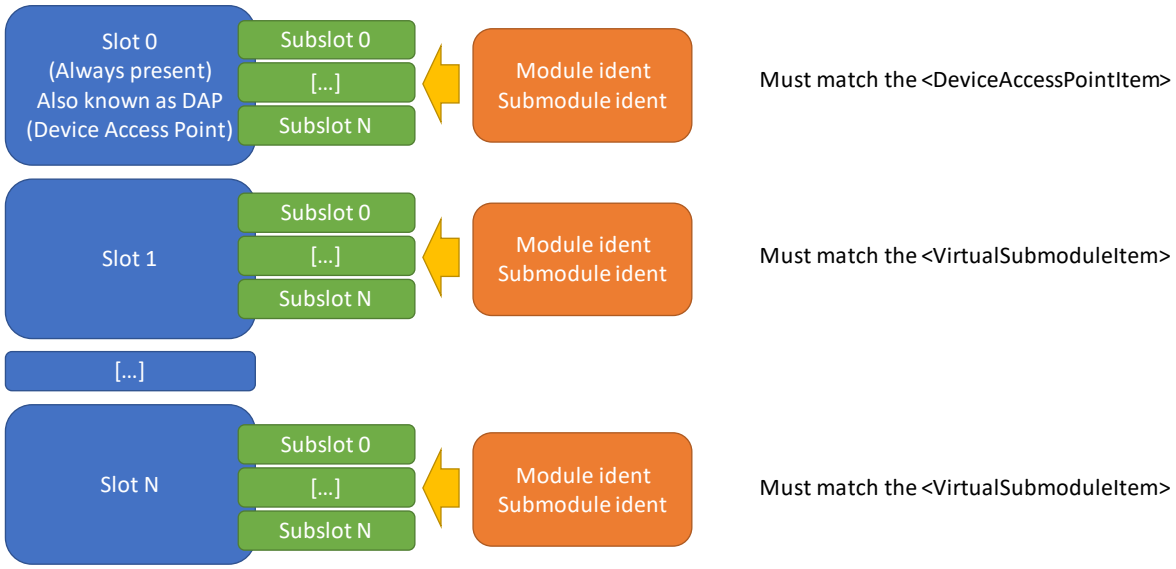


Figure 6 - Plugging and layout

The configuration scheme needed into the MARTe2 configuration file strictly adheres to the PROFINET standard for the peripherals, retracing concepts previously described in the protocol introduction. Basic configurations need the network interface for the binding, the station name, the periodic interval and reduction ratio. As a requirement, the underlying operating system (or bare metal) must be able to send full-sized L2 raw ethernet frames. Except for the network interface and reduction ratio, the parameters need to be strictly matching to them described in the GSDML file, following this scheme:

Table 2 MARTe2 to GSDML base parameters

MARTe2 cfg	**GSDML**
StationName	ApplicationProcess/DeviceAccessPointList/DeviceAccessPointItem
PeriodicInterval	ApplicationProcess/DeviceAccessPointList/MinDeviceInterval

For the base identification and for the I&M data, same matching rules apply, as depicted in the following two excerpts.

```
VendorIdentifier = 0xFEED [1]
DeviceIdentifier = 0xBEEF [2]
OEMVendorIdentifier = 0xC0FF [3]
OEMDeviceIdentifier = 0xEE01 [4]
DeviceVendor = "rt-labs" [5]
ManufacturerSpecificString = "PNET demo"
[6]
```

```
<DeviceIdentity VendorID="0xfeed[1]"
DeviceID="0xbeef[2]">
    <InfoText TextId="IDT_INFO_Device"/>
    <VendorName Value="rt-labs[5]"/>
</DeviceIdentity>
```

In the snipped above, notice how the (1), (2) and (5) keys have a matching counterpart, while (3), (4) and (6) can be freely customised. Again, same rules apply for the whole I&M data block.

The slot and subslot layout, as previously stated, defines the layout of the IO device. This layout represents the exact module configuration which must match the GSDML configuration. The configuration file uses the *Expected* suffix to underline the concept previously described.

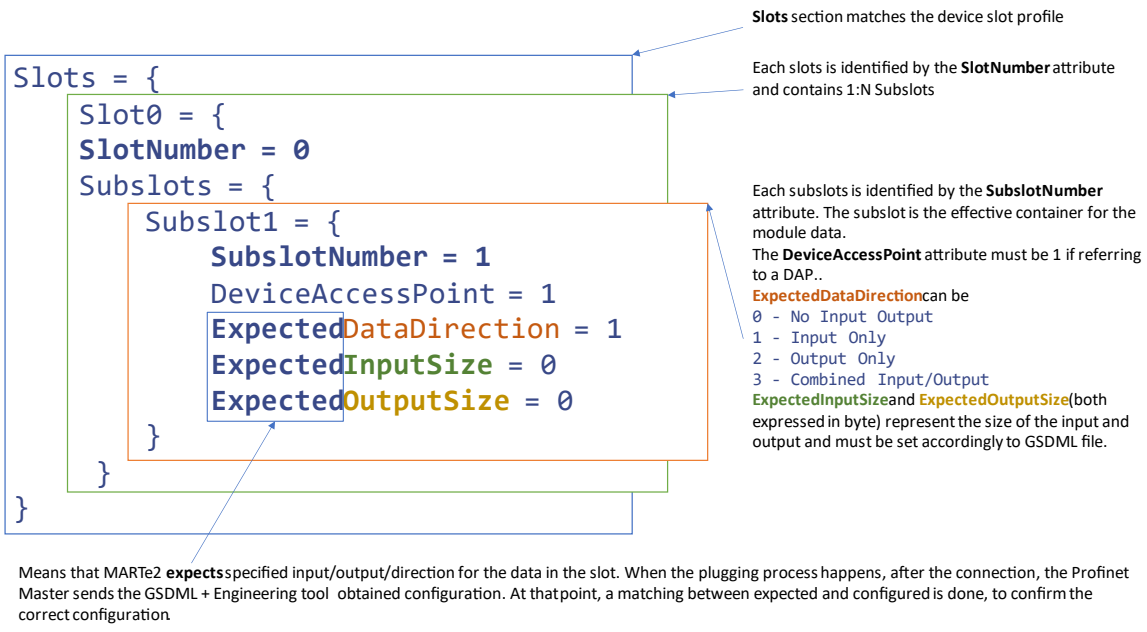


Figure 7 - Slots and subslots structure

The MARTe2 signal related portion of the configuration file need additional keys to place the signal in the correct slot/subslot/offset (i.e., the position in the memory bank) plus a direction

attribute (input/output). The last parameter, *NeedsSwapping* is used to manage the endianness between the PROFINET standard and the architecture standard. It should be set to 1 (true – enabled) to let PROFINET manage automatically the endianness but can be disabled if a binary blob needs to be transferred. Moreover, an internal mapper takes care of the conversion between IEC 61131 types and MARTe2 types, to make sure the data exchange is consistent.

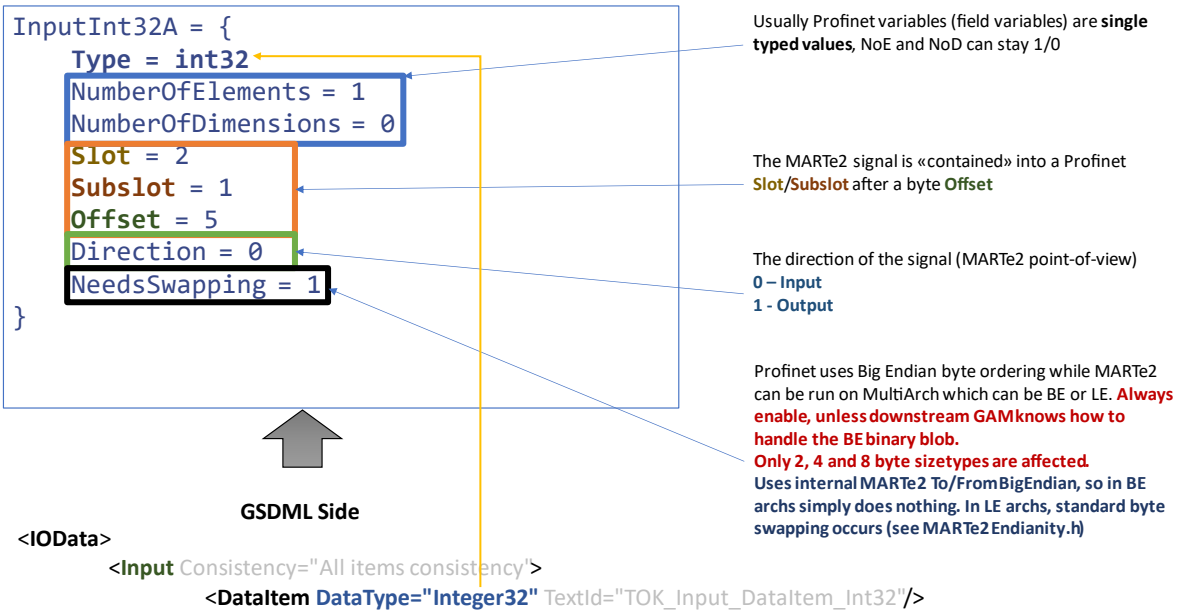


Figure 8 - Signal declaration and GSDML matching counterpart

Last words on the ancillary signals, previously referred to as operational signals. They are also made available from the DataSource and are used to represent the PROFINET IO device LED status and the readiness signal. The latter comes handy to inform downstream GAMs that data coming from the DataSource is stale and should not be considered for the computation. This may happen during the start-up phase but can also happen if any problem occurs.

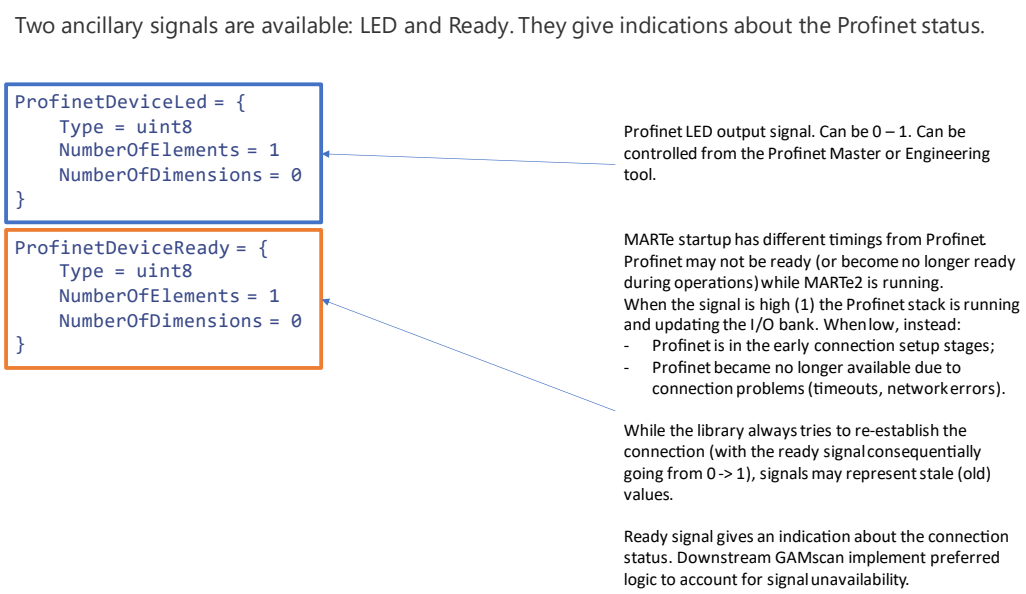


Figure 9 - Ancillary (operational) signals layout

3.6.6 DEVELOPMENT AND TEST

The ProfinetDataSource was entirely developed, from scratch, as part of the collaboration with Fusion for Energy. Its main purpose is to support the factory acceptance tests (FAT) for the Electron Cyclotron Resonance Heating system at the ITER plant (ECRH) [31].

It is currently part of the MARTe2-components bundle (Components / DataSources / ProfinetDataSource) currently hosted on the F4E GitLab and GitHub servers [30]. It is also under QA, has a comprehensive suite of tests to bring the coverage to the community standards and is also kept under the F4E continuous integration (CI/CD) system. The DataSource was also successfully linted against the MARTe2 MISRA C++:2008 rule set, for the relevant part (ProfinetDataSource and ancillary classes and interfaces). Portions of the adapter, due to its internal dependencies on the p-net stack, are not subject to the linting process.

During the development steps, to emulate the IO controller, a CODESYS instance was deployed on a Raspberry Pi model 3B (CODESYS Control for Raspberry Pi SL). This instance allows the execution of a PLC on a single board computer. As engineering tool, an instance of CODESYS was also installed on a standard machine, to load and test configuration code and GSDML on the PLC. The CODESYS environment was chosen to allow a comfortable environment for the continuous integration toolchain. The CI toolchain test is based on a run of a fully-fledged MARTe2 real-time application with the DataSource and a simple “loopback code” on the PLC which copies input on outputs and the MARTe2 application matching sent

with received data. The loopback uses all the different data-formats available to consider also representation problems.

The ProfinetDataSource also underwent an on-field testing using the production PLC environment (Siemens SIMATIC S7-1500 equipped with PROFINET interface) and engineering tool (Siemens TIA Portal) to assess the validity and correctness of the DataSource operation.

The extensive test campaign allowed the discovery of a problem in the p-net library. The opened issue (issue #315 – Correct Shutdown Sequence) allowed the stack library developers to fix the encountered bug with the proposed solution. The commit (Set SO_REUSEADDR for Linux UDP sockets to enable fast restarts) closes the case and was merged into the main branch of the rt:labs p-net library, available on their GitHub official repository (close #374 for issue #315) [37].

3.6.7 PROFINETDATASOURCE CONTEXT

ITER ECRH & CD system

The ITER Electron Cyclotron Resonance Heating and Current Drive (ECRH&CD) system is designed to inject 20 MW of millimetre-wave at 170 GHz into the vacuum vessel. The EC system is composed of different sub-systems, each one with its own local controller called Subsystem Control Unit (SCU). The main sub-systems are the High Voltage Power Supplies (HVPS), RF sources (Gyrotrons), Transmission Lines (TL), Ex-vessel Waveguides (EW) and Launchers. The integration of all the sub-systems, the system preparation for operation and the execution of real-time requests coming from the plasma control system are some of the main EC Plant Controller (ECPC) functions.

The main functions of the ECPC are:

- Manage EC subsystems parameters and waveforms
- Allow operation of each single EC subsystem
- Coordinate the state machines of the EC plant
- Control the EC plant following real-time requests and references coming from the PCS
- Implement fast and slow protection functions
- Publish to CODAC (Control, Data Access and Communication) signals to be monitored

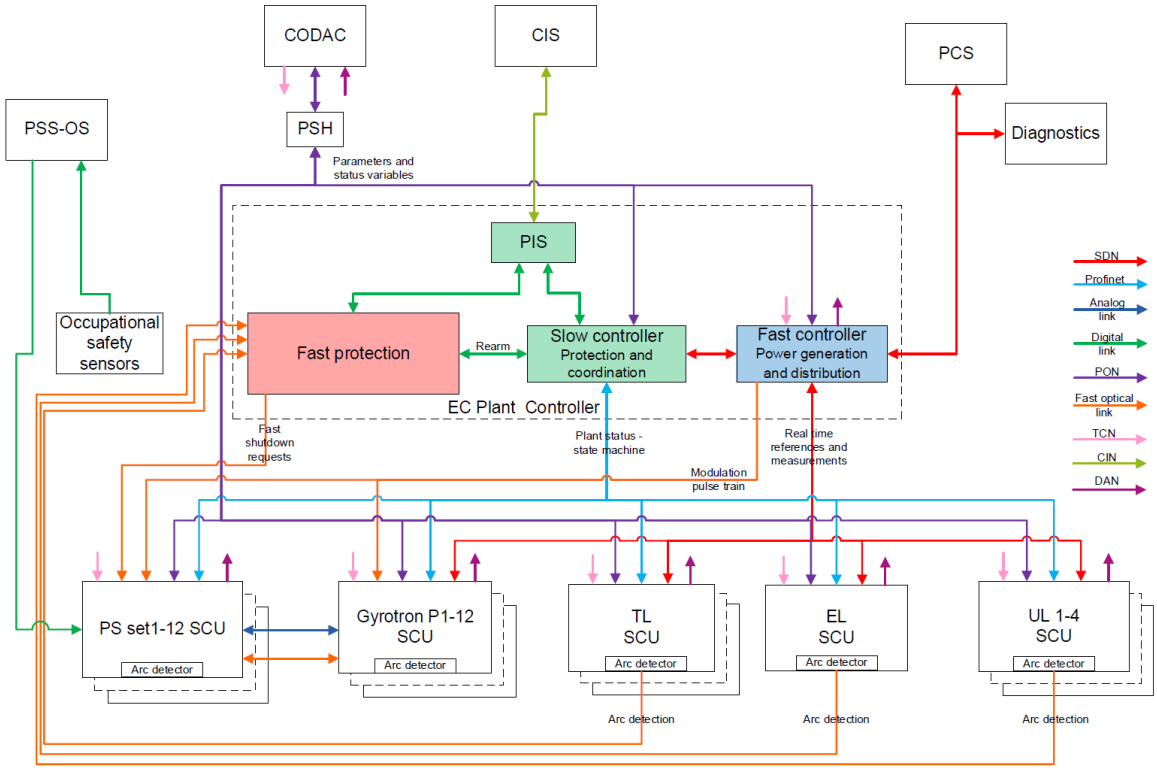


Figure 10 - ECCS Architecture and, in dotted rectangle, the ECPC

The ECPC interfaces with the ITER central I&C, which is composed of the following control and protection functions:

- The Central Safety System (CSS): plant-wide nuclear (CSS-N) and occupational safety (CSS-OS)
- The Central Interlock System (CIS): plant-wide investment protection
- CODAC Systems and Networks: overall plant systems coordination, supervision, plant status monitoring, alarm handling, data archiving, plan visualization (HMI) and remote experiment functions.

The ECPC state machine is the highest node in the hierarchy of the state machines. It interfaces to Central I&C and coordinates all the systems of the plant which are to be operated synchronously [38].

FAT-Tools and ProfinetDataSource

The factory acceptance testing tools system (FAT-Tools) is a hardware/software toolset used to emulate systems which are connected to and controlled by the ECPC, to validate the correct operation of the control logic. The emulation takes place in emulated sub-system control units which are functionally equivalent to the real ones. The ProfinetDataSource is used in the SCU2,

where the FAT-Tools project provides a SCU-PLC in the form of a MARTe2 application, that is used for the SCUs simulation. The objective of the SCU2-PLC MARTe2 simulator application is to simulate a PLC, allowing communication with ECPC-PLC through PROFINET [31].

3.6.8 EXTENDED USAGE SCENARIOS AND CONCLUSIONS

While the ProfinetDataSource need was dictated by the needs of the ECRH FAT-Tools suite scenario, its development and vision achieved a wider result than the basic intended. In fact, the ProfinetDataSource can be used in the entirety of scenarios where the interfacing with PROFINET bus is needed. Two main operating modes were identified:

- **Slave emulation** mode: the DataSource can be used to emulate the field, to assess, validate and test PLC software algorithms. The field data, generated inside specific MARTe2 applications, can be presented in the PROFINET bus with the DataSource, by perfectly emulating a manufacturer IO device and its GSDML description. Testing the PLC logic, especially without the field, can be seen as an accelerator and helps improving the overall quality of the code, also taking into account the number of tests that can be driven by emulating each possible field condition in a production-ready environment. The ProfinetDataSource, with MARTe2 driven field data supply, can emulate almost every field configuration, presenting data at the IO device inputs and behaving like a controlled field.
- **Bridging** mode: the DataSource can be used to bridge the industrial field control with other worlds that are not directly compatible (e.g., Data Acquisition Cards) by using MARTe2 as a bridge and translating signals between the worlds. The bridging mode also allows the creation of “intelligent” field peripherals, that are seen by the field as IO devices but can run opportunely configured MARTe2 instances to accomplish logics that are not normally available in this kind of devices (signal pre-processing, complex logic). This scenario is extremely interesting, considering the portability of MARTe2 (also part of this thesis work) on microcontroller-based devices. What happens is essentially that a MARTe2 real-time application can source or sink data from the ProfinetDataSource. Sourcing (or sinking) signals from the PROFINET bus allows slow controllers (PLC) to send or receive) data to/from otherwise non-compatible devices and viceversa, using MARTe2 as bridge to adapt data between the two worlds.

The bridging mode can also be seen, in a wider vision, as a first step to move towards the integration between the off-the-shelf solutions typical of the ICS world with the custom solutions of the PCS world [39]. In this vision, the two systems, that actually require a separation between the two contexts, can also be integrated together towards a totally integrated plant monitoring and control system. The interesting scenario brought by the merging of the PROFINET bridging use case with the MARTe2 porting on ARM platforms, especially for the STM32 series of microcontrollers, opens interesting ways to rapidly develop powerful control systems solutions, ready for the field application.

CHAPTER 4. MARTE2 PORTING ON ARM

The MARTe2 tiers and layers structure also encompass the definition of an additional subdivision, with respect to previously described layered structure. Two sub-layers, *architecture* and *environment* contain the definition of the hardware (architecture) related aspects of the framework and underlying software connections (environment). As the tier/layer structure is tightly followed, in terms of dependency chain, this further differentiation allows the porting of the framework become a streamlined process [25].

Porting MARTe2 to another architecture, reduces the activity to the writing of the two sub-layers code (architecture and environment), located inside the lowermost (L0 and L1) and uppermost (L6) layers of the framework structure. The fact that there are no explicit dependencies on the platform, outside of the designated containers, allows the porting of the framework also to bare-metal platforms, without an underlying operating system.

4.1 CHAPTER KEY-POINTS

- Porting was developed starting with several platforms, to abstract and generalise the procedure. This step resulted in core framework modifications.
- The activity landed to the final production environment, allowing MARTe2 to run on high performance embedded platforms.
- The target embedded platform and specifically developed MARTe2 applications will run in the magnetics diagnostics system to acquire, process and stream sensor data at the ITER experiment.
- The sensor data processing application required further development of MARTe2 components.
- All the developed modifications, porting and components become part of the MARTe2 framework.

4.2 INTRODUCTION TO THE PORTING

The porting of MARTe2 was developed as part of the traineeship collaboration with Fusion for Energy, to create a signal processing platform for the ITER magnetics diagnostics project. The

porting main objective was to run MARTe2 on two custom boards based on the AMD Xilinx Zynq Ultrascale+ SoC (Quad ARM Cortex A53 + Dual ARM Cortex R5 + ARM Mali 400 + FPGA). On the two boards, two specific MARTe2 applications are executed to process the sensors signal from the ITER tokamak vessel-positioned magnetic probes.

An operating scheme, not very common for multi-core CPUs, called Asymmetrical Multi-Processing (AMP), sees the quad-core CPU as four single-core independent units, each one running a separate MARTe2 application. The on-board RAM is split in 4 partitions and a complex scheme based on shared memory is used to share signals data between cores. On one of the two bespoke hardware solutions, the three independent MARTe2 applications are also different: two of them run a bare-metal porting while one is based on a real-time operating system. While the first uses the full core power to process the ADC data coming from the magnetic probes, the latter runs a monitoring, diagnostic and control application on the real-time OS.

The porting activity also included the generalisation of the original MARTe2 porting scheme, together with a porting *stub* that can be used, and the targeting of ARM Cortex F4 series of microcontrollers, which are common is STMicroelectronics STM32 series and the Raspberry Pi's Cortex A53/72, again in bare-metal configuration. The first, MCU based target, running MARTe2, represents a perfect solution for the rapid development and deploying of cost-effective field control solutions.

In this chapter, the thesis work in the field will be explored, starting with an introduction to all the techniques and challenges that were encountered during the development.

4.3 EMBEDDED SYSTEMS AND ARM

On the yearly production of around ten billion of processors, only the 2% of them becomes the central part of personal computers. The largest shareholders of processors are the embedded systems manufacturers. Every modern device, from household and toys to the vehicles and plant controllers run a processor with a custom software to do a specific thing. This leads to the definition of *embedded system*, a combination of hardware and software designed to perform a specific task. Some definitions to the embedded system add other clauses or restrictions to the term, but the essence is that the system becomes embedded when the controller unit is a fundamental part of a complete device relying on a bespoke electrical, electronic and sometimes

mechanical solution. Embedded systems are dedicated to a specific task, leading engineering processes to optimize performances, energy usage, reliability or every other aspect which is key to the success of the controlled system.

The thesis work is centred on two classes of embedded systems: the System-on-Chip (SoC) and the Microcontroller (MCU):

- System-on-Chip: they integrate most (or all) components of an electronic system into a single die. A SoC may integrate CPU, GPU, I/O devices, FPGA, RAM and secondary storage, DSP and everything else needed for the target application. SoCs are the hearth of smartphones, tablets and mobile/edge computing systems.
- Microcontroller: in an approach like the SoC, microcontrollers tend to integrate the needed peripherals inside the chip die. Microcontrollers tend to integrate less computational and more field related peripherals (i.e., no GPU, lower performance cores, less RAM, little secondary storage but lots of I/O, ADC/DACs, interconnection buses like I2C, SPI, USART, CAN). MCUs are the hearth of most automation controllers' systems.

The AMD Xilinx Zynq Ultrascale+ belongs to the SoC class of embedded systems, as previously stated it integrates four ARM Cortex A53 cores, Two ARM Cortex R5 cores, an ARM Mali 400 GPU and an AMD Xilinx FPGA, plus a series of ancillary devices. The STM32 F-series belongs to the MCU class of embedded systems, it integrates a single ARM Cortex M core aside a multitude of external interfaces (ADC/DACs, general purpose I/O, I2C, SPI, CAN, USART) [40].

ARM is a semiconductor design company based in England. It provides design for semiconductor intellectual property core (SIP or IP cores), reusable components which can be used in the design of systems. Usually, an ARM core IP is bought by an original design manufacturer, which integrates it with other parts and peripherals to produce a complete device which can be successively built by a semiconductor fabrication plant. Smartphones are the most notable example of this kind of market, where Samsung Exynos or Apple A/M series are, essentially ARM Cortex cores IPs with customisation and integration of application-oriented peripherals (radio/modem, notably).

4.4 BARE-METAL VS OPERATING SYSTEM

In a usual configuration, a CPU is used in conjunction with an operating system which manages and mediates access to all the hardware available on the platform, by means of drivers. User code, sometimes called *programs* run in the context of the operating system. Reference to the hardware is not always necessary in a user program which runs under an operating system, system calls are used instead, to ask access to a peripheral, by means of an operating system driver. The driver oversees accessing the hardware while the OS mediates and abstracts the call into a more generic one. A simple example is represented by the access to the file system or the network, where, instead of caring of network or disk related issues (disk geometry, sectors, heads, physical layers, media converters), the user simply calls an OS primitive (connect, bind, send, receive, open, close, read, write, …). Among the various tasks accomplished by the operating system we can find the core and memory management and the sharing of the resources (time, memory, execution unit).

An operating system takes care of a multitude of behind-the-scenes tasks, allowing the user to concentrate on the application itself and not on the management. This approach eases the development of applications but at the cost of a reduction in overall performances (not always true but realistic, as long as the resource usage of the user application moves close to the performance limits of the platform), especially in terms of jitter (fluctuations around periodicity).

In complete contrast with the Operating System approach, the bare-metal programming has no supporting operating system taking care of the sharing, mediation and arbitration on the available resources. The user code is the only code running on the platform and is responsible for everything happening. Drivers are usually a mere layer of libraries (hardware abstraction libraries – HAL) and hardware access is almost direct.

A halfway solution between bare-metal and operating system is the availability of real-time operating systems (RTOS). There are many solutions, from commercial to open-source ones. The key in a RTOS is the complete control over the execution time and deadlines of all the tasks running on the platform. Some real-time operating systems offer an almost bare-metal approach where a minimal real-time scheduler with abstractions related to CPU time sharing management are offered. In some RTOSes, networking stack and filesystem access is achieved similarly to bare-metal approach (libraries relying upon hardware abstraction layers, like lwIP or FatFs).

The presented work of thesis, in the topic of the MARTe2 porting, deals with both the bare-metal and real-time operating system porting of the framework on ARM cores. A key understanding is that MARTe2 offers layers that can be mapped both directly on hardware (bare-metal) and upon an operating system (OS, also RTOS). The upper layers are hinged on these contact points and have no further dependencies, allowing the porting to virtually any device.

4.5 Asymmetrical vs Symmetrical Multiprocessing

Common embedded systems available nowadays rely on multiple core (or multiple CPU) on the same die. From software point of view, two strategies are possible: Asymmetrical and Symmetrical Multiprocessing (AMP vs SMP). In Asymmetrical multiprocessing configuration each core (or CPU) operates as it is alone on the platform. In Symmetrical multiprocessing, the operating system (or a software called hypervisor) runs on all the cores and manages its workload across them. AMP mode can be achieved also on different cores and CPUs (heterogeneous vs homogeneous). For SMP to happen, instead, the system must be homogeneous (all cores or CPUs of the same identical architecture). SMP is an extremely common approach, it is used by the almost every operating system (Linux, *NIX, Windows, etc.) while AMP is not very common and is used in very particular applications, especially in its bare-metal AMP flavour.

The thesis work deals with the porting of MARTe2 on:

- Bare-metal AMP (AMD Xilinx Zynq Ultrascale+ Cortex A53)
- FreeRTOS AMP (AMD Xilinx Zynq Ultrascale+ Cortex A53)
- HAL-based single core and AMP (STM32 ARM Cortex M4, ARM Cortex A53/A72)

The first two porting flavours, bare-metal and FreeRTOS AMP on the Ultrascale+ platform, are used for the ITER Magnetics Diagnostic bespoke electronics. These boards oversee the processing sub-system connected to the magnetic probes using a combination of FPGA and MARTe2 processing to produce filtered streams of data coming from the magnetic sensors.

The scope of the work is planted on the porting of MARTe2 platform, on the transfer of the FPGA processed data inside the MARTe2 real-time application, on the inter-core, real-time data exchange in AMP configurations and on the coexistence of bare-metal and FreeRTOS on three cores of the same CPU. The work proceeded along a staged pipeline of tasks with short

term objectives where all of them became part of the MARTe2 framework bundle, the specific Ultrascale+ application instead is part of the 55.A0 Magnetics Electronics and Software interface project, serving as interface between the magnetic diagnostic sensors and the ITER CODAC I&C networks. Some side projects, like the Raspberry Pi and STM32 portings still need some refinements to reach the quality level needed to be released, however they represent an extremely important milestone and the growing interest, especially towards the STM32 MCU porting, will surely impact the future development objectives.

4.6 MARTe2 Code organisation

As previously stated, MARTe2 is organized in tiers and layers, where the lowermost (L0 and L1) and the uppermost (L6) layers contain two sub-level differentiation: *Architecture* and *Environment*. The architecture sublevel contains abstraction related to the hardware platform while the environment contains abstractions related to the underlying software or operating system, specifically the layer interested in the porting are:

- BareMetal
    - L0 Types contains the definition of the basic types used across the framework (architecture only)
    - L1 Portability contains the atomic operations and memory management operations (both architecture and environment)
    - L6 App contains the base application kickstart (environment only)
- FileSystem
    - L1 Portability contains the code for sockets and filesystem access (plus other abstractions) (environment only)
    - L6 App contains the base application kickstart which depends on the filesystem (environment only)
- Scheduler
    - L1 Portability contains the scheduler code aside OS or architecture dependent primitives (both architecture and environment)

The porting process consists of writing platform code for the highlighted layers, caring for upper layer primitives' expectations. This is true and simple when dealing with a porting for an operating system but, when dealing with bare-metal platform, extra steps must be taken to ensure that the hardware is correctly initialised and ready before running MARTe2 on-top. The

work undertaken for this thesis work explored several platforms before landing on a suitable universal solution which required some modifications to the structure of the framework. Amongst them, entry points for the hardware initialisation before and after the whole MARTe2 framework initialisation where introduced. In the Ultrascale+ platform, this work was exacerbated by the presence of an extremely complex platform which needed initialisations at several stages of MARTe2, plus hooks to handle the network stack and UART output.

4.7 MARTe2 porting scaffolding

This procedure started from the existing MARTe2 porting code and guidelines to achieve a streamlined porting procedure, also with the help of a "empty porting stub" which was specifically made. Official MARTe2 porting before the start of the presented work were available for Linux (architecture x86_gcc, environment Linux) and Windows (architecture x86_cl, environment Windows).

4.7.1 The Makefile mechanism

As previously described, layer folders may contain one or both the Architecture and Environment subdirectories, with the relevant code kept into them. Architecture and Environment folders contain other folders, in turn which represents the specific platform (architecture) and operating system/environment (environment). A porting is represented by a complete couple of architecture-environment. This convention and naming scheme is done to encourage code reusage across the possible combinations and avoid the proliferation of (Architecture x Platform) combinations of mostly copy-paste code.

Bundled (official) MARTe2 architecture and environment directories must reside under the MARTe2 tree, before the rework of the Makefile mechanism. One of the modifications brought to the framework was the introduction of the support for the external custom porting placement. With this modification, "unofficial" or own work can be kept well clean outside the MARTe2 directory tree, with all the deriving advantages. This mechanism is obtained by leveraging the MARTe2 Makefile structure.

This mechanism is based on the directory MakeDefaults which contains two files (MakeStdLibDefs and MakeStdLibRules) for each couple of Architecture and Environment. Official portings have these two files placed under the MARTe2/MakeDefaults. The introduced new mechanism allows own or "unofficial" work to re-define the MakeDefault path with an

environment variable to suit for a specific directory placement. This mechanism also allows partial porting (Architecture or Environment only) to exist outside of the MARTe2 directory tree, to further encourage code base re-utilisation (e.g., reuse bundled architecture with user own environment or vice-versa).

4.7.2 ARCHITECTURE AND ENVIRONMENT UNBINDING

Another introduced concept was the Architecture/Environment unbinding. Some scenarios (notably the Zynq Ultrascale+ platform) need a customisation to the architecture or environment porting which contains elements bound to the platform itself, resulting in a genericity loss. For instance, the armv8-gcc porting types in the Ultrascale+ platform is bound to the platform itself and contains definitions which are clearly outside the scope of armv8-gcc universality. In these scenarios, the approach expects a modification in the generic porting which, using a directive defined in the *MakeDefaults* selects the right specialisation for the platform. This mechanism also offers a fallback to the default “clean” implementation.

4.7.3 TEST ENVIRONMENT PORTABILITY

MARTe2 quality assurance is based on an extensive test suite. This test suite is in turn based on the Google Test (GTest) unit testing library for the C++ programming language [41]. The GTest suite is not always portable to all architectures, especially on embedded systems. The need to have the ARM porting integrated into the MARTe2 QA and CI systems leaded to the choice to implement a small suite which could replicate the behaviour of the GTest suite and keep a 1:1 compatibility with the already implemented MARTe2 test codebase.

For these reasons a Portable Test Environment was implemented and integrated into the MARTe2 test toolchain. This test environment relies on the static linking of the library under test and exposes the relevant test macros in order to maintain full compatibility with the existing test suite. As the static linking of both the framework and test suite generates binaries which could be larger than the storage (especially true in MCU porting scenarios), a link filter was also implemented. This link filter can work in two modes, which essentially select the granularity of the selection (layer or library mode) and the inclusion or exclusion (whitelist or blacklist mode).

he GTest “emulation” mechanism is based on a suite of macros, which, when compiled expand the single test suite to a static binary which calls sequentially all the test cases. It also provides an output mechanism to generate the same output as GTest, for the continuous integration mechanism to produce the pass/fail report.

```
#define ASSERT_TRUE(x) \
this->testResult = x; \
printf("%s", \
(this->testResult)? \
" PASS\n":" FAIL!\n")


#define ASSERT_FALSE(x) \
this->testResult = !x; \
printf("%s", \
(this->testResult)? \
" PASS\n":" FAIL!\n")
```

```
#define TEST(x,y) \
class Tester_##x##_##y : \
public TestMarkerInterface {\
public: \
Tester_##x##_##y() : \
TestMarkerInterface(#x, #y) {} \
virtual void Test(); \
}; \

static \
Tester_##x##_##y \
Test_##x##_##y; \
```

The two snippets above show two macros that are “emulated” from the GTest suite (ASSERT_TRUE / ASSERT_FALSE) and how the macro-based expansion mechanism manages to generate the test suite from the sources, keeping the compatibility.

The moving from GTest to a custom Portable Test suite had a series of advantages, both applicable to the development of the porting itself and to the integration of it in the MARTe2 QA/CI/CD process. Moreover, keeping the same interface avoided a complete rewriting of all the test suite and allowed the inheritance of the actual code metrics, without impairing future releases or modifications to the sources.

4.7.4 START-UP HOOKS AND BOOTSTRAP PROCESS

The MARTe2 application (MARTeApp) already includes a main function which starts up the application, after the complete initialisation of the platform. Moreover, before the entry point, static initialisation of some MARTe2 vital structures takes place. Some platforms, especially embedded systems and microcontrollers, require hardware and peripheral initialisation to occur before everything else.

Three hooks where added, two occurring before the MARTe2 *GlobalObjectsDatabase* constructor begins its initialisation loop and one occurring inside the L6 Application bootstrap, which is the MARTe2 entry point.

These three hooks are:

- `InitHardware()`: called first
- `InitPlatform()`: called immediately after `InitHardware()`
- `MARTe2HardwareInitialise()`: called in the MARTe2 `main`

Note that the two early hooks (Hardware/Platform) follow the MARTe2 Architecture/Environment separation of concerns. This is done again to promote the code reuse and maintain the same philosophy across the whole framework.

The MARTe2 FreeRTOS porting on the Ultrascale+ platform raised another scenario which resulted in further modifications to the application start-up process. FreeRTOS on the Ultrascale+, when using networking, requires that some initialisation relative to the network physical layer occurs before the *GlobalObjectsDatabase* start-up. Conversely, the initialisation of the networking stack (lightweight IP – lwIP) must occur inside the FreeRTOS environment; that initialisation step must instead execute once and before the MARTe2 task is executed. Moreover, the network initialisation task must run another task which oversees the data transfers to and from the stack and manages the timers (timeout, TCP, DHCP). This twist, which requires the suspension of the main MARTe2 task until the networking initialisation one has finished, required a modification of the start-up process. The MARTe2 application, instead of implementing directly the main function entry point, calls a further entry point passing the desired code for the main function to be executed in the Bootstrap. In this way, the Bootstrap porting, which knows the specific environment needs, decides if calling directly the desired main function (as in the standard Linux or bare-metal) or to execute it in the context of a task (as in the FreeRTOS implementation). The FreeRTOS environment specific MARTe2 bootstrap calls a pre-loader function which suspends itself using the `vTaskSuspend()` primitive until the context inside the *MARTe2HardwareInitialise*. The *MARTe2HardwareInitialise* itself spawns another task which initialises the network and signals the resume to the pre-loader function. Finally, the resumed pre-loader can run the MARTe2 application and tasks devoted to initialisation can be safely deleted. This extremely intricated situation happens since a particular execution order of operations that must execute in the context of a task is required. The execution order cannot be guaranteed by FreeRTOS, which would have scheduled tasks and started its time slicing. However, the need to force the execution order led to the development of a general solution for the start-up initialisation and order of execution.

4.7.5 PORTING STUB

The porting activity landed to the implementation of a “porting stub” which is a collection of “empty” files which only contain a structure ready to be filled. The stub is also supported by a walkthrough with a per-class implementation hints, procedures and suggestion. The stub is available, together with the walkthrough and porting guide, as part of the MARTe2 documentation.

4.7.6 ACHIEVEMENTS

The described framework for the porting, the stub, the companion walkthrough and all the modifications made are now merged and part of the MARTe2 framework on the master (release) branch. The documentation work became part of the MARTe2 official documentation and is available for users to launch their own porting. The code, documentation, the stub and all the presented comments are available on the Fusion for Energy code repository both on GitLab and GitHub.

4.8 MARTE2 PORTING FOR THE BARE-METAL RASPBERRY PI

One of the first attempts to port MARTe2 was targeted to the Raspberry Pi bare-metal. The justification for this particular choice was related to the fact that the Raspberry Pi single-board PC offers a cost-effective solution with some expansion capabilities which suit the possibility to run MARTe2 in a practical control application. As a plus, foreseeing the Zynq Ultrascale+ platform porting in bare-metal, it was deemed a fundamental step to investigate porting opportunities beforehand, also considering that some Raspberry Pi models share the same cores of the Zynq Ultrascale+ (i.e., ARM Cortex A53 – ARMv8, which can be found in Pi3B, Pi3B+, Pi3A+, Pi02W).

Moreover, the bare-metal porting of MARTe2 on the Raspberry Pi was also considered for its QA implications in the maintenance on the CI of the ARMv8-gcc architecture. This idea was also enforced by the fact that a Raspberry Pi could be emulated using Quick EMUlator (QEMU) virtualisation technology and its image could be quickly deployed into the CI chain.

The Raspberry Pi porting is based on the Circle bare-metal programming environment, targeted for almost every Raspberry Pi (except for Pico). It is written in C++ and provides facilities which match the requirements of MARTe2, to access the filesystem and the network stack. The Raspberry Pi porting was also key to the development of the test framework, which

implementation posed first porting problems. However, the Raspberry Pi porting still is not deemed mature enough to be included in the MARTe2 master branch and to the current date resides in a dedicated branch (develop-circle), waiting for a further development cycle. Beside the achievement of having MARTe2 running bare-metal on the RPi, the extensive testing with the MARTe2 framework allowed the discovery of a bug in the circle-stdlib library, on which the bare-metal circle library relies. The problem was related to a memory move (memmove) primitive call in AARCH64 environment (Issue #22 Bad memmove behaviour). Fix for this bug was implemented and become part of the library with the version v44.1 and circle-stdlib v15.8 (commit 86094cf, “Implemented fix for issue #22”).

4.9 MARTe2 porting for STMicroelectronics STM32

An early attempt to port MARTe2 on microcontroller was developed, starting from an already existing codebase from the MARTe2 repository. Some modifications were made, essentially to demonstrate the capability of the MARTe2 platform when running on this class of devices, which already own plenty of interfacing options for the field. Modifications that were made served to suit the newer boilerplate HAL code produced by the STMicroelectronics Integrated Development Environment (STM32CubeIDE) with the existing implementation.

The MARTe2 porting for the STM32 platform was successfully tested on an STM32F746 platform (ARM Cortex M7 core) and two custom DataSources were developed, specifically one to access the ADC and the other the UART peripheral. Although the code is not deemed mature enough to be included in the MARTe2 master branch, the porting on this class of devices seems extremely promising for the field of real-time control applications.

This particular platform for the execution of MARTe2 can be considered as a turnkey solution even for complex real-time control applications, where MARTe2 is used in conjunction with the plethora of onboard peripherals available. The porting, which now is currently in standby, will return in development and brought to operation as there is growing interest in its potentialities and use cases.

4.10 MARTe2 porting for the AMD Xilinx Ultrascale+ platform

Another key development is represented by the porting of MARTe2 on the AMD Xilinx Zynq Ultrascale+ Multiprocessor SoC (MPSoC) hosted on the Trenz Electronics TE0808 System-on-

Module (SoM). The Ultrascale+ SoC is integral part of the bespoke electronics for the 55.A0 magnetics electronics and software, providing the interface between the magnetic diagnostic sensors and the CODAC I&C networks.

The development activity for the Ultrascale+ platform was not only limited to the porting of MARTe2 but also included the assembly of a complex real-time control application which spawned on three of the four ARM Cortex A53 cores onboard, the development of ancillary modules to support the inter-core communication and the data transfer from the FPGA to the CPU (Programmable Logic – PL to Processing System – PS) and an extensive test campaign to validate the performance requirements of the whole hardware/software ensemble.

Before retracing all the development steps and achievements, an introduction to the magnetics diagnostics and on the context will be given, to allow a deeper understanding of the architectural and implementation choices that were made. [42]

4.10.1 INTRODUCTION TO THE MAGNETIC DIAGNOSTICS IN GENERAL

Magnetic diagnostics is a passive diagnostic method which is used to measure the field and flux of the plasma, by means of magnetic probes. The magnetic diagnostic allows the assessment of macroscopic plasma characteristics and, under favourable conditions, the plasma parameters in detail and the MHD activity. These measurements allow the detection of several key plasma parameters, such as:

- Plasma current defines plasma equilibrium, is measured with multiple distributed solenoids called Rogowski coils, which surround the plasma column. The plasma current value is obtained by integrating the signal from the probe. The current can also be obtained by applying Maxwell-Ampère equation to a set of discrete pick-up coils. Independent measurements of the plasma current are key to the machine operation reliability.
- Plasma vertical speed for vertical stabilisation control.
- Loop voltage characterizes plasma contamination with impurities and is one of the parameters used to compute the average resistivity, together with plasma current and cross section.
- Position along the major radius, serves to avoid contact between plasma and plasma facing components.
- Shape of the outer magnetic surface.

- Plasma MHD perturbation parameters, amplitude, frequency, mode structure, correlation coupling by measuring field fluctuation characteristics.

Magnetic measurements involve some difficulties, especially in large machines. As the estimation of many parameters requires the integration of the sensor signal (to compute the flux/field from the measured voltage), when the discharge duration is long a value drifting is encountered, leading to a degradation of the integrated measurement. [2]

4.10.2 55.A0 MAGNETICS DIAGNOSTICS

The ITER magnetic diagnostic, 55.A0, provides measurements of the magnetic properties of the plasma, from raw parameters (local field and flux changes) through time-integrated quantities (field and fluxes) to complete equilibria and derived plasma properties (shape, position, speed, energy, slow and fast instabilities). To do this the diagnostic uses multiple sensor groups as subsystems:

- Set of pick-up coils, saddle and voltage loops mounted on the inner wall of the vacuum vessel.
- Rogowski coil mounted around earth straps of the blanket shield modules.
- Sets of pickup coils, Rogowski coils and shunts mounted on the divertor diagnostic cassettes.
- Sets of pick-up coils, loops and steady state sensors mounted on the outer surface of the vacuum vessel.
- Continuous poloidal (Rogowski) loops mounted within the TF coil case.

Signals from these sensors are conditioned, calibration factors are applied, and plasma properties are derived, in quasi-real time and offline using a dedicated set of processing I&C.

The basic end-to-end outline of the system is relatively simple. The 25 groups of sensors transmit their signals via wiring systems to a set of conditioning units that are placed as close to the machine as feasible. Another set of wiring transmits the signals to the ITER diagnostic hall where the final analog conditioning takes place. The signals are acquired by dedicated ADCs and distributed to software modules that reside within the CODAC system.

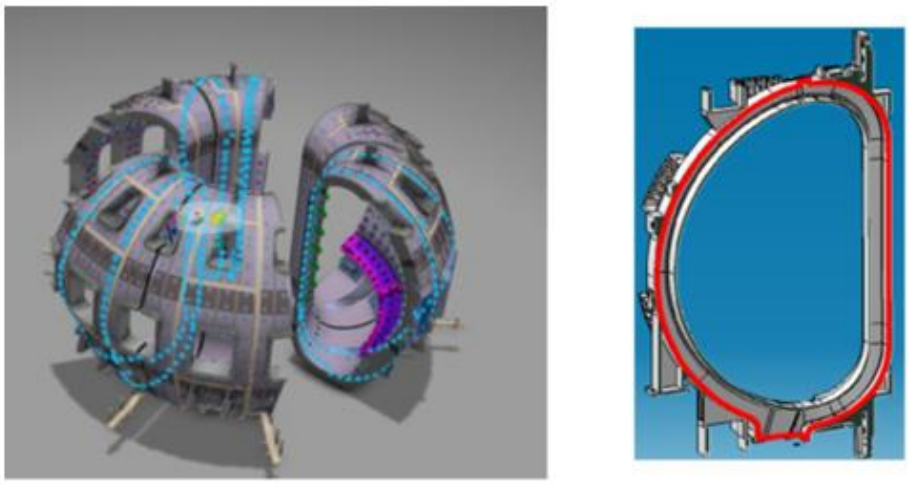


Figure 11 - General overview of the ITER Outer Vessel A3/A4 Magnetic Sensors and side view of the CER loop in the TF coil

The ITER plasma diagnostics system is required to provide accurate measurements of plasma behaviour and performance. There are three categories of parameters to be measured:

- Group 1a1 includes those measurements needed for machine protection
- Group 1a2 includes those measurements needed for basic machine control
- Group 2 includes those measurements required for evaluation and physics studies

The machine is unable to operate without a working diagnostic providing every Group 1a parameter. Group 1b includes those measurements required for advanced plasma control. For advanced operation, the machine is unable to operate without a working diagnostic providing every Group 1b parameter. The machine may operate without a Group 2 parameter diagnostic in operation.

4.10.3 Instrumentation and Control specifications

The magnetics diagnostics will provide proportional (generally flux variation) and integral (flux) magnetics measurements to the Plant System Controller. These signals will then be used to computer relevant plasma parameters, such as the plasma shape and position. The estimated parameters will be used:

- To control in real-time plasma parameters (e.g., position, shape, velocity, MHD instabilities).
- To protect the machine from operating outside its operating space (e.g., interlock control of maximum plasma current values).
- To study plasma physics.

The signal processing chain of the magnetics diagnostics is composed by three key layers:

- A **signal acquisition component**, responsible for acquiring the probe signal and digitising, called “bespoke electronics”.
- A **DSP section** which verifies the quality of the signal, performs the digital integration and routes data to the archiving network, to the interlock system and finally to the fast controllers that are responsible for executing scientific algorithms and marshalling validated data.

The main function of the 55.A0 magnetics electronics and software is to condition, validate and supply the magnetic sensors output to the real-time plasma control, to the CIS interlock and to the Data Archiving Network (DAN) for post-pulse physics data analysis.

As described in figures below, the main components of the anticipated systems are:

- An **integrator** board to acquire data from the sensors.
- A **data processing** board to aggregate and distribute data from many integrator boards.
- A **computing system** to compute the physics parameters and distribute the results to the above-described systems.

Considering the long plasma pulse duration foreseen for the ITER experiment, the integrators must exhibit extreme stability and low drift, for the reasons also described above. The large number of sensors (> 1600) and the fact that they are spatially scattered over the tokamak vessel, needs a highly distributed I&C architecture with many cubicles and components, interconnected with networks with latencies < 100 µs and bandwidth > 200 Gbps.

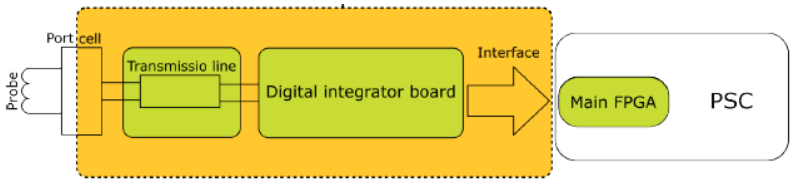


Figure 12 - Bespoke electronics component

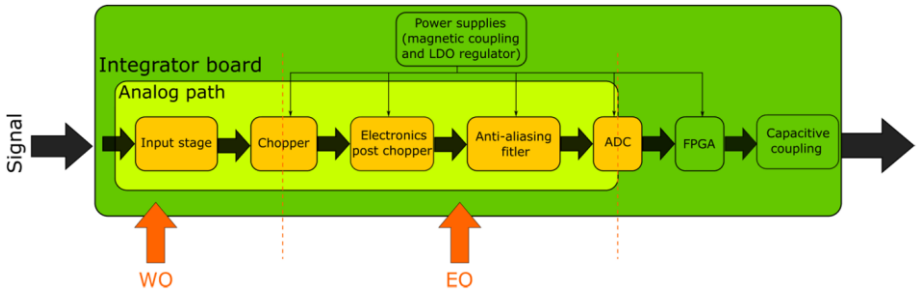


Figure 13 - Analog stages of the integrator board

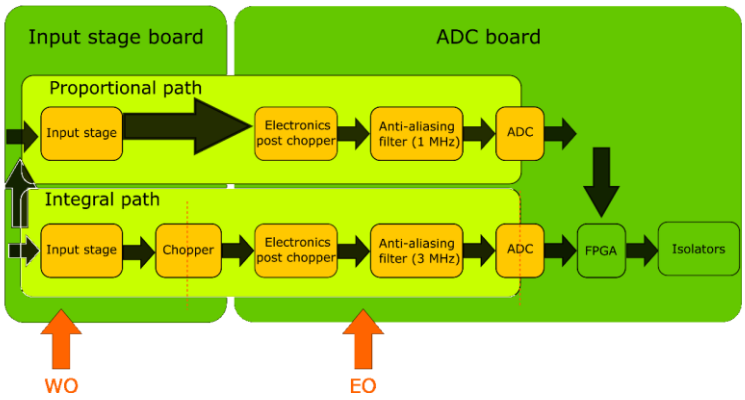


Figure 14 - Functional scheme of the digital integrator board

Referring to previous diagrams, describing the architecture and components of the bespoke electronic, a short description of each stage is given.

Input stage. Adapts the input voltage to electronic levels, it is kept as simple as possible in order to minimize the thermal voltages. This stage attenuates the larger input signals so they fit within the ADC input range. The integral channel includes a capacitor implementing a first order filter in order to remove the high-voltage high frequency signals.

Chopper. Square-wave modulation, only for the integral channel. This stage is the key component of the digital integrator and provides a conceptual barrier between the input stage and the following electronics. The modulation stage consists of an analogue chopper which inverts the signal periodically with a 50% duty cycle. Being a square modulation, the signal is spread around the chopper frequency.

Electronics post-chopper. Very high input impedance segregates the input stage from the antialiasing filter, avoiding overloading the input stage. This stage is a simple buffer in a 1 gain configuration. It increases the impedance seen by the chopper aiming at reducing any current to a minimum and avoiding the asymmetric voltages that these currents might generate.

Anti-aliasing filter. Operational amplifier with a second order low-pass filter. It is implemented as multi-feedback second order filter with a fully differential operational amplifier. The response type is Bessel, to avoid over voltages on the chopper transients. The cut-off frequency depends on the channel:

- **Proportional channel** has a cut off frequency of 1 MHz. This value allows to measure the 500 kHz of the AJ coils and, at the same time, it attenuates the frequency above the Shannon limit. In addition to the anti-aliasing filter, the PCB is placed inside an aluminium box and the cables are shielded and twisted.

- **Integral channel** has a cut-off frequency of 3 MHz, to allow for a better demodulation. A lower cut-off frequency could generate an unwanted DC voltage if the input frequency is twice the chopper frequency.

ADC. Analog to digital converter, running at a maximum 2 Msps with a 22 bits resolution, aiming to obtain the maximum possible effective number of bits (ENOB).

All the electronics are isolated: the communication of the FPGA with the outside world is done via capacitive coupling, while the power supplies are isolated via push-pull converter with magnetic coupling. The full isolation of the electronics avoids ground loops and outside noise.

The main goal of the integral path is to keep the offset as low as possible and its subsequent drift after integration in order to meet the 500 µV*s in 3600 s drift requirement. [43] [44] [45]

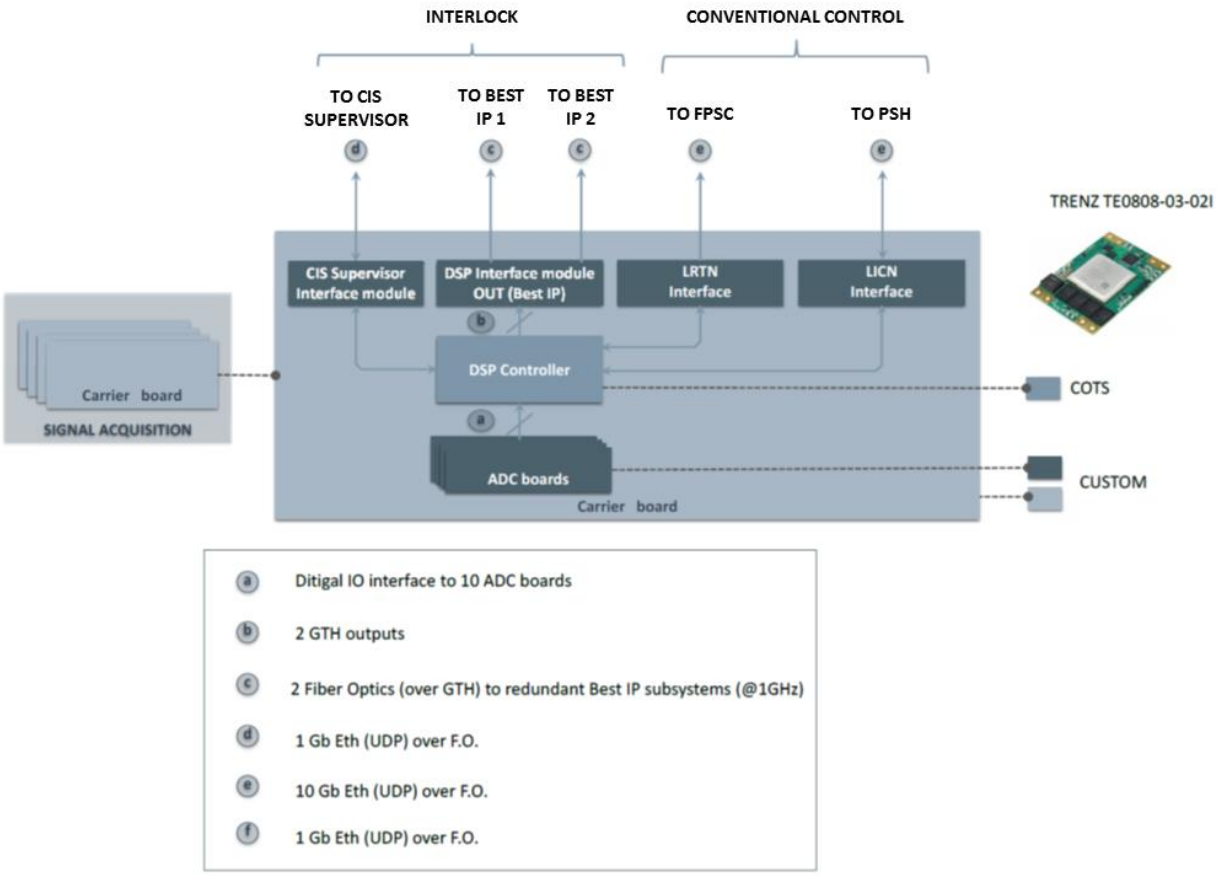


Figure 15 - High level architecture of the signal acquisition subsystem

4.10.4 AREA OF INTEREST

Two components of the 55.A0 magnetics diagnostics are interested in the MARTe2 porting activity. One is relative to the Main FPGA aggregator board and the other to the Best Ip board.

The Main FPGA board carries on the signal processing for all measures. It first performs an Electronic Offset (EO) compensation with a calibration parameter received from the configuration functions. After EO compensation the system demodulates the signal that has been inverted by the chopper. In parallel, the system performs the integration of the Wiring Offset (WO) calibration parameter received from the configuration functions. Then the signal is integrated and corrected with the results of the WO integration. After integration the signal is converted to physical units and decimated (filtered and down sampled) to 2 kHz. Each board integrates up to 30 ADC signals.

Processed sensor data will then be sent to the functions for the computation of the plasma parameters for interlock. The functions to compute the plasma current for interlock will perform the following activities:

- Select the sensors to be used for the computation of a plasma current.
- Perform plasma current validation.
- Compute Best Ip with all the valid plasma currents.
- Send the Best Ip to CIS PPM.
- Send the individual Ip to CIS supervisor.

In simple words, the Main FPGA board provides the interface between the integrator sensors and all the CODAC I&C networks.

The Best Ip board, instead, implements the interlock function of computing the best plasma current with all the valid plasma currents. The best plasma current is sent to CIS PPM and other individual plasma currents are sent to CIS supervisor. The computation of the plasma current for interlock involves three sub functions:

- Select sensor data from Ip.
- Compute 6 / 12 Ip. The function computes the Ip as a linear combination of sensors.
- Compute Best Ip averaging all the valid plasma currents.

The quality parameter for the Ip is not computed in real-time but provided through configuration.

Both the boards run on an AMD Xilinx Zynq Ultrascale+ MPSoC and TE0808 SoM and also using the same carrier board.

The porting activity was aimed at having a stable software platform able to run on the hardware modules to accomplish the signal processing and data streaming tasks required for the two applications. The power of delegating the solution to MARTe2 applications allows to go beyond the classical vertical firmware development approach, where the effort is concentrated for the specific solution with only a small (or non-existent) portion of reusable software components produced. With the MARTe2 approach, the effort goes solely into the direction of implementing correctly the porting, by meeting the framework expectation. The effort has twofold advantage: once MARTe2 is ported, all the robustness and dependability characteristics of the framework (QA, MISRA C++:2008, test suites) are ported and, with the fully fledged framework ported, all the existing (and tested) MARTe2 components bundle becomes available. Moreover, as the framework is maintained, new features and fixes become available, these will be also readily available also for the newly integrated platforms.

4.11 MARTe2 BARE-METAL PORTING

The activity of porting MARTe2 on the Zynq Ultrascale+ platform followed the process which was previously defined. The process, as aforementioned, requires the implementation of the code in architecture and environment portions of the framework outer layers. As some of the processes were already mentioned in the previously dedicated paragraph, only the relevant and peculiar implementations will be further analysed. The porting target is defined as a combination of arm_gcc architecture and BareUS environment.

4.11.1 L0 TYPES

The L0 types, which contains the mapping between MARTe2 and hardware types was ported using the Xilinx provided type-wrapper header. The usage of the manufacturer's provided header for supported types, instead of the standard one (stddef/stdint), provides a homogeneous implementation and application across the whole Xilinx supported processors family.

During the porting, the possibility to support (explicitly) also the NEON instruction set and, consequently, the needed NEON types was also explored. However, it was not deemed necessary, nor convenient as current algorithms are not suitable to support those types. Moreover, the explicit support or reference to hardware-related aspects in GAMs is strictly prohibited in the MARTe2 framework.

Xilinx provided types are all in the form [u - unsigned, s – signed] [8, 16, 32, 64] bits plus float, double and char (e.g., s32 for the C/C++ standard signed int type).

4.11.2 BAREMETAL / L1 PORTABILITY / ARCHITECTURE

The porting of this layer relies on ARM gcc built-in functions for memory model aware atomic operations. These functions are all prefixed with *__atomic*. These built-in functions require a memory order to be specified, that was defined using the less relaxed *__ATOMIC_SEQ_CST* [46], which enforces total ordering with all atomic operations. The choice to implement the most restrictive was done to favour safety over efficiency which would derive by the usage of a more relaxed requirement.

Endianness conversion functions were delegated to the intrinsic *__builtin_bswap*[16, 32, 64] bits functions [47].

4.11.3 BAREMETAL / L1 PORTABILITY / ENVIRONMENT

The MARTe2 high resolution timer abstraction leans above the Xilinx XTime library, which abstracts the access to the Cortex A53 MP core timer. The library is a simple wrapper to the CPU register read but takes care of eventual timer start-up and exposes convenient constants which define the tick frequency. In the Cortex A53 for the considered setup, the timer ticks at 33.333.332 ticks per second (33,33 MHz, thus giving the microsecond accuracy needed for the framework).

4.11.4 BAREMETAL / L6 BOOTSTRAP / ENVIRONMENT

Hook functions are implemented to configure the memory management unit (MMU), instruction/data caches and execute low level initialisation for the clock distributor, which enables the network interfaces on the board.

4.11.5 FILESYSTEM / L1 PORTABILITY

The realised porting supports FatFs for the long-term storage, even though the production carrier board will have no direct usage. The FatFs can be used in SD Card mode or in RAM filesystem mode, the porting enables the RAM FS mode although it is never used.

Networking stack was ported on lightweight IP (lwIP) stack in raw API mode. The raw API mode expects a specific stack call in the main loop function, that is not available in MARTe2, as the whole loop is internal and not exposed. To overcome this limitation, the network stack call was implemented directly inside the socket read and write functions via a specific hook function. This approach will allow the porting on similar platforms, which require an explicit

call to move data to/from the network interface inner ring-buffers to the user space. A select-like behaviour was implemented, as receive function in raw API are callback based, instead of sequential. To implement the select behaviour a global singleton was created, where all created sockets are registered, with their respective handling callback. Once the packet is arrived, it is saved in the singleton structure aside ancillary data which help determine if the socket was selected and if a packet has arrived since then. In fact, while send functions are straightforward, with respect to MARTe2 expectations, the callback-based behaviour on receive is not directly suitable for the framework. The implementation uses the callback to fill a packet buffer and meets MARTe2 expectations in read function call, by returning the previously filled buffer.

Another peculiar implementation is related to the multicast group join request for the BestIp board. In the standard lwIP foreseen implementation, the join request must be implemented in the main loop aside the network stack call. As this cannot be achieved, even with the usage of the hook, the join request enqueues its intention into a queue, which is progressively emptied in the network interface hook, when it is called, avoiding the stall situation which is created with the direct call.

4.11.6 GAM BARE-METAL SCHEDULER (*GAMBARESCHEDULER*)

To suit the needs of the bare-metal porting, a very simple scheduler was added to allow calling in an infinite loop all the GAMs that are declared in the MARTe2 real-time thread. The bare scheduler is not part of the porting itself but became directly part of the framework, as a drop-in replacement for the standard GAM scheduler, which needs instead a more complex architecture to allocate thread, priority and CPU and other features that are not available on bare-metal.

4.12 MARTE2 FREERTOS PORTING

Again, the porting procedure followed rules described before, also for the FreeRTOS environment. This porting is backed by a real-time operating system. The real-time operating system adds to the MARTe2 application the multitasking ability, which is instead unavailable in the bare-metal porting.

The bare-metal Architecture sub-layer code base is reused (arm_gcc) entirely, as mandated by the MARTe2 coding conventions. Changes are encountered in the Environment sub-layer,

where the framework relevant parts are laid upon the FreeRTOS real-time operating system. To avoid repeated descriptions, only the substantially different parts will be further described.

4.12.1 BAREMETAL / L1 PORTABILITY / ENVIRONMENT

MARTe2 differentiates between the concept of busy sleeping and operating system sleeping. The first implementation is based on a spinlock over the evolution of the timer counter value and its implementation resides inside the framework, the latter instead relies directly on the operating system *sleep* functions, which are rarely implemented as spinlock. OS sleep functions, instead, rely on the scheduler perception of time advancement. The sleep timespan can be strictly observed (as in real-time operating systems) or specified as *minimum required* time to sleep (as in best-effort / general purpose operating systems). The result is that the CPU is freed for the execution of another task until the time expires and the sleeping task can be scheduled again. In FreeRTOS the sleep function is implemented using the `vTaskDelay()` function, which requires a timespan expressed in scheduler ticks. In FreeRTOS, the scheduler tick rate is 100 Hz, meaning that the minimum allowed, OS-based sleep function is 0.01 s. As a side effect, the `vTaskDelay()` function call causes the task yielding (control released to the scheduler). In the MARTe2 porting, this aspect is taken into account as, when the sleep function calculates a number of ticks equals to zero using the ms-to-ticks conversion macro, the `vTaskYIELD()` is called, voluntarily releasing the control again to the scheduler.

The standard heap functions were laid upon the `pvPortMalloc()` primitive. Using the `pvPortMalloc()` explicitly means relying on FreeRTOS internal memory management, independently from the selected heap management algorithm. The dual function, `vPortFree()` is also mapped for the opposite functionality. The FreeRTOS heap is pre-allocated and allocation/deallocation primitives manage the access to the area (e.g., fragmentation, allotment, ...).

4.12.2 BAREMETAL / L6 APP /ENVIRONMENT

Referring to the MARTe2 boot process previously described, the loader function is passed as task argument, before calling the FreeRTOS scheduler start function. The pre-loader suspends itself, to give the hardware initialisation task the opportunity to complete its execution before calling the concrete Bootstrap main function. The hardware initialisation, on its side, must call the resume towards the pre-loader task before finishing, in order to allow the framework execution to proceed.

4.12.3 FileSystem / L1 Portability / Environment

While the FatFs layer is essentially the same, main differences can be found in the lwIP stack. It supports both the raw and socket APIs. The raw is implemented as already described for the bare-metal version. Socket API interface instead reflects almost identically the *UNIX socket interface.

4.12.4 Scheduler / L1 Portability / Environment

Thread, Event semaphores and Mutex semaphores are directly mapped to FreeRTOS equivalent functions. As FreeRTOS set of primitives to query the number of tasks and get related handles is system-wide scoped (returns all the tasks instead of ones related to MARTe2), the porting recurred to the usage of the MARTe2 internal *ThreadDatabase* to keep the inventory of MARTe2 owned tasks. This choice was also pursued to preserve test cases which relied on the thread count to pass.

4.13 MARTe2 integration challenges

One of the key concepts pursued during the porting activity was to keep the board-specific implementation outside MARTe2. This requirement was achieved, as already shown, with the usage of hooks to abstract functionalities at key execution points of the framework, like the initialisation, execution, network transfers. This separation allows the usage of the common IDEs to build the platform environment (BSP, HAL) boilerplate code, thus reducing the user intervention to simple linking of the MARTe2 Application binary.

However, some hardware related aspects must be kept outside the platform, because their code is extremely specific, not only in term of execution platform itself but also bound to the local configuration (e.g., the initialisation of an onboard peripheral needed to enable the networking subsystem, the interrupt controller configuration).

In the specific *MainFPGA* and *Best Ip* need a precise order of initialisation in the main function for the platform, which takes place in the various hook function. These boards require:

- MMU initialisation, in order to mark the correct portions of RAM as core-private or shareable (i.e., the shared memory DataSource requirement, discussed later). The MMU needs a setup of the translation lookaside buffer (TLB) which is carried out using a primitive function exposed from the board support package (BSP). The Double Data Rate RAM (DDR) is split equally between the three cores, leaving a small segment at

the end of the first bank to enable data sharing between cores (inter-core communication, data payloads), the TLB is configured to have it as an inner shareable portion. The On-Chip Memory (OCM) is also used for data sharing (inter-core communication, signalling) and is also marked as an inner shareable portion. Caching on the shareable OCM portion is disabled.

- I-Cache and D-Cache setup and enable, in the hardware initialisation. These two calls enable the instruction cache and the data cache.
- Generic Interrupt Controller (SCU-GIC) which is needed to handle interrupts and exceptions, a fundamental requirement for the network interface card and for the RTOS (ticks). The GIC needs to be configured in order to route network interface interrupt request (IRQ) to the right core. As each core uses independently a network interface, the GIC must be initialised in every core. The interrupt distributor instead must be configured only once, in the first core (core #0) and each core must configure the distributor to route interrupts correctly. The BSP has a special compilation symbol (USE_AMP), which disables the distributor initialisation in the other cores, while keeping the same code structure to setup and enable the GIC.
- Network physical layer (PHY) initialisation, by means of a Si5345 (10 output programmable PLL clock multiplier + NVM OTP). The Si5345 is configured by using a custom interface library which was implemented. This library communicates with the Si5345 using the I2C bus. The library initialisation calls must be executed before the network interface and lwIP stack initialisations.
- The *MainFPGA* board also requires the setup of the exchange system between the FPGA and CPU (Programmable Logic to Processing System – PL to PS), which is carried out using a DMA ring buffer. The specific mechanism and the linked DataSource implementation will be discussed in the dedicated chapter. To test and assess the whole chain, IP logic in the PL is also able to simulate the signal coming from the integrators with variable signals (constant, ramp, ramp chopper, waveform modulated). This also required the implementation of additional hooks, which are used in the dedicated DataSource.

4.14 MARTe2 on the MainFPGA board

4.14.1 Introduction

The *MainFPGA* board is part of the ITER 55.A0 Magnetics Diagnostic project. A *MainFPGA* board is equipped with the AMD Xilinx Ultrascale+ MPSoC on its carrier board and connected to 30 integrator boards via ADCs. Data produced by the ADCs is consumed by two MARTe2 bare-metal applications while a third MARTe2 FreeRTOS application is used to generate a telemetry streaming and serves as control for the other two instances. Two of the three applications (one bare-metal and one with FreeRTOS) have their own private network interface. Some specific DataSources allow the data exchange between the FPGA (PL) and CPU (PS) and between the cores of the CPU. This chapter will introduce first the whole application to step into the specific components that were implemented. At the end an analysis of the performances will also be presented.

4.14.2 MainFPGA MARTe2 application

MainFPGA application runs on three of the four Cortex A53 cores. First (Core 0) and third (Core 2) run bare-metal MARTe2 porting, while the second core runs the FreeRTOS MARTe2 porting. Schematically:

- Core 0 transfers the ADC data with some additional diagnostic information from the PL, using a dedicated DataSource (*PLPSDataSource*) that was implemented.
- Core 0 has not enough computation power to handle PL-PS transfer, computation and network streaming, so 19 out the 30 ADC channel data is moved to Core 2 where another bare-metal MARTe2 application is running.
- Core 2 reads data coming from Core 0, executes the computation algorithm and returns processed data to Core 0.
- Core 0 again merges own (11) channels with Core 2 processed (19) and streams them over ITER Synchronous Data Network via two streams, one at 10 kHz and the other at 2 kHz.
- Core 1, which runs the FreeRTOS porting of MARTe2, intercepts data transfers between cores and samples data at 10 Hz from the processed data chunk.
- Core 1 also runs a system monitoring DataSource, which was specifically designed and implemented, to decorate the whole telemetry data produced with board diagnostic data. The so produced data stream is sent using the dedicated network interface to feed the ITER S/T network.

4.14.3 PL TO PS DATA EXCHANGE

The *PLPSDataSource* is a DataSource which synchronises on the DMA Ring Buffer. The PL (FPGA) populates the ring buffer with ADC samples data. The *PLPSDataSource* works on a shared memory space which contains blocks of a struct. Each block contains a counter which is incremented by the PL on each written sample. The DataSource, on its side, synchronises on this counter and expects an incremented counter on the next synchronisation point.

The MARTe2 DataSource is implemented following the separation of concerns philosophy: it is totally agnostic on the DMA aspects of the memory buffer. However, two hooks are provided to the DataSource, to obtain:

- The base memory address of the data buffer, which in the specific is the DMA base address.
- The maximum number of packets, together with the packet size, allow the pointer arithmetic to follow the ring buffer filling.

A third hook, which is specifical to this implementation, is used to enable the FPGA IP which provide simulated signals instead of integrator fed data.

These hooks are implemented as external functions in the DataSource and resolved at linking time as part of the BSP/HAL package. Hooks rely on Xilinx HAL for the DMA ring buffer management. The initialisation code for the DMA ring buffer takes part in the MARTe2 platform initialisation hook, as previously described also for other scenarios.

4.14.4 INTER-CORE COMMUNICATION

The communication between the three cores where the MARTe2 *MainFPGA* application runs is achieved using a DataSource: *RealTimeThreadCoreSynchDataSource*. This DataSource allows to synchronise applications running in different CPU cores by using a sophisticated shared memory mechanism. A GAM, designed as writer, writes data on this DataSource, one or more GAMs running on different MARTe2 applications (in the specific also on different cores) will read and synchronise against their independent instance of this DataSource. This DataSource has a configurable mechanism which allows writers to lock on readers on request and a mechanism to allow the (aliased) down sampling of data, to allow readers at a lower frequency to be able to read data.

The communication is backed by a shared memory library which uses both the on-chip memory (OCM) and the DDR memory (RAM). DDR contains the payload data (MARTe2 signals) that

is shared amongst cores, OCM contains ancillary data which are used to arbitrate the data exchange. The payload data instead is based on a dual-buffer, zero-copy mechanism: writers and readers work on two distinct payload areas that are swapped (only pointers) when the write is completed. It is also based on a writer-first approach, where the data produced is deemed more critical than the consumer, however this behaviour can be configured, in order also for the writer to wait for a defined number of readers to complete. The OCM is a 256 kB memory (4 x 64 banks) spatially near to the CPU with very high throughput support located on AXI interconnect bus with ECC support.

4.14.5 ANCILLARY DATA

Ancillary data, which is located on the OCM memory contains:

- Pointer to the pointer (double star - **) to the write buffer
- Pointer to the pointer (double star - **) to the read buffer
- Pointer to the writer heart-beat signal location
- Pointer to the mutex flag
- Pointer to the reader mask
- Pointer to the announcement mask

The first two (read and write) are used to reference the two buffers, they are double-referenced to allow the zero-copy swap.

Each write causes an update (increment) of the heartbeat, that is a 32-bit counter. The heartbeat is used to allow readers understand if the writer has already been started. It is used on the very first read attempt of a reader to read from the shared memory, which otherwise will fail its initial read cycles until a writer is seen. The heartbeat mechanism was implemented as cores are not aware of their reciprocal execution status, as start-up procedures require different periods for the MARTe2 application to start.

The mutex flag is an 8-bit unsigned int that is used in conjunction with the atomic test-and-set intrinsic to guard the access to the shared ancillaries. The atomic test-and-set (*__atomic_test_and_set*) intrinsic is wrapped inside a convenient Mutex class. This portion of the code does not rely on MARTe2 layers, which could provide similar functionality, in order to have it available as independent software module. This approach also come handy during the development and testing. Although it is configurable (only at compile time, via a defined constant) the memory model used for the mutex are

- Acquire for the lock (__ATOMIC_ACQUIRE) [46]: barriers to hoisting of code and synchronizes with release (or stronger) semantic stores from another thread.
- Release for the unlock (__ATOMIC_RELEASE) [46]: barriers to sinking of code and synchronizes with acquire (or stronger) semantic loads from another thread.

The test-and-set (TSL) call, which is implemented in a loop, is interleaved with an ARMv8 assembly *YIELD* instruction. The instruction belongs to the special subgroup of *NOP* instructions and is simply put to avoid extreme TSL pressure on the mutex flag. Two other companions are implemented, to enlighten the polled mutex flag, that are the time between two consecutive calls and the pause after the mutex release. There is also, optionally available a Test-And-Test-And-Set-Lock (TTSL) approach [48] available in the source code, as a result of the intensive test and performance assessment campaign that undertook the component. As no improvement was detected with the TTSL over TSL, it was disabled and cannot be enabled by configuration options.

The announcement mask is used to inform the writer that a reader is present, in order to become able to wait correctly. Moreover, the announcement mask is used to avoid double announcement of the same reader with respect to a writer. A reader identifier is in fact configurable for this purpose. The announcement mask shall be seen as a bit mask, where each reader announces (sets to 1) the bit identified by its value.

The reader mechanism, with a similar bit-flag based mechanism, is used both by the reader and writer. The writer uses (optionally, when configured) to wait until all readers have consumed the last produced sample while readers, synchronising on the falling edge (1 to 0) of their respective bit, know if a fresh sample has been produced.

4.14.6 PAYLOAD DATA

The payload data is simply a double buffer where the writers and readers work on their half. The two buffers' pointers are exchanged as long a write operation occurs, thus avoiding multiple copies of data between cores. The output DataSource (writer) itself is structured to work directly on the writer half, while the input DataSource (reader) must copy on its own side the otherwise destroyed or corrupted data. This is mainly done because, while write operation usually occurs at the last stage of the data manipulation chain, the data read is instead at first stage. This means that read data samples must be manipulated, which is most probably a time-consuming operation that could not be in synch with the writer operations happening on the other core.

4.14.7 INTER-PROCESSOR INTERRUPT

While the polled approach is used in the *MainFPGA* MARTe2 applications, the DataSource internals allow also for an inter-processor interrupt (IPI) based version, where the exchange is based on the direct event notification using the interrupts issued between cores. The IPI mechanism allows for a core to send interrupts to another core (or more than one, via a mask). The interrupt carries a message which size can be up to 8 words (32 bytes). The DataSource uses the IPI message to signal other cores data is ready to be consumed, using the IPI message to carry a message code, a sequence number and 30 bytes of payload.

The IPI mechanism can be enabled at runtime, via configuration parameters only after the related compilation flag has been enabled to include it. *MainFPGA* MARTe2 applications do not rely on the IPI mechanism for the inter-core communication signalling. [49]

4.14.8 BOARD PARAMETERS MONITORING

Board parameters are monitored using another implemented DataSource, which relies on internal SYSMON peripheral to retrieve its vitals. Parameters that are retrieved include voltages, temperatures, clocks and are provided by the onboard peripheral which is queried using AMD Xilinx BSP primitives. This DataSource is extremely vertical, as its provided data is thought to be binary compatible with the payload of network streamed diagnostic data. Its internals leverage a sequencer, which is programmed at the beginning of the DataSource execution cycle. The sequencer is instructed on the kind of variables to read, the sampling frequency and average filtering rules. The DataSource is synchronised, it waits for a fresh sample to become ready on the system monitoring PSU by polling the peripheral interrupt. Once data is ready, using the provided conversion function, data is converted and MARTe2 DataSource output signal bank is populated with the samples. [50]

4.14.9 SIGNAL PROCESSING

Core 0 and 2 carry out the ADC data processing, following the data flow previously described. The processing task is split between the two cores and merged back before being streamed on the ITER SDN network. Splitting occurs since core 0 alone has no sufficient computational resources to copy from PL, run the signal processing algorithm and stream data on the network. The signal processing algorithm, that was developed by the Fusion for Energy team and is not part of this thesis, was only integrated with small modifications in the previously described and implemented signal processing chain that is instead part of the thesis work.

On the marginal modifications that were made to the PS algorithm, the most important was the conversion of the underlying socket from the advanced buffered to the simple unbuffered, to better suit the lwIP ported stack primitives. With the same philosophy, the Internet Group Management Protocol (IGMP) multicast join mechanism was also suited to the MARTe2 ported, based on the join mechanism previously described.

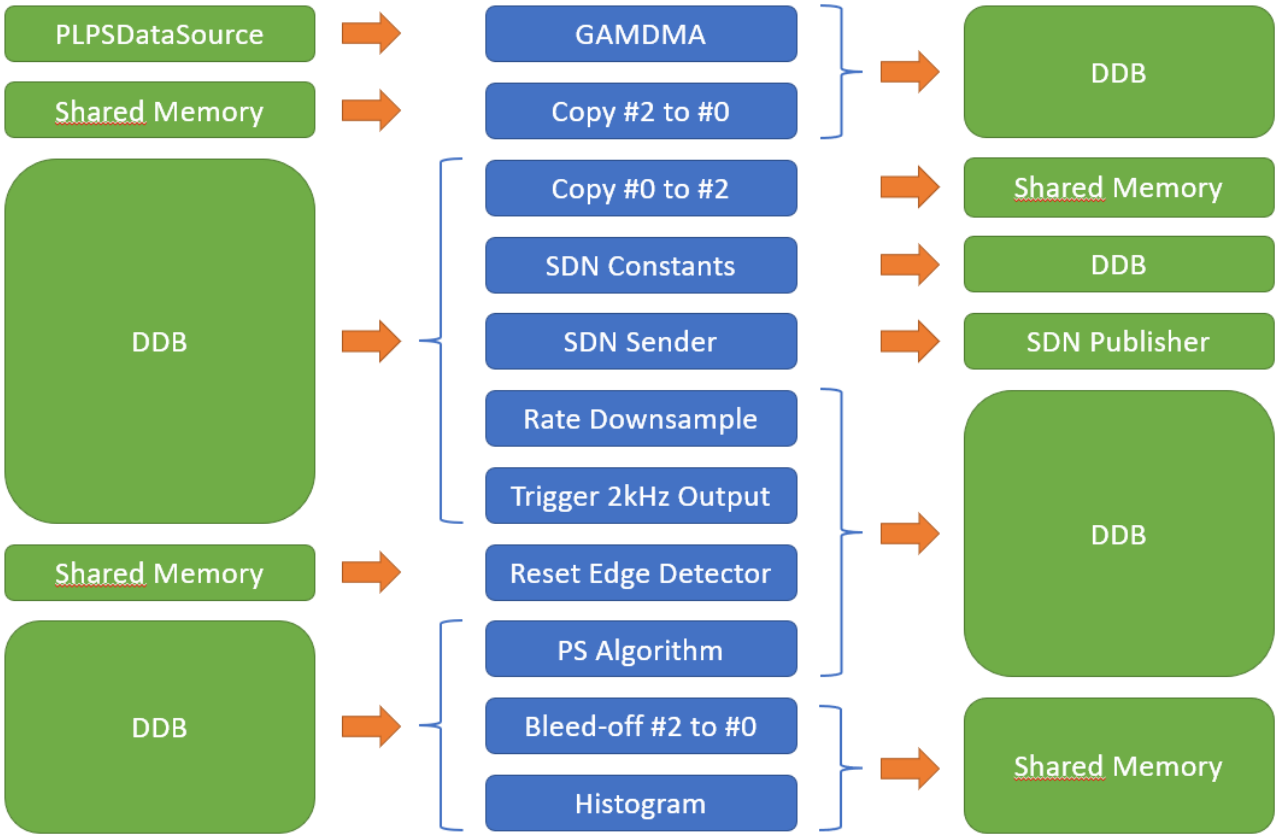


Figure 16 - MainFPGA Application running on core #0

Core 0 simplified application layout is shown above, on the left and on the right in green the DataSources while in the middle the GAMs. GAMs run in the context of the MARTe2 real-time thread with the specified order (top to bottom). All the components indicated were already in-depth analysed in previous chapters,

- *PLPSDataSource* is the component used to move ADC data from the PL (FPGA) to the PS (CPU). The *GAMDMA* function (*IOGAM*) is a simple copy of the data in the *DDB*, which can be considered as a temporary heap to store signals and move them across functions and DataSources.
- The Copy #2 to #0 algorithm and the Copy #0 to #2 move data between cores using the shared memory component. Their order is strategic, it allows to effectively parallelise the PS algorithm execution leaning on the simultaneous execution of the PS algorithm

on core 0 and 2. Intuitively it would have been placed at the end in a “merge” step, but doing so, all the time would have been spent in the whole loop (including the SDN send time).

- PS Algorithm is the ADC data processing algorithm, working on 11 channels.
- Bleed-off indicates the low-frequency data sent to Core #1 for telemetry and diagnostics, which is produced at 10 Hz.
- Reset edge detector is used to find edges in the integrator reset signal, which is coming from the telemetry core (#1).
- Histogram runs statistic data on the cycle execution time.

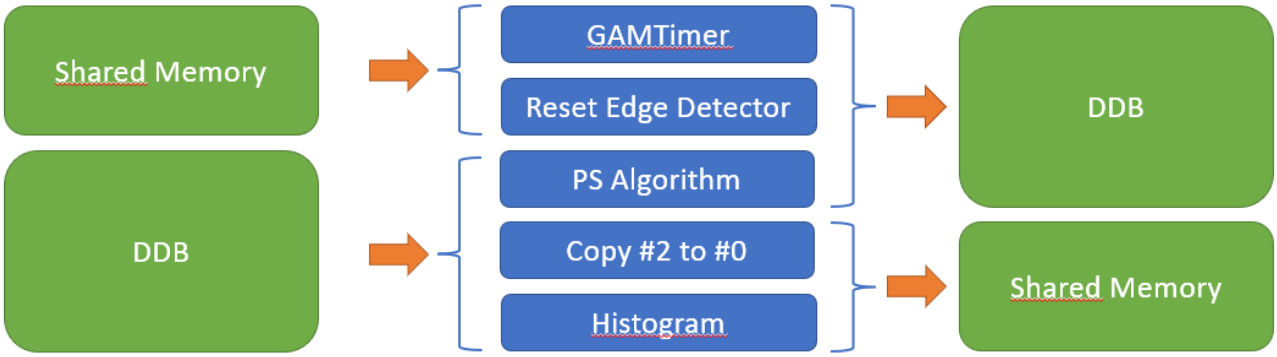


Figure 17 - MainFPGA Application running on core #2

As shown above, MARTe2 application on core 2 simply copies data, coming from core 0 and executes the PS algorithm plus statistics on the execution time data (histogram). Again, shared memory is used to move data back and forth cores.

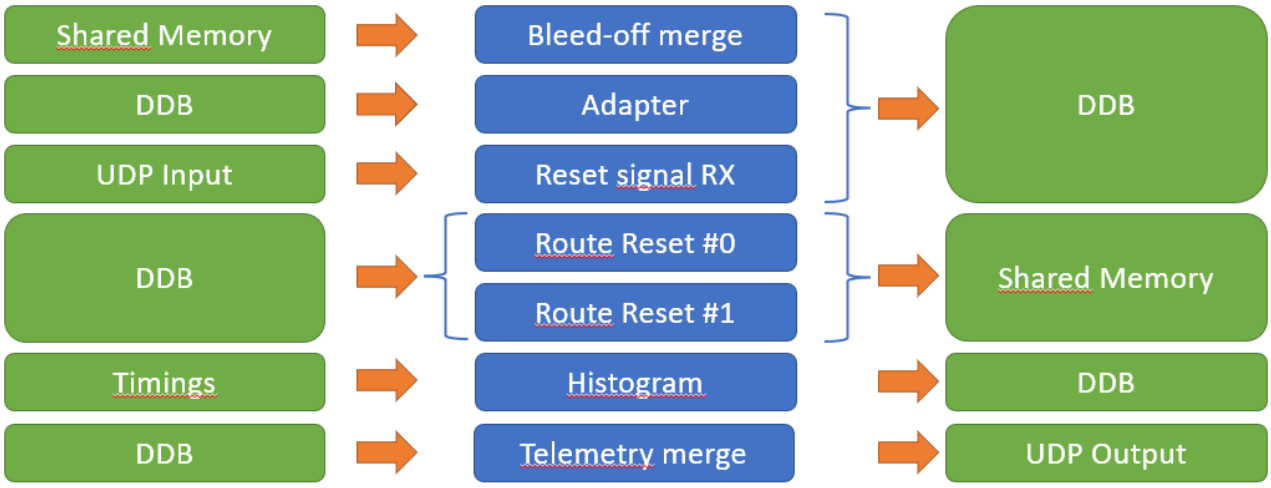


Figure 18 - MainFPGA Application running on core #1

Previous diagram shows how the bleed-off data is merged with the telemetry data and diagnostic from the board system monitoring to produce the UDP output stream. Conversely,

the reset signals for the integrators are routed, using the shared memory, to cores 0 and 1 PS algorithm. The highlighted *Timings* DataSource is a built-in MARTe2 component which provides execution times at every GAM stage. When Timings is enabled, read and write times for each function can be tapped, as a precious instrument to assess and profile execution performances.

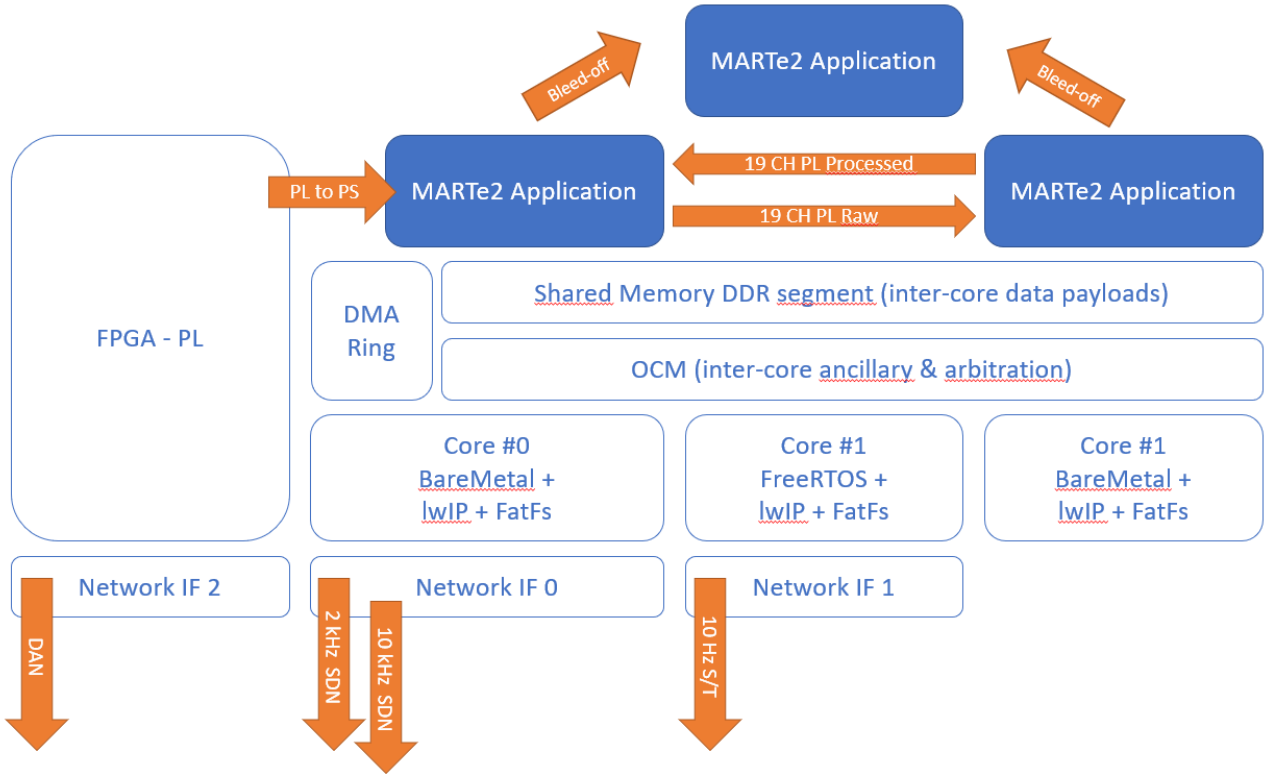


Figure 19 - Simplified interconnection overview

The diagram above shows a simplified view of the PL/PS and between cores interaction, briefly resuming previous paragraph explanations.

4.14.10 Core start-up sequence

The application requires a specific start-up order for cores, to ensure that the shared memory segments, data structures and dependencies (producer-consumer) are fulfilled. For the purpose, a Startup Manager library was developed. This library can formalize the before / after start-up ordering of cores. It must be instantiated and configured in the hardware initialization hook, before the MARTe2 framework. Its configuration requires three parameters: own core, previous core, next core. For each involved core, the library:

- Waits on the specific OCM area to see the previous (if any) core heartbeat signal (that is a switching between two values).
- Once locked on the previous core heartbeat, the core starts its own heartbeat and waits to see the next core alive (if any).
- Once the next core is seen alive, the core starts its own operations.

The library exposes two methods, one to wait and the other to signal the startup has occurred. This mechanism allows inter-locked core start-up, verifying the correct dependency between them. Start-up management is based again on OCM, using a small portion at the end of the memory bank. It is independent from MARTe2 and must be considered as a part of the BSP/HAL. By design, MARTe2 shall remain unaware of these specific details.

4.14.11 TRIGGER MODULO GAM

A very small component (GAM) for MARTe2 was developed, to suit the need to have a reduction ratio mechanism for the sampling frequency, in the transfer of signals. The GAM has one input (the signal generating the trigger) and an output (the trigger signal itself) and works on an internal counter which is incremented on each algorithm execution. When the algorithm execution cycle matches the configured reduction ratio, the trigger signal is raised for one cycle.

4.14.12 OTHER DEVELOPMENT

A simple system to store board specific configuration data was implemented, although it is not part of the production environment nor of the official branches. It was developed to prepare the ground for non-volatile storage over the board flash of some data which can be part of the configuration (also factory) of the running application. This component, storage manager is a helper which can be used to access the TE0808 QSPI flash, however it should be compatible with any QSPI flash as it relies on some basic mechanisms shared across flash memories and on the AMD Xilinx BSP for Quad-SPI.

The library supports flashes in single, stacked and parallel mode and uses the JEDEC identifier to search among all officially supported Xilinx brands and flash geometries. Using the JEDEC id, the internal dictionary provides:

- Sector size and number of sectors
- Page size and number of pages
- Flash size
- Sector start address mask

- Number of dies

The library exposes the flash as three basic primitives (Read-Write-Erase). As a reminder, a flash memory cannot be written before being erased. The erase process (sector / bulk / die) is managed by the library. These exposed primitives are then used by another on-top built library which is used to structure a custom “file-format” on the flash, containing all the relevant data for the configuration.

The library was implemented always following MARTe2 standards, although it is not subject to the strict QA process (linting/coverage). It is also bundled with a test suite to be run on the Ultrascale+ board. It is foreseen its integration as part of the production step of the boards, to provide a factory initialisation and customisation (upload same firmware, customise parameters writing data on flash, when application starts reading and customises its execution based on flash contents).

4.14.13 AMP AND MIXED BARE-METAL / FREERTOS APPROACH

One of the peculiarities of the MARTe2 application scheme in the *MainFPGA* implementation is the asynchronous multiprocessing scheme with two distinct operating systems over three cores. The choice comes as a result of a performance assessment campaign over the ability to meet sampling rate design requirements. In fact, using FreeRTOS, high sampling rate tasks (PS to PL, Shared Memory, Networking, PS Algorithm) were impaired by a periodic glitch exhibiting jitters over 10 ms over the 100 µs required by the 10 kHz stream. The 10 ms glitch culprit was pinpointed to the FreeRTOS scheduler settings, in particular the tick rate, pre-emption and time slicing. An additional source of jitter, which could not be precisely assessed in the test environment, was due to the FreeRTOS task semaphore waking-up because of the arrival of packets on the network interface. Incrementing the tick rate up to 1 kHz was tested but was not deemed practical. FreeRTOS in fact does not impose a maximum tick rate, although some internal macros and time calculations cannot represent fractions of milliseconds, thus limiting the maximum attainable tick rate to 1 kHz. The increase in tick rate, in the *MainFPGA* scenario, brought no visible advantage but only a decrement in performances, due to the higher rate of context switching, causing the scheduler to effectively consume more processing power than the rest of application itself.

Due to these limitations and glitches, that were assessed during the initial design steps, the bare-metal approach has been chosen to suit the PS processing algorithm environment. Aside from the requirements and technical limitation of the RTOS, the bare-metal approach was preferred

due to the single-task nature of the application, giving to the MARTe2 porting the full control over the platform execution.

Conversely, the telemetry application, which requires a parallelism in operations, albeit for a small fraction of operations, was laid upon FreeRTOS. As the sampling and loop execution frequencies are one order of magnitude smaller than the default OS tick rate, the flexibility was deemed to be more significant.

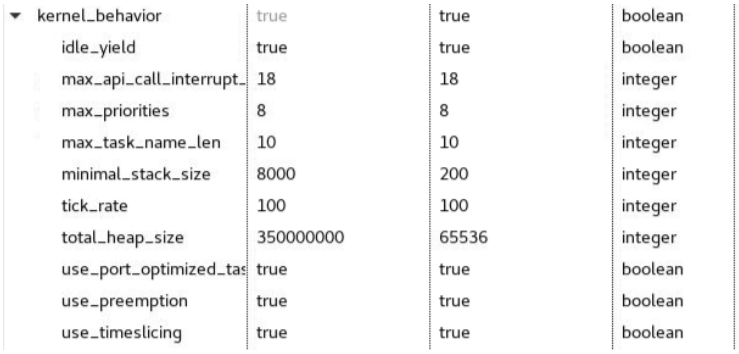


Figure 20 - Excerpt from the FreeRTOS kernel behaviour settings

4.14.14 ITER DATA EXCHANGE MODEL

The three network interfaces on the *MainFPGA* board stream data to the ITER network, through different data models: Data Archiving Network (DAN), Synchronous Data Network (SDN) and Plan Operational Network (PON). DAN is used for data acquisition which require high sampling rate and bandwidth for multichannel data. As a matter of fact, DAN data is directly processed and streamed through PL IPs, which is part of the fast controllers. SDN is also used in fast controller but relies on the synchronicity, where each node achieves full bandwidth for an allocated communication slot. SDN communication relies on the topic concept, in a publisher-subscriber pattern based on the IGMP multicast concept of groups. In simple terms, the SDN topic is converted with a hashing algorithm to a multicast IP address and port combination. The PON is a low sampling rate network (in the picture above it is also marked as S/T). PON is sample oriented and generally include EPICS PV control variables with all the metadata and attributes related. Sample rate on the PON network is up to 10 Hz. [51]

4.14.15 PERFORMANCE EVALUATION

The whole setup was thoroughly tested to verify requirements meetings. First verification is easily done by leaning on the PL sample production rate on the ring, by using the *PLPSDataSource* and its internal sample counter. As the PL (FPGA) internal timing is extremely reliable, running the DataSource and evaluating if two consecutive counter values

can be read from the ring without losses gives an upper bound of the performances. As the main DataSource synchronises on the PL data production, all the downstream computation must occur in the period between two consecutive samples. This kind of assessment was immediately accomplished by simply running the whole MARTe2 application and adding ADC channels to the processing chain until a resynchronisation event occurs on the PL to PS DataSource, meaning that the allotted runtime was violated.

First rough estimates were validated using MARTe2 timings DataSource, previously mentioned. The timing DataSource uses the MARTe2 high resolution timer as source, a component which was previously mentioned during the illustration of the porting steps. As aforementioned, high-resolution timer on the AMD Xilinx Ultrascale+ ARM Cortex A53 CPU core is backed by Xilinx BSP time functions, which in turn are a wrapper for the CNTPCT_EL0 register timer [49]. This gives a hardware native and reliable time base for measurements. Test campaign was conducted measuring execution time of the whole data exchange between two bare-metal cores by executing a portion of the final MARTe2 MainFPGA. The portion that was executed and tested included a shared memory and an UDP streaming DataSource and excluded the algorithm itself. The test was conducted in this way as the only missing performance metric was the jitter. As the algorithm execution is not a source of jitter, it was excluded, to give instead computational headspace to keep and process enough counter samples. Tests assessed jitter at 1, 10, 100, 1000, 2500, 5000, 10000 Hz frequencies by moving a small and a large payload of data between the two cores (100 bytes and 10000 bytes) and having already verified that DDR raw performances were orders of magnitude over the needed application performances. DDR performances were validated over datasheet declared rates using ZynqMP bundled DDR example and tests. Counter absolute values were converted in microseconds by dividing it for the sampling frequency (CNTFRQ_EL0) scaled value (33.333332). Results include both sides of the application and give a minimum, maximum, average, standard deviation value and the statistical distribution of the jitter across the min/max range for all the frequency/sizes couples. Only relevant extremes are presented, to avoid overwhelming data.

A small table in the beginning of each measurement campaign summarizes results and is followed by two charts, a line representing the time trend of the jitter and a histogram representing the distribution of samples over the jitter values.

Jitter line chart X axis reports the sample number (which can become time by multiplying it for the sampling frequency), Y axis reports the jitter expressed in microseconds.

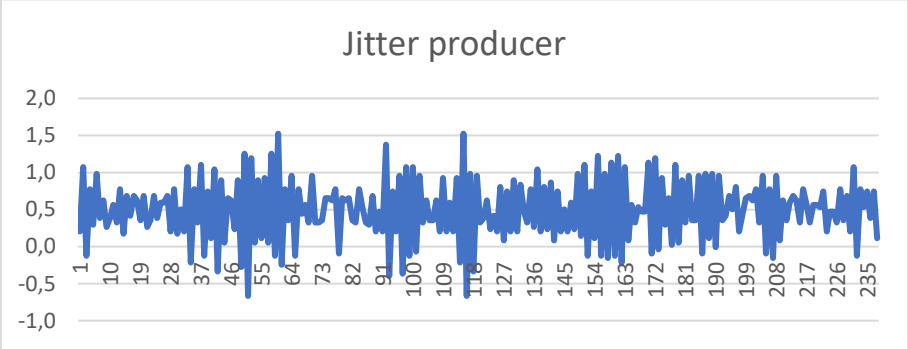
Jitter histogram chart reports the bin on the X axis, expressed as range in microseconds, spanning over the minimum and maximum jitter. On the Y axis is reported the number of samples falling in the bin.

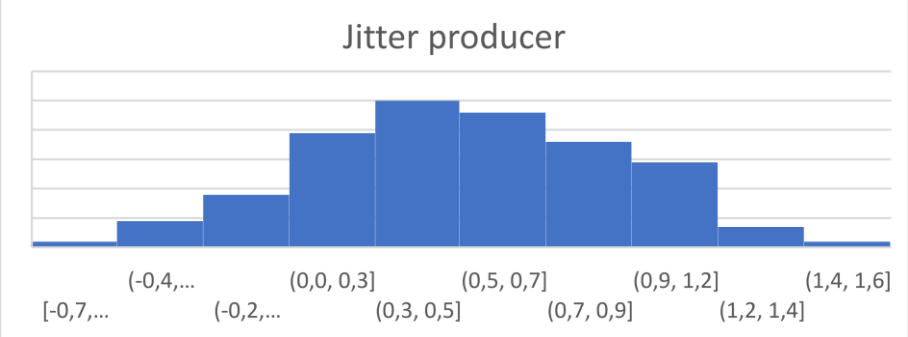
Results are presented one per page, to better show graphs and demonstrate no surfacing of periodic or peculiar patterns.

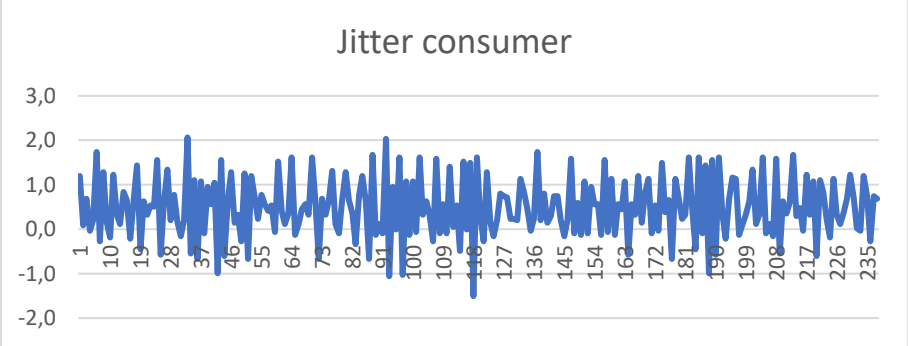
Presented results show that jitter maximum is well below 10 microseconds in the worst case growing as the loop frequency grows. Notice also that the maximum jitter is hit only occasionally during the time span, indicating that this is probably due to internally occurring interrupt (hardware queues). These measurements clearly show that the whole MARTe2 porting on the Ultrascale platform meets and exceeds the requirements for control loop frequency and jitter.

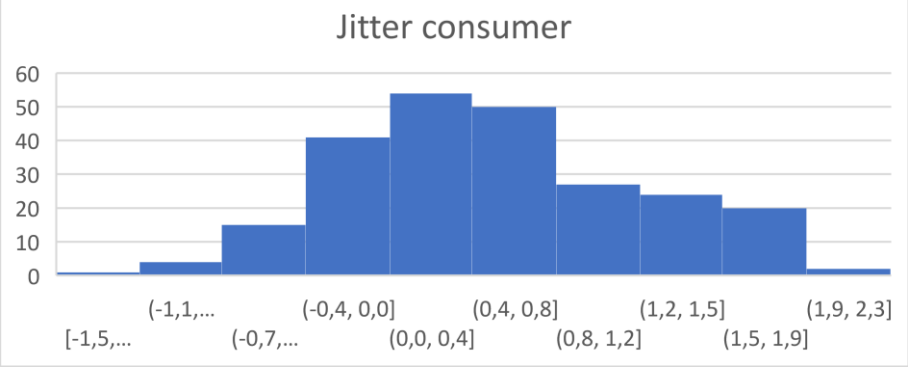
10 Hz, small payload

Jitter min		Jitter max		Jitter average		Jitter dev std	
Producer	Consumer	Producer	Consumer	Producer	Consumer	Producer	Consumer
-0,7 µs	-1,5 µs	1,5 µs	2,1 µs	0,5 µs	0,5 µs	0,41 µs	0,67 µs



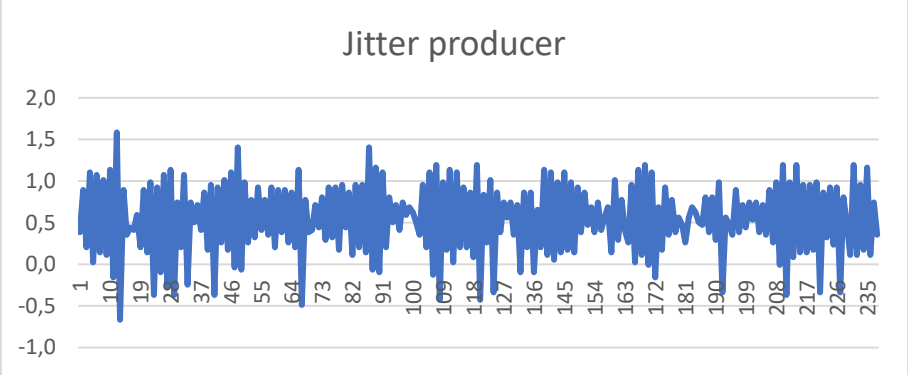


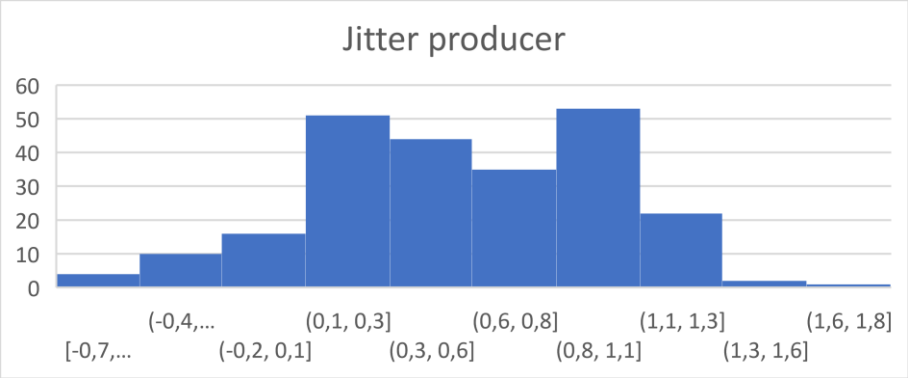


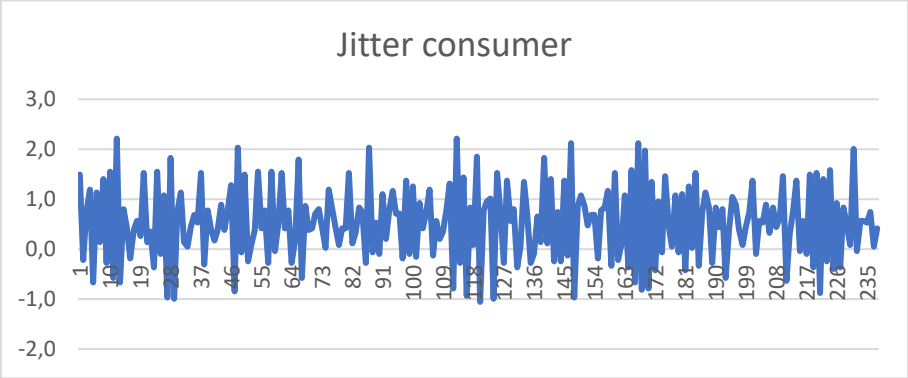


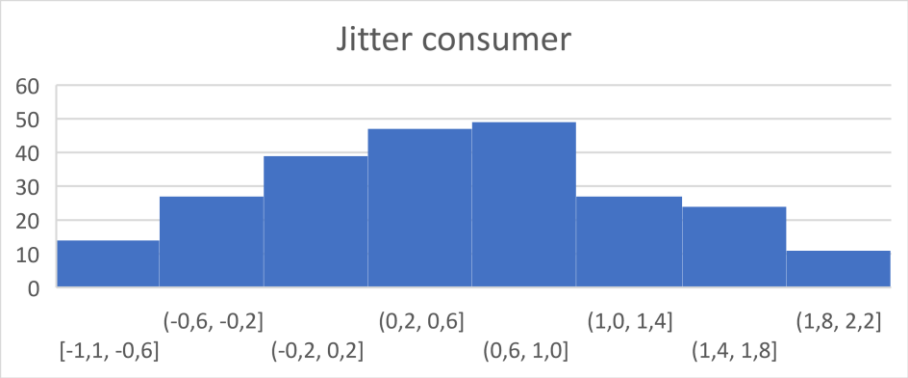
10 Hz, large payload

Jitter min		Jitter max		Jitter avg		Jitter devstd	
Producer	Consumer	Producer	Consumer	Producer	Consumer	Producer	Consumer
-0,7 µs	-1,1 µs	1,6 µs	2,2 µs	0,5 µs	0,5 µs	0,44 µs	0,73 µs



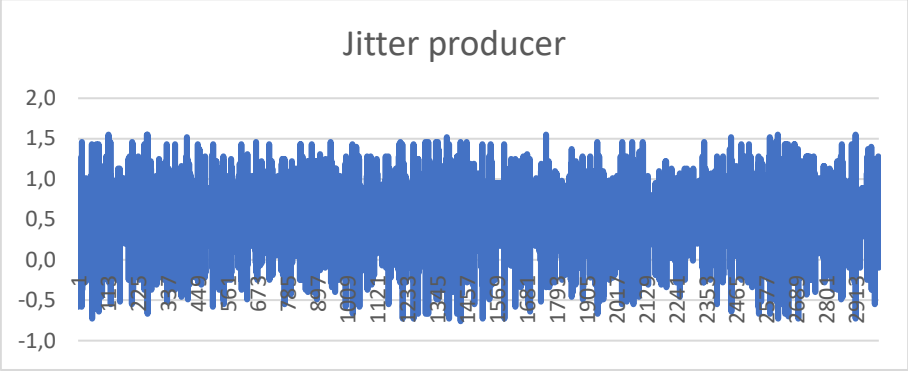


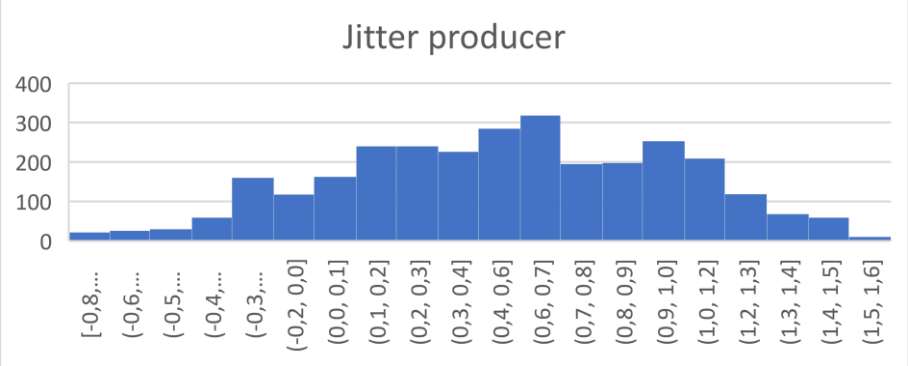


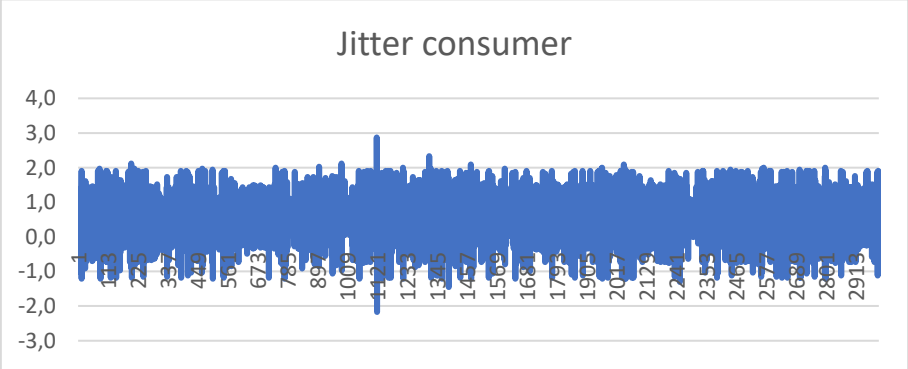


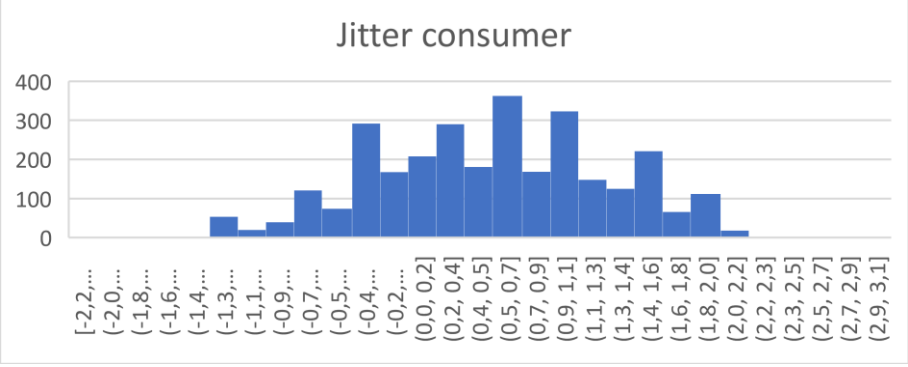
1 kHz, small payload

Jitter min		Jitter max		Jitter avg		Jitter devstd	
Producer	Consumer	Producer	Consumer	Producer	Consumer	Producer	Consumer
-0,8 µs	-2,2 µs	1,6 µs	2,9 µs	0,5 µs	0,5 µs	0,48 µs	0,74 µs



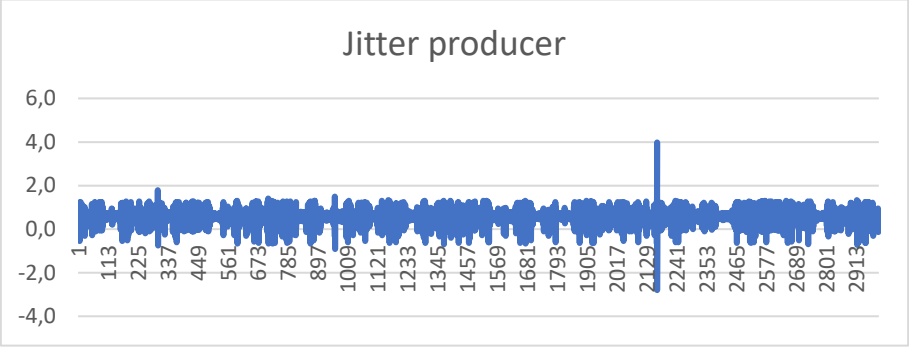


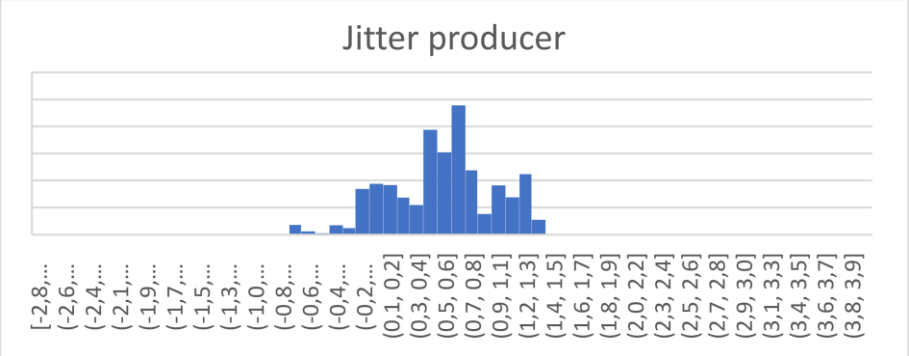


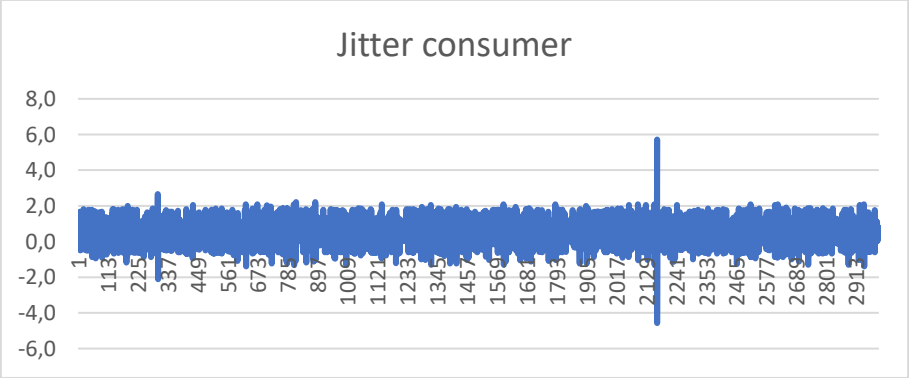


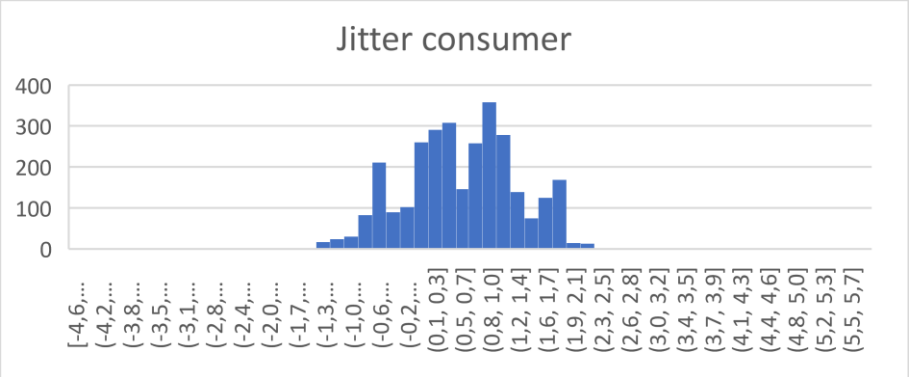
1 kHz large payload

Jitter min		Jitter max		Jitter avg		Jitter devstd	
Producer	Consumer	Producer	Consumer	Producer	Consumer	Producer	Consumer
-2,8 µs	-4,6 µs	4,0 µs	5,7 µs	0,5 µs	0,5 µs	0,43 µs	0,73 µs



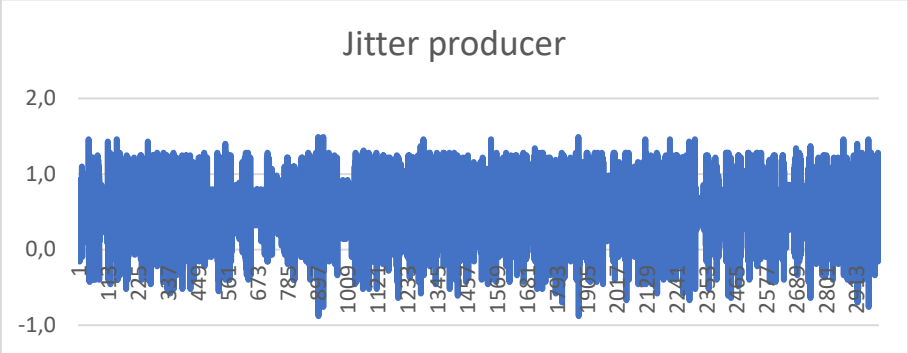


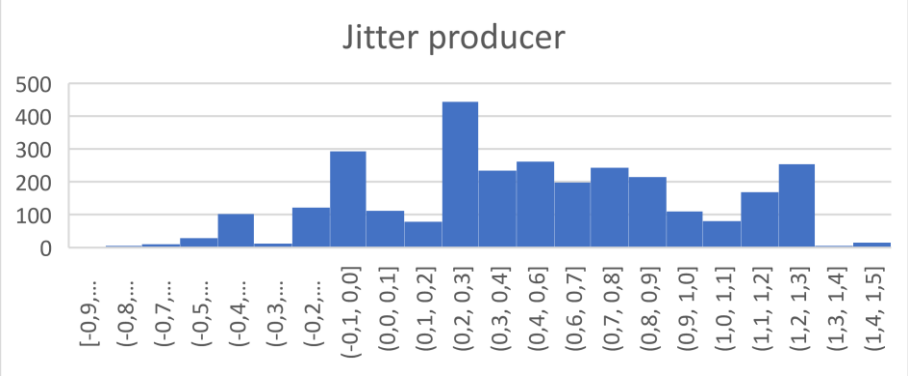


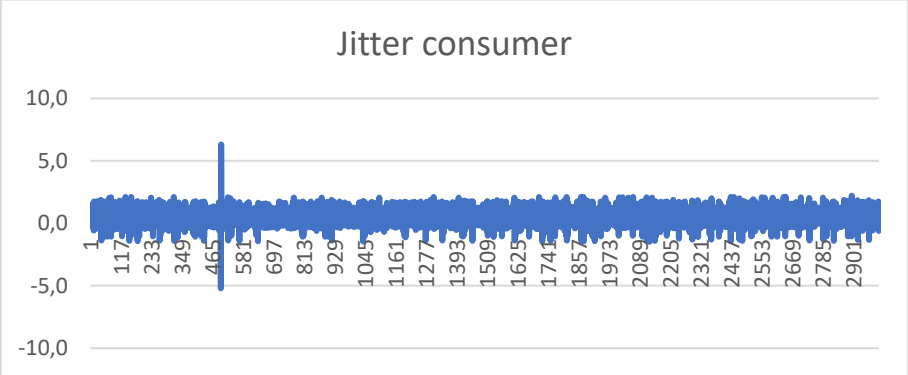


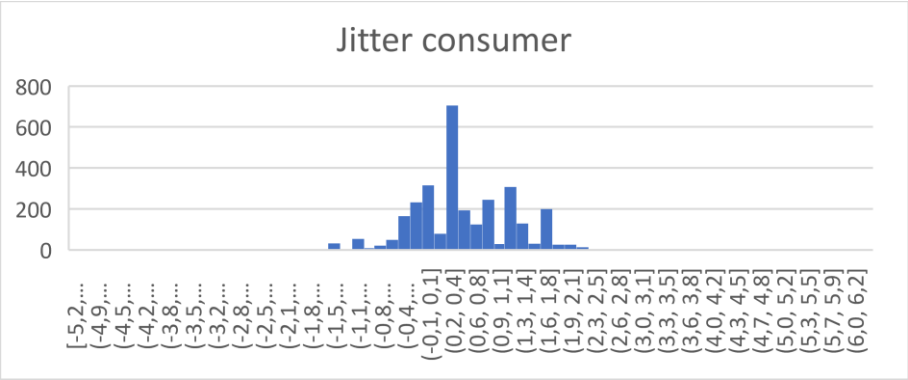
10 kHz small payload

Jitter min		Jitter max		Jitter avg		Jitter devstd	
Producer	Consumer	Producer	Consumer	Producer	Consumer	Producer	Consumer
-0,9 µs	-5,2 µs	1,5 µs	6,3 µs	0,5 µs	0,5 µs	0,47 µs	0,71 µs



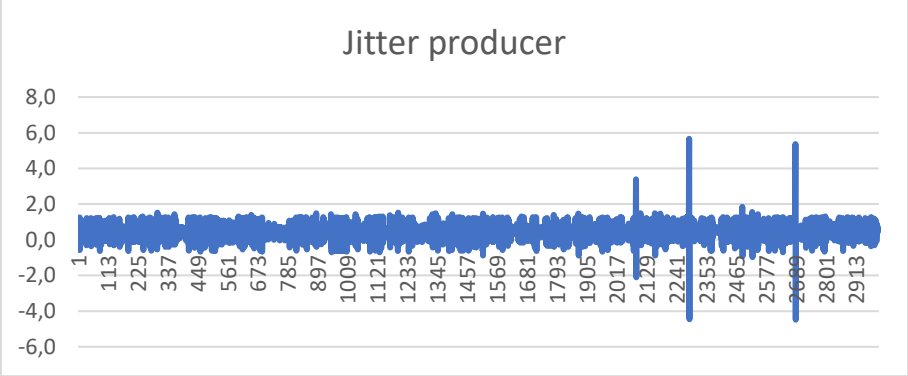


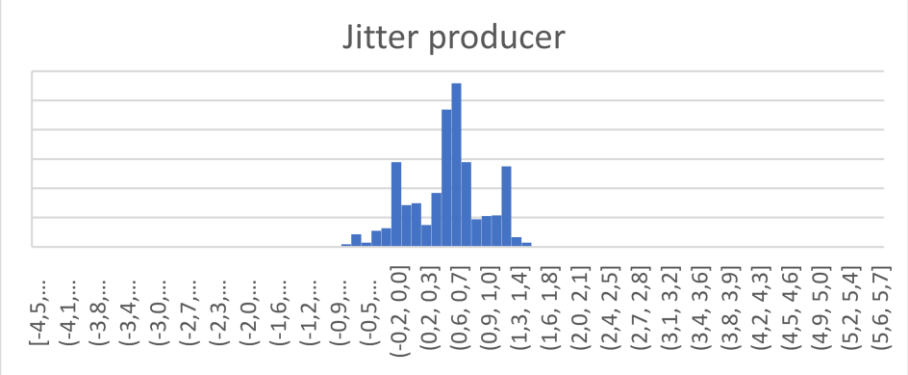


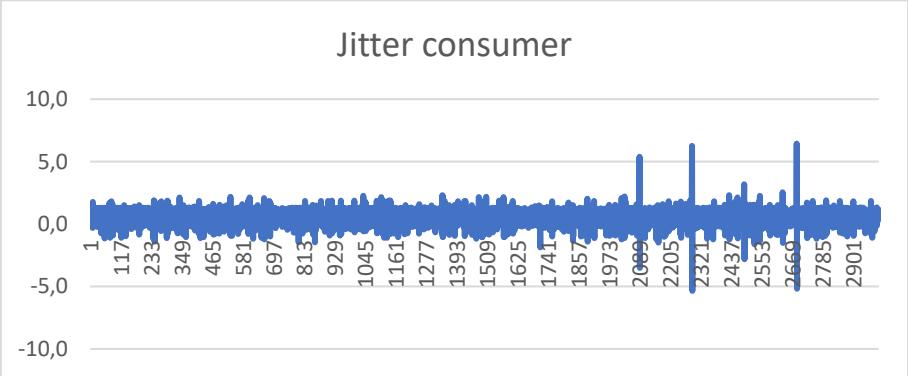


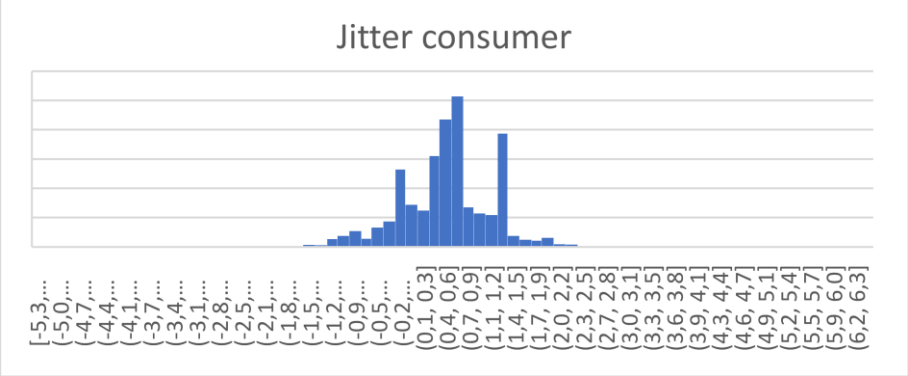
10 kHz large payload

Jitter min		Jitter max		Jitter avg		Jitter devstd	
Producer	**Consumer**	**Producer**	**Consumer**	**Producer**	**Consumer**	**Producer**	**Consumer**
-4,5 µs	-5,3 µs	5,7 µs	6,4 µs	0,5 µs	0,5 µs	0,49 µs	0,67 µs









4.15 MAINFPGA DEPLOYMENT

The solution for the MainFPGA board requires a multitude of steps to be correctly deployed. An outline of these includes:

- Creation of the hardware project, starting from the HDF file. The HDF is a sort of container file which contains all the further files needed for the project, including the FPGA bitstream and the board configuration. A hardware project is the base for all subsequent projects.
- Creation of the board support package (BSP) project. The SDK needs a BSP for each core / platform combination. The BSP is bound to the hardware project.
- Creation of the application project, bound to the BSP. The application project is essentially an empty container which will be used for the complete compilation and linking on the final ELF binary.
- Patching of the BSP related files, both by changing project variables and by modifying sources, for non-exposed parameters. Exposed parameters (e.g., FatFs interface) can be modified directly by issuing XSCT SDK commands and accessing BSP Project API variables. Non-exposed parameters require instead a direct-file patching approach (e.g., FatFs FF_USE_FIND and lwIP IGMP network card group join/leave to overcome 8 maximum number of IGMP groups that can be joined). These patches are applied using GNU patch mechanism, that in turn relies on several patch files that were created with incremental functionalities. Direct line modifications are instead accomplished using GNU sed (stream editor).
- Copying needed files into the application project (e.g., Si5345 library, MARTe2 configuration file).
- Setting the compiler, the needed include paths, link path, link libraries.
- Compile MARTe2, MARTe2 components, 55A0 Magnetics Component, Ultrascale support pack components.
- Create Shared Memory and Startup Manager projects.
- Setting up the LD script to suit for DDR splitting, OCM usage, shared memory location (both of ancillary and payload portions), heap and stack sizes.
- Put everything together, compile (where needed), link, generate the ELF binary and deploy (in RAM).

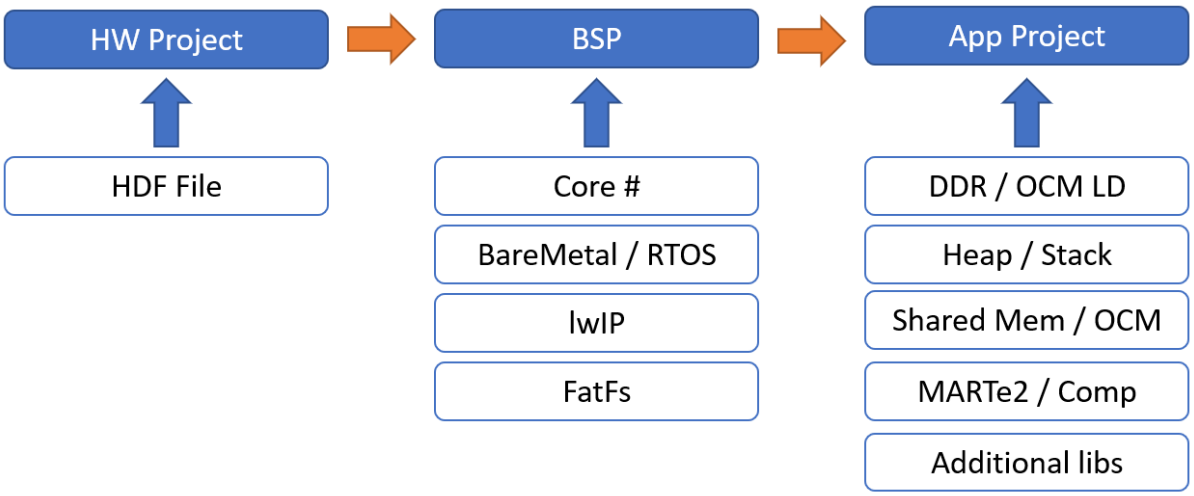


Figure 21 - Script execution and dependencies

All these steps also change, based on the configuration and carrying out this procedure manually is an extremely error prone task, which also requires a considerable amount of time. This led to the creation of a standardised system to prepare and deploy the MARTe2 application starting from scratch. The system is based on a series of scripts, mainly bash and TCL, which can setup the whole environment to run MARTe2 on an Ultrascale+ board. These scripts are created in a layered fashion, each one runs on top of another. Top-level script is the final customization bash, which coordinates the whole environment creation. TCL scripts rely on AMD Xilinx SDK internal scripts to generate the needed code base and project structure.

TCL scripts were structured in a configurable way, to be recycled across the different projects. They are independent (only their output artifacts have a dependency relationship) and have a complete command-line interface. The only vertical script is the final bash, which contains the platform and application specifics (type of application, core, needed libraries, network parameters, MARTe2 configuration file), as shown in following snippets.

```
    set options {
        { w.arg             "" "Workspace path" }
        { hw.arg            "" "Hardware project name" }
        { bsp.arg           "" "BSP project name" }
        { c.arg             "" "Core number" }
        { os.arg            "" "Operating system (standalone or
freertos10_xilinx" }
        { lwip.arg          "" "lwIP stack (raw or socket)" }
        { fatfs.arg         "" "FatFs setup (1 FS on SD Card - 2 FS in RAM)" }
        { rtosheap.arg      "" "FreeRTOS heap size" }
        { rtosstack.arg     "" "FreeRTOS stack size" }
    }
```

```
    set options {
        { w.arg             "" "Workspace path" }
        { hw.arg            "" "Hardware project name" }
        { bsp.arg           "" "BSP project name" }
        { c.arg             "" "Core number" }
        { os.arg            "" "Operating system (standalone or freertos" }
        { out.arg           "" "Output project name" }
        { ramstart.arg      "" "Core reserved RAM starting address" }
        { ramlen.arg        "" "Core reserved RAM length" }
        { heapsize.arg      "" "Size of the heap" }
        { stacksize.arg     "" "Size of the stack" }
        { shmemstart.arg    "" "Shared memory start address in RAM" }
        { shmemlen.arg      "" "Shared memory shared memory length" }
        { lmarte2              "Link MARTe2 and MARTe2-components" }
        { cstartup             "Creates and links the startup management
library"}
        { lstartup             "Links an already existing StartupManager
library"}
        { cshmem               "Creates and links the shared memory library" }
        { lshmem               "Links an already existing shared memory library"
}
        { iic                  "Copy iic header/sources for SI5345 init/setup" }
        { main                 "Include the main file for MARTe2 and core"}
    }

}
```

A script, which is not actually part of the suite but is foreseen its integration in a further step, was implemented to deploy MARTe2 on the QSPI flash. It takes care of the erase and write process of MARTe2 on the flash memory and has as requirement the creation of a first-stage bootloader (FSBL). The FSBL is in charge of initialising the platform and loading the bitstream onto the FPGA fabric. The FSBL and QSPI boot process was also tested. The FSBL however needs the setting of some physical dipswitches on the board to change the boot behaviour. To overcome the limitation of accessing physically to the board, a substitution to manually changing the dip was found and is based on the direct access to the BOOT_MODE_USER (CRL_APB) register using XSCT and command line to write it. The register can assume a series of values to boot from the PS JTAG, QSPI in 32- and 64-bit mode, SD Card, NAND, eMMC, USB, and PJTAG, with some of them being unavailable on the MainFPGA carrier board. The register is volatile (lost after a power cycle) and was a convenient quick way to test the MARTe2 deployment on the flash NV storage. However, this script and the whole procedure of storing

and booting MARTe2 from the QSPI needs to undergo some design, implementation and refinement steps, which are part of the foreseen work.

4.16 THE BEST IP BOARD

Using the previous accumulated knowledge, code base and scripts, another environment for the MARTe2 execution was implemented. The environment is suited for the Best Ip board, the aforementioned system in charge of computing the plasma current. While environment preparation was almost straightforward, requiring only small adaptations to the script, most of the work was concentrated on understanding Xilinx lwIP contrib porting due to a limitation on their implementation. The Best Ip board has to subscribe 14 IGMP multicast groups on SDN, each stream containing the 30 ADC channels from the 55.A0 Magnetic Diagnostics (MainFPGA) board to compute the best plasma current parameter.

Current lwIP implementation by default supports only 4 IGMP groups, this number can be increased to 8 by changing the MEMP_NUM_IGMP_GROUPS parameter placed inside the main opt.h stack configuration header. To further increase the number of join-able IGMP groups, a more substantial modification was made on the concrete hardware porting of the stack (xemacpsif). The modification removes the limitation by directly calling the board hash update mechanism with the group information, skipping the management part internally used to keep track of the currently joined groups.

Work on the *Best Ip* board is still undergoing, at the moment of this writing it is under test to verify the basic performance requirements are met, with respect to the stream incoming from the IGMP subscriptions. While the *MainFPGA* suite is already available on the main branch, *Best Ip* work still is not production ready as it misses parts of the processing algorithm and telemetry implementation details.

4.17 OTHER MINOR CONTRIBUTIONS

Aside from the two main development tasks, a series of smaller contributions were also brought to MARTe2. Some of these were assigned during the initial period of the collaboration, to become familiar with MARTe2 development, the QA and CI/CD chains. The contribution for these is marginal and no particular challenges were encountered that are worth of mention. However, they represented a key step for the further development, both for their gradually

increasing effort and comprehension of the whole framework. Amongst these contributions, work on an already existing component is worth mentioning instead: the Simulink Wrapper GAM.

The *SimulinkWrapperGAM* is a GAM component of the MARTe2 components bundle which is able to run a Simulink® model generated and compiled with the MATLAB® Embedded Coder® tool. Although this component had already been actively developed and deployed, a new feature was requested from the user base: the ability to handle structured signal seamlessly in MARTe2. [52]

The feature was implemented, the *SimulinkWrapperGAM* is now able to deal with structured signals (buses) as inputs and outputs. Reworking of the GAM code required new QA procedures (linting, add coverage for the new test cases) and is now available on the MARTe2 components official repository.

Another small contribution was related to the VCIS system (GitLab, Wiki, Redmine, Jenkins), finalised to upgrade the whole underlying Linux distribution to CentOS 7 and to the staging of a Zabbix setup to monitor all the machines which compose the whole VCIS infrastructure. The migration procedure formalisation and testing, developed together with Fusion for Energy team members was carried out. The Zabbix monitoring task was accomplished by installing the Zabbix server on a dedicated machine to supervise with agents every network component. This task also included the creation of rules for warning and alarming for the whole infrastructure, dealing with operational status of each system (usage, free disk space, online status).

4.18 Conclusions and foreseen work

The porting work started with a generic, readily available platform, with the same CPU as the final production platform. The development underwent a series of cycles, also touching other heavily different platforms (STM32), which helped gain needed abstractions to produce a general procedure to implement the porting. Following, the implementation of the porting stub and scaffolding allowed the necessary refinement to become able to move to the AMD Xilinx Ultrascale+ platform in a streamlined and organized way.

The Ultrascale+ implementation presented several challenges, especially due to the asynchronous multiprocessing configuration that was chosen for the execution environment. The challenges were related to the correct handling of the CPU internals, the interrupt and the

correct routing, the management of the I and D caches, the TLB and MMU settings to allow private core and shared memory. Basic functionalities and performance assessment procedures led to several further development and refinement steps, to move towards a full understanding of the platform.

Development and debugging, on a multicore AMP system is an extremely challenging task, that was greatly helped by the MARTe2 modular structure. MARTe2 layers were introduced, debugged and profiled gradually. The final integration works of MARTe2, developed components and hardware interfacing was eased thanks to the MARTe2 configuration file structure, which allowed quick reconfiguration of the whole platform by just changing it.

Once porting was finished, the further step towards continuous integration and continuous delivery (CI/CD) led to the creation of generic scripts which now allow the quick setup of a MARTe2 application ready to be deployed on the platform. Moreover, the whole experience helped develop a procedure and code with the portability in mind, with several re-usable components and a solid and comprehensive documentation and code stub suite.

The work for the MainFPGA and Best IP will be part of the ITER Magnetics Diagnostics system, serving and processing magnetic sensor data to feed the plant controller and safety systems. However, this work is already foreseen as a viable opportunity for other experiments which involve the MARTe2 framework in control loops for the field. A notable example is represented by the Divertor Tokamak Test (DTT) experiment in Frascati (Rome), where a similar SoM will be employed for data acquisition loops.

This thesis work consolidated some MARTe2 concepts and practices into a standardised procedure for the development of further portings. By adding the support for the ARM based platforms, this work laid the foundation for the employment of embedded platforms running MARTe2 as convenient, flexible and proven systems to run control solutions also for loops running at tens of kHz sampling rates with high number of channels, also considering network streaming.

The porting for the MCU series also opens a set of opportunities where these hardware solutions become directly part of field controllers, running a distributed control algorithm. The usage of MARTe2 over the development of custom firmware adds an enormous degree of flexibility, maintainability, robustness and ease of development. The employment of MARTe2 as alternative for firmware development on these systems also enables team working and co-operation, where developers concentrate on platform aspects leaving control issues to the field

experts. The ability to use already existing or develop own components and reuse them across different solutions allows the rapid production of devices which are already field-ready. Moreover, once developed and tested, the same MARTe2 application setup can be run on different architectures. This demonstrated extremely important as debug and test on embedded platform is always a more difficult and time-consuming task, with respect to standard application debugging.

The porting on MCU, together with the PROFINET is deemed to be a work with some potential wider audience and scope of application and will become part of some further implementation in the near future. It is foreseen that MARTe2 running MCU platforms, together with an industry standard protocol can represent a very interesting ground for the development of powerful and flexible field controllers, also as a cheaper alternative to the current consolidated solutions.

In short words, MARTe2 demonstrated, also through this work, to be able to bring Hardware-In-The-Loop, Software-In-The-Loop platforms and vertical firmware development combined advantages in systems ranging from the low-cost microcontroller to the HPC running complex models, bringing the same, consistent approach across them. This work also demonstrated the suitability of MARTe2 on MPSoC running bare-metal AMP for high performance control loop schemes.

To conclude, MARTe2 represents an extremely flexible, ready and proven framework for the development of applications for the control. The support for different hardware platforms, its current portings and streamlined process to move towards new platforms give it an extremely convenient alternative to the vertical custom firmware development. Its approach based on functions and connections allows also a separation of tasks which allows field control expert to elaborate the control solution by working on the MARTe2 configuration file and leaving implementation and platform details to the MARTe2 core and components developers.

5. CONCLUSIONS

The presented work follows a logical path through the modelling, the implementation and the realisation of a control system, particularly oriented for the nuclear fusion field. The implemented code, as a result of the collaboration with the Fusion for Energy CODAC group, will become part of the ITER experiment plant I&C.

The PROFINET component will become part of the ITER Electron Cyclotron Resonance Heating (ECRH) factory acceptance test suite (FAT-Tools), to emulate the field periphery to help develop and validate control algorithms, in particular for the gyrotrons power supplies.

The streamlining of the porting process and the analysis of the procedure on several heterogeneous systems allowed the introduction of key features to the framework, which laid the foundation for the further development of the MARTe2 framework. These introductions helped moving MARTe2 framework towards the embedded SoC and MCU world. The flexibility, reliability and dependability of the MARTe2 framework running on these devices offer a lean yet powerful approach to the real-time control systems solutions development.

The porting for the Ultrascale+ platform and the integration of the sensor data processing algorithms will become part of the ITER Magnetics Diagnostics system, helping process and stream data from the magnetics sensors to the Plasma Control System.

On a bigger picture, the usage of MARTe2 framework on these platforms, allows an alternative approach to the custom firmware development, oriented to the control field. Leaning on MARTe2 brings all the framework advantages, in terms of quality of the solutions and ability to develop more complex schemes. The employment of microcontrollers, like the STM32, with MARTe2, allow the development of complex solutions for the control, also bringing the advantage of being field-ready, with plenty of analog and digital interfaces.

As these platforms, both SoC and MCU, offer also a lot of other features (GPU, RPU, DSP), a lot of work can be done in the direction of a further MARTe2 integration, to leverage their full functionalities and processing power.

REFERENCES

[1] J. Kenneth Shultis and R. E. Faw, Fundamentals of Nuclear Science and Engineering, New York - Basel: Marcel Dekker Inc., 2002.

[2] International Atomic Energy Agency IAEA, Fusion Physics, Vienna: International Atomic Energy Agency IAEA, 2012.

[3] J. Wesson, Tokamaks, Oxford: Oxford University Press, 2004.

[4] M. E. Sawan and M. A. Abdou, "Physics and Technology Conditions for Attaining Self-Sufficiency for the D-T Fuel Cycle," *Fusion Engineering and Design,* 2005.

[5] C. Corradino, Buscarino A., L. Fortuna and A. Murari, "Modeling spatiotemporal complexity during plasma instabilities," *European Conference on Circuit Theory and Design ECCTD,* pp. 1-4, 2017.

[6] R. B. White, The Theory of Toroidally Confined Plasmas, London: Imperial College Press, 2014.

[7] R. Fitzpatrick, Plasma Physics, CRC Press, 2014.

[8] J. Journeaux and A. Wallander, Plant Control Design Handbook, ITER Technical Report, 2020.

[9] G. De Tommasi, F. Piccolo and F. Sartori, "A flexible and reusable software for real-time control applications at JET," *Fusion Engineering and Design,* vol. 74, pp. 515-520, 2005.

[10] ITER, "ITER," [Online]. Available: https://www.iter.org. [Accessed 15 08 2022].

[11] ITER, "ITER Project Milestones," [Online]. [Accessed 15 08 2022].

[12] RFX Consortium, "RFX Consortium," [Online]. Available: https://www.igi.cnr.it/chi-siamo/. [Accessed 15 08 2022].

[13] P. Piovesan, D. Bonfiglio, F. Auriemma, F. Bonomo, L. Carraro, R. Cavazzana, G. De Masi, A. Fassina, P. Franz, M. Gobbin, L. Marrelli, P. Martin, E. Martines, B. Momo, L. Piron, M. Valisa, M. Veranda, N. Vianello, B. Zaniol, M. Agostini, M. Baruzzo, T. Bolzonella, A. Canton, S. Cappello, L. Chacòn, G. Ciaccio, D. F. Escande, P. Innocente, R. Lorenzini, R. Paccagnella, M. E. Puiatti, P. Scarin, A. Soppelsa, G. Spizzo, M. Spolaore, D. Terranova, P. Zanca, L. Zanotto and M. Zuin, "RFX-mod: A multi-configuration fusion facility for three-dimensional physics studies," *Physics Of Plasmas,* no. 20, 2013.

[14] ENEA Italian National Agency for New Technologies, DTT Divertor Tokamak Test Facility - Project Proposal, Frascati: Frascati Research Center, 2015.

[15] European Joint Undertaking for ITER and the Development of Fusion Energy, "Who We Are," [Online]. Available: https://fusionforenergy.europa.eu/who-we-are/. [Accessed 15 08 2022].

[16] European Joint Undertaking for ITER and the Development of Fusion Energy, "Our Mission & Values," [Online]. Available: https://fusionforenergy.europa.eu/our-mission-values/. [Accessed 15 08 2022].

[17] G. De Tommasi, F. Piccolo and F. Sartori, "A flexible and reusable software for real-time control applications at JET," *Fusion Engineering and Design,* vol. 74, pp. 515-520, 2005.

[18] T. Bolzonella, V. Igochine, S. C. Guo, D. Yadikin, M. Baruzzo and H. Zohm, "Resistive Wall Mode Active Rotation in RFX-mod".

[19] G. Marchiori, M. Baruzzo, T. Bolzonella, Y. Q. Liu, A. Soppelsa and F. Villone, "Dynamic simulator of RWM control for fusion devices: modelling and experimental validation on RFX-mod," *Nuclear fusion,* vol. 52, no. 2, 2012.

[20] G. De Tommasi, F. Piccolo, A. Pironti and F. Sartori, "A flexible software for real-time control in nuclear fusion experiments," *Control Engineering Practice,* vol. 14, pp. 1387-1393, 2006.

[21] A. Neto, F. Sartori, F. Piccolo, A. Barbalace, R. Vitelli and H. Fernandes, "Linux real-time framework for fusion devices," vol. 84, pp. 1408-1411, 2009.

[22] A. Neto, F. Sartori, F. Piccolo, R. Vitelli, G. De Tommasi, L. Zabeo, A. Barbalace, H. Fernandes, D. F. Valcarcel and A. J. N. Batista, “MARTe: A Multiplatform Real-Time Framework,” *IEEE Transactions on Nuclear Science*, vol. 57, no. 2, pp. 479-486, 2010.

[23] T. Bellizio, G. De Tommasi, N. Risoli, R. Albanese and A. Neto, “A MARTe based simulator for the JET Vertical Stabilization System,” *Fusion Engineering and Design*, vol. 86, pp. 1026-1029, 2011.

[24] Fusion for Energy, “MARTe2 Overview,” [Online]. Available: https://vcis.f4e.europa.eu/marte2-docs/master/html/overview.html. [Accessed 15 08 2022].

[25] Fusion for Energy, “MARTe2 Code organisation,” [Online]. Available: https://vcis.f4e.europa.eu/marte2-docs/master/html/core/structure/code.html. [Accessed 08 15 2022].

[26] Fusion for Energy, “MARTe2 Real-time Applications,” [Online]. Available: https://vcis.f4e.europa.eu/marte2-docs/master/html/core/gams/rtapp.html. [Accessed 15 08 2022].

[27] Fusion for Energy, “MARTe2 GAM,” [Online]. Available: https://vcis.f4e.europa.eu/marte2-docs/master/html/core/gams/gam.html. [Accessed 15 08 2022].

[28] Fusion for Energy, “MARTe2 DataSource & Brokers,” [Online]. Available: https://vcis.f4e.europa.eu/marte2-docs/master/html/core/gams/datasource.html. [Accessed 15 08 2022].

[29] Fusion for Energy, “MARTe2 Quality Assurance,” [Online]. Available: https://vcis.f4e.europa.eu/marte2-docs/master/html/contributing/qa/qa.html. [Accessed 15 08 2022].

[30] Fusion for Energy, “ProfinetDataSource MARTe2 Components,” [Online]. Available: https://vcis-gitlab.f4e.europa.eu/aneto/MARTe2-components/-/tree/master/Source/Components/DataSources/ProfinetDataSource/Docs. [Accessed 15 08 2022].

[31] GTD Science, Infrastructure & Robotics, “IDM Code D02.07 FAT Tools Design”.

[32] PROFIBUS Nutzerorganisation e.V. (PNO), PROFINET System Description Technology and Application, Karlsruhe: PROFIBUS Nutzerorganisation e. V. (PNO), 2014.

[33] RT-LABS, “P-NET,” [Online]. Available: https://rt-labs.com/product/profinet-device/. [Accessed 15 08 2022].

[34] RT-LABS, “P-NET,” [Online]. Available: https://github.com/rtlabs-com/p-net. [Accessed 15 08 2022].

[35] RT-LABS, “OSAL,” [Online]. Available: https://github.com/rtlabs-com/osal. [Accessed 15 08 2022].

[36] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design patterns, software engineering, object-oriented programming, Addison Wesley, 1994.

[37] G. Avon, “Correct shutdown sequence #315,” 26 01 2021. [Online]. Available: https://github.com/rtlabs-com/p-net/issues/315. [Accessed 15 08 2022].

[38] G. Carannante, M. Cavinato, K. Cindric, M. Ferrari, G. Ferrò, A. C. Neto and F. Sartori, “Status of the ITER ECRH&CD control system development”.

[39] A. Buscarino, G. Avon, J. Stillerman, S. Lane-Walsh, F. Santoro, G. Manduchi, E. De Marchi, F. Zanon, F. Sartori and A. C. Neto, “Integration of tools for management and control of fusion experiments,” in *OMC Med Energy Conference & Exhibition*, Ravenna, 2021.

[40] AMD Xilinx, “UltraScale Architecture and Product Data Sheet: Overview,” 2022.

[41] Google, “Googletest Primer,” [Online]. Available: https://google.github.io/googletest/primer.html. [Accessed 15 08 2022].

[42] G. Avon, A. Buscarino, A. C. Neto and F. Sartori, “MARTe2 embedded signal processing unit for the ITER Magnetics Diagnostics,” in *IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society*.

[43] Fusion for Energy, “System Design Description (DDD) 55.A0 Magnetic Diagnostic,” 2022.

[44] Fusion for Energy, “55.A0 - DA DD - Plant System Controller,” 2020.

[45] Fusion for Energy, “55.A0 - DA DD - Plant System Controller”.

[46] GCC the Gnu Compiler Collection, “Built-in Functions for Memory Model Aware Atomic Operations,” [Online]. Available: https://gcc.gnu.org/onlinedocs/gcc/_005f_005fatomic-Builtins.html. [Accessed 15 08 2022].

[47] GCC the Gnu Compiler Collection, “Other Built-in Functions Provided by GCC,” [Online]. Available: https://gcc.gnu.org/onlinedocs/gcc/Other-Builtins.html. [Accessed 15 08 2022].

[48] G. R. Andrews, Foundations of Multithreaded, Parallel, and Distributed Programming, Addison Wesley, 2000, pp. 100-101.

[49] ARM, Programmer’s Guide for ARMv8-A, 2015.

[50] AMD Xilinx, “UltraScale Architecture System Monitor,” 2021.

[51] R. Castro, Y. Makushok, L. Abadie, J. Vega and J. Faig, “Data model implementation in ITER data archiving system,” *Fusion Engineering and Design,* vol. 146, pp. 1903-1906, 2019.

[52] Fusion for Energy, “SimulinkWrapperGAM MARTe2 Components,” [Online]. Available: https://vcis-gitlab.f4e.europa.eu/aneto/MARTe2-components/-/tree/master/Source/Components/GAMs/SimulinkWrapperGAM. [Accessed 15 08 2022].