Politecnico di Torino

Master's Degree Course in Computer Engineering

Master's Degree Thesis

Development and test of an Iron Bird

Supervisor:
Prof. Maurizio Morisio

Candidate:
Angelo Cordaro

Co-supervisor:
Prof. Luca Ardito
Prof. Riccardo Sisto

Academic year 2020/2021
Torino

To Gaetano, Paola and Lucia

"We may have our differences,
but nothing's more important than family."

- Miguel,'Coco'

Abstract

The thesis project develops an *Iron Bird*, a simulation of a wing of a regional airplane. The simulator is partially physical(half wing,sensors,actuators) and partially virtual(second half wing).

The project is part of the *AstIb Clean Sky2(CS2) European project.* The ASTIB (development of Advanced Systems Technlogies and Hardware/Software for the flight simulator and Iron Bird demonstrators for regional aircraft) project brings together 7 European companies and academic partners led by LEONARDO. It aims at supporting the improvement of the Technlogical Readiness Level for a significant number of equipments that are being considered of critical importance for the future Green Regional Aircraft (GRA). The design and production of the Iron Bird is responsibility of CERTIA. The Iron Bird is the ground test bench allowing the integration of the different aircraft systems. This Iron Bird is equipped with new innovations(semi-virtual,innovative loading systems, health monitoring,etc..). The thesis work is about receiving the models (developed by aeronautical engineers), deploying them on the computers, testing and improving them until complete validation. The simulation of a wing during a flight requires several computers: **FMSC**(Flight Mechanics Simulation Computer) to simulate the flight dynamics; **FCC**(Flight Control Computer) to control the airplane during the flight; **HMS**(Health Management System) for reactive and proactive maintenance. The computers rely on various MatLab/Simulink models.

The workflow follows the principles of the *Model-based Software Design*(MBSD), that is based on the development of a model of the plant and the controller focusing on the details that are useful to understand system's behaviour. A significant advantage of following that approach is the automated code generation, which allows to automatically translate the Simulink model into code that can be executed on a dedicated hardware. In a first phase models are tested in a simulation environment, then they have to be executed in real time to verify the interactions between the models and the shared memory. Executing a model in real time requires the usage of high-performance platforms. The chosen one is NI VeriStand, a software designed also for Hardware-in-the-loop(HIL) and Software-in-the-loop (SIL) simulations. In order to import the model in VeriStand we need a **dll**(dynamic linkink library) file. It is obtained as result of code generation. The models received from

the partners are modified to make them suitable for code generation. This means substituting every source block with an Input Port and every sink block with an Output Port. Then code is generated and the model, translated into a dll, is ported to NI VeriStand. Next step is, exploiting VeriStand, to map inputs and outputs on the reflective memory. In this way the 3 models composing the project shall be able to interact with each other. Last step is, indeed, to verify their interactions.

Contents

List of Figures

Chapter 1

Introduction

1.1 A More Electrical Aircraft

Air transport contributes today about 3% to global greenhouse gas emissions, with traffic expected to triple by 2050. Although other sectors are more polluting (electricity and heating produces 32% of greenhose gases), this expected growth makes it necessary to address aviation's environmental impact. Meeting the EU's climate and energy objectives will require a drastic reduction of the sector's environmental impact by reducing its emissions. Maximising fuel efficiency, using less to go farther, is also a key cost-cutting factor in a very competitive industry - and as air traffic increases, better noise reduction technologies are needed. But game-changing innovation in this sector is risky, complex and expensive, and requires long-term commitment. That's why all relevant European stakeholders must work together to develop proof-of-concept demonstrators.

Reducing the power consumption and thus the fuel burn is a major target for the next generation of aircraft. Two technological areas on which can be possible work are wing load alleviation and electrical actuation. Load alleviation is a technique for redistributing aircraft loads encountered during flight with the purpose of reducing the wing root bending moment. Electrical actuation can contribute to the reduction of the non-propulsive power because electromechanical actuators rely on a power less subject to losses and lighter to distribute, besides presenting higher reliability and maintainability with a lower life-cycle cost. These two areas have been widely addressed in the past years.

Over the last years, several industrial programmes initiated the concept of a More Electric Aircraft. In particular, the aero-equipment industry has started to develop a more electrical actuation with *Electro Hydrostatic Actuators*(EHA) and to introduce *ElectroMechanical Actuators*(EMA) for auxiliary equipment. This has provided incremental approaches to address hydraulic circuits issues (Power-by-Wire technologies, 2-hydraulic/2-electric power distribution architecture and use of EMA for some systems).

Several collaborative research and development projects also started to develop an **All-Electric Aircraft**, showing the effectiveness of electrical actuation. EMA systems are so considered the best option for the aircraft of the future considering they are:

- Less complex;
- Better suited to long term storage;
- Energy saving with respect to hydraulic systems;
- Installation and maintenance are easier;
- Power distribution management easier.

Nevertheless, there still exist technological barriers for a widde adoption of EMA. For example, the sensitivity to certain single point of failures that can lead to mechanical jams, and so on.

The promising perspectives of load alleviation / load control technologies as well as electrical actuation for flight control surfaces and landing gear need to be thoroughly investigated and verified in order to gain the necessary confidence and maturity level for moving to their implementation in a flying demonstrator. This requires:

- Development of suitable prototype components integrating the innovative features capable of making electrical actuation an accepted proposition for future flight controls and landing gears;

- Design and construction of an integration test rig (Iron Bird) allowing verification and validation of:
  - Enhanced electrical power distribution and load management technology;
  - Electrical landing gear technology;
  - Flight control system technology
- Development of a health monitoring platform able to collect and process data provided by the sensors of the aircraft structure and systems such to assess the aircraft health status.

The Iron Bird will enable integration and testing of the new technologies in a relevant environment.

1.2 The ASTIB project

The **ASTIB**(development of Advanced Systems Technologies and hardware/software for the flight simulator and Iron Bird demonstrators for regional aircraft) project brings together 7 European companies and academic partners led by LEONARDO. Its goal is to develop new technologies for the future regional aircraft. ASTIB is specifically focused on contributing to generate technological advancements to be implemented in a future Green Regional Aircraft(GRA); its main objectives are to support the improvement of the Technological Readiness Level up to above TRL 5 for a significant number of electrical equipment that are being considered of critical importance for the future GRA. This will help supporting industrial application decisions for future deployments of GRA.

In particular, ASTIB will develop:

- Electromechanical actuators(EMAs) with their associated electronic control units (ECUs) for selected flight control surfaces;
- Electrically actuated one main and one nose landing gears;
- Reliable prognostics and health management functions for the electromechanical actuators;

- An advanced multi-functional integration, verification and validation test rig ("Iron Bird");
- Contribution to the development of a health monitoring validation platform;
- Tools for evaluation of the benefits of an integrated health monitorin platform.

1.3 Model-based Software Design

The project has been developed following the principles of *Model-based Software Design*. To understand what this means, let's start with an example. Let's think we want to design a controller for an industrial robot or a vehicle and put our attention on the code necessary. Software complexity is becoming one of the dominant cost items in several sectors, including automotive, avionic and industrial markets. It can be reduced using:

- a structured process
- a more effective way of developing and validating the code,
- a more effective way of reusing code.

Following a traditional approach implies handwritten code and the only way to verify requirements fulfilment is through a "trial and error" procedure. Tests can be done only at the end of the process and on a physical prototype, increasing costs. If a single requirement changes, the whole system must be redesigned, increasing the already large time needed for production and delivery of the final product.

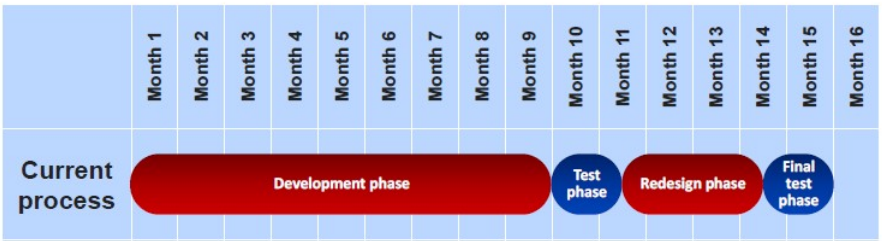


Figure 1.1: Traditional approach timetable

As can be seen in figure 1.1, test and development phases are strictly separated and if a redesign phase is needed, the time to have a finalized product becomes longer (15 months). This is not acceptable nowadays.

Model-driven Engineering(MDE) offers an alternative that allows to reduce time-to-market and development costs.

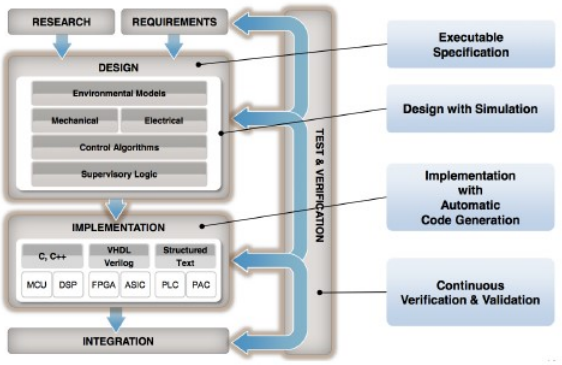


Figure 1.2: Model-driven engineering design flow

Principles of MDE are:

- *Abstraction* from specific realization technologies
  - A model should care about system's behaviour not its implementation
  - Requires modelling languages that do not hold specific concepts of realization technlogies
- Automated *code generation* from abstract models
- Separate development of application and infrastructure

According to this approach, there will be a software tool to translate a project in software. Usually the project consists of one or more models. A **model** is a representation of the original system that includes only the key features useful to understand its behaviour. *Modelling* is the key of this approach. The functionality of the application we want to develop will be modelled using a language that is abstract enough to allow to work without specifying too much details (e.g. Simulink and Stateflow). A transformation tool will automatically generate the code relative to the model, applying to it some transformations defined through a specification

language. The advantage of this approach is that the model can be continuously refined throughout the whole development process and it can be simulated at any time to see how the system behaves. Multiple scenarios can be tested without any risk and without using specific and expensive machinery.

The typical design flow of model-based software design is the so-called *"V-Shaped"* development flow, shown in the picture below.

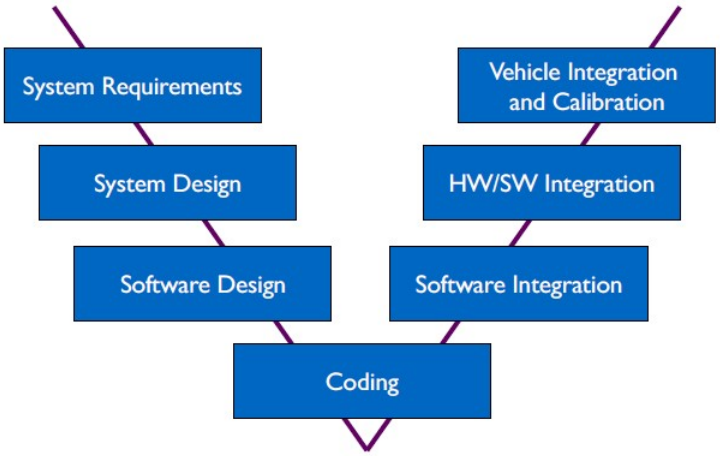
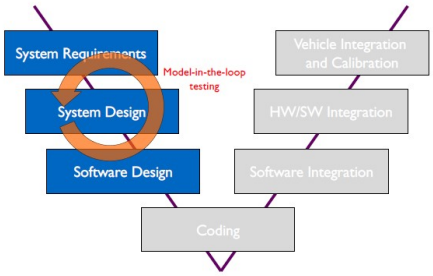


Figure 1.3: MBSD V-shaped development flow

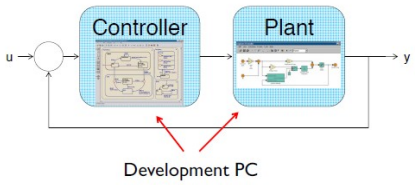
It is possible to perform simulation during the whole process in order to avoid unwanted behaviours at the end. The best thing is to perform simulations to verify each step to avoid going back of several steps if an error occurs. Different types of test can be performed: Model-in-the-loop (MIL), Software-in-the-loop (SIL), Processor-in-the-loop (PIL) and Hardware-in-the-loop (HIL).

Model-in-the-loop testing

It can be performed in the first steps of the V-Shaped design flow when both the plant (system to be controlled) and controller (algorithm controlling the plant) are modelled. It is performed on a PC. Models are developed using a suitable domain-specific language (e.g. Simulink/Stateflow). The simulation is used to validate the correct behaviour of the controller looking only at the functional aspect. For example, we can test if the plant is able to follow a given input.



(a) MIL in V-shaped

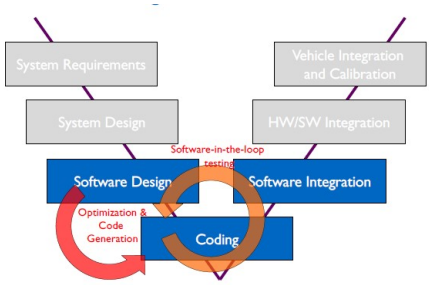


(b) MIL scheme

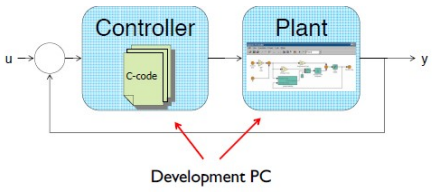
Figure 1.4: Model-in-the-loop testing

Code generation and Software-in-the-loop testing

When MIL tests give acceptable results, the model of the controller is translated into code by means of the so-called code generation. It implies the application of some transformations defined by trough a specification language. For example, it allows to transform a Simulink model into C code. Software-in-the-loop is the step done after code generation to validate the correct behaviour of the software resulting from the controller model. The software is co-simulated with the plant model, by executing both on the development PC. The goal is to evaluate the behaviour of the code resulting from the controller model, looking again at its functional aspect.



(a) SIL in V-shaped



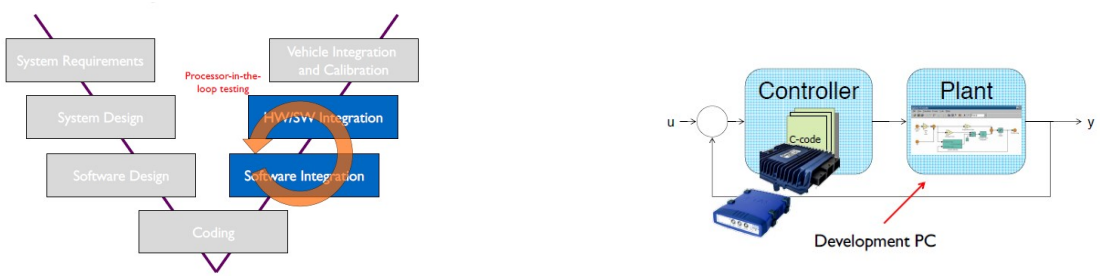
(b) SIL scheme

Figure 1.5: Software-in-the-loop testing

Processor-in-the-loop testing

After validating the software, the successive step is the Processor-in-the-loop test. It is performed using different machines. The model of the control algorithm is translated into code and deployed on an embedded processor (e.g. an evaluation

board or an ECU). It is then executed in combination with a model of the plant running on a simulation environment into the development PC. In this way, the designer can verify the correct operations of the code implementing the controller while running on the hardware that will be used in the final product or one close to it. In this kind of test we are interested in the functional aspect of the controller too, still neglecting the real time behaviour.



(a) PIL in V-shaped

(b) PIL scheme

Figure 1.6: Processor-in-the-loop testing

Hardware-in-the-loop testing

The last type of test is called Hardware-in-the-loop (HIL). It is performed using different machines. It consists in running a software implementation of a control algorithm in a microprocessor-based system, for example an evaluation board or the target hardware, in combination with a real-time plant model that is executed by a real-time computer. The input/output connections between the two elements are implemented using the very same connections that will be used in the real application. Some hardware components belonging to the physical plant may also be connected. In this stage, validation of the software is done in real-time, consider both the functional and timing properties of the software.

(a) HIL in V-shaped

(b) HIL scheme

Figure 1.7: Hardware-in-the-loop testing

1.4 Goals and used tools

The thesis work aims to validate the models, received from and designed by aeronautical engineers, that are part of the Iron Bird. This implies deploying them on the development PC, testing and improving them until complete validation. In particular, it is about performing SIL tests using NI VeriStand.

MATLAB Simulink

According to MBSE the key concept is the *model* of the functionality we want to design. A model reflects the relevant section of the original system properties, so focusing on important properties only. Modeling requires to use a language that is abstract enough to allow the designer not specifying too much details. The used language is Simulink.

Simulink is a MATLAB-based graphical programming environment for modeling, simulating and analysing multidomain dynamical systems(Wikipedia, n.d.). Simulating block diagrams, it allows to understand and analyse complex systems and implement them with high quality. It provides tools useful to improve design efficiency.

Starting from a model designed in Simulink is possible to generate code to prototype, test in real time and deploy onto an embedded system. Throughout the whole development process, the designer can continuously verify and validate the design.

NI VeriStand

To perform SIL test and consider also the real time aspect of the simulation, a different simulation environment is required. The chosen one is *NI VeriStand* that is designed also for HIL tests.

VeriStand is a software for real-time test applications, such as stimulus generation, data acquisition, and calculated and custom channel scaling (National Instruments, n.d.).

This software helps in configuring I/O channels, data logging, stimulus generation. It is possible also to import simulation models and control algorithms. With VeriStand is possible interact with and monitor the application using a run-rime editable user interface. Its usage doesn't require coding knowledge but it is possible to use several software environments (ANSI C/C++, Python, ...) to add custom functionality. VeriStand helps test engineers reducing time needed to test their products.

Chapter 2

Iron Bird Architecture

Investigating and verifying the effectiveness of load alleviation, load control technologies and electrical actuation requires the design and construction of an integration test rig, as we already said. Develop that advanced multi-functional integration, verification and validation test bench is one of the main objectives of ASTIB and the main point of this thesis. The test rig is known as *Iron Bird* (IB) and CERTIA is in charge of its design as well as its production.

Considering the IB, ASTIB objectives are:

1. Design, development and construction of a specif infrastructure able to:
    - Allow testing of new flight control architectures
    - Accept the installation and allow testing of the electrical power distribution system
    - Reproduce the aircraft installation and allow testing of the electrical landing gear system
    - Reproduce the aircraft installation of a few selected flight control surface
    - Install programmable load banks simulating aircraft electrical loads in order to complete the global aircraft electrical system simulation
2. Implement a simplified aircraft model capable to verify system's behaviour in a simulated flight condition by allowing:
    - Simulation of aircraft mechanics

- Simulation of the sensors
- Real-time computation of aerodynamic loads
- Simplified aeroelastic model of the aircraft
- Interfacing with the flight control computer

3. Implement real-time features to enhance the Iron Bird functionalities as:
   - Generation of aerodynamic loads acting on the flight control surfaces and on the landing gears
   - Mathematical models of the flight control actuators
   - Virtual wing computer that will simulate in real-time the same actual equipment installed on the IB
4. Enable faults simulation in the components under test
5. Support the testing activities
6. Provide a platform suitable for future installation of other components of the flight control system

The Iron Bird will be used to verify and validate the functionality and performances of significant equipment of the flight control system; it will be developed for permitting verification of this equipment under simulated failure conditions, thereby supporting the aircraft integration and certification process (Andrea Dellacasa, 2017).

It will allow to simulate a wing of a regional airplane. The simulator is partially physical and partially virtual. Indeed, the Iron Bird is equipped with half real aircraft system,that is the left semi wing, while the other half, the right semi wing, will be simulated by a specific module.

Two configurations are possible: with winglet (WL) installed or the wingtip (WT) installed.

The following picture shows the architecture of the Iron Bird.

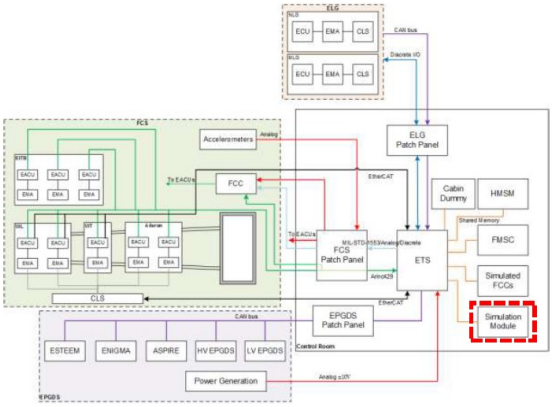


Figure 2.1: Iron Bird Architecture

In particular, figure 2.1 highlights modules and units composing the test rig and how they shall communicate with each other exploiting the shared memory.

As *Politecnico di Torino*, our job is about the real-time conversion of the **Flight Mechanics Simulation Computer**(FMSC), the **Flight Control Computer**(FCC) and the **Health Management System Module**(HMSM). Each of them consists in a Simulink model, designed by aeronautical and mechanical engineers, and deployed on a PC. The red-dotted module is the *Actuator Simulation Module (ASM)* and it shall simulate the response from all those modules.

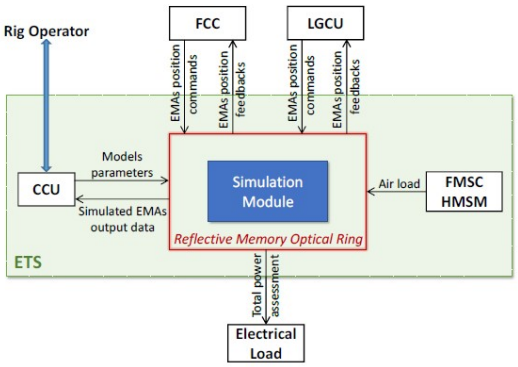


Figure 2.2: ASM communication diagram

2.1 Actuator Simulation Module

The ASM is a part of the Iron Bird, designed to:

- Verify and validate aircraft systems technologies up to TRL 5
- Support the Flight Control System (FCS) Load Control/Load Alleviation system design
- Enable inter-system integration and verification
- Support the the achievement of the FCS permit-to-fly for the demo flight configuration of FTB#1

Thus, the main goal of ASM is to simulate in real-time actuators for the right wing, the right main landing gear and tail actuators to get a full aircraft configuration for ground testing. The second target is to enable tests to be executed even if a real actuator is missing. In this case it will be replaced by its real-time model.

ASM will not interact directly with FCCs, FMSC and HMSM; all data shall pass through the *Engineering Test Station*(ETS) and be stored in the reflective memory optical ring. Simulation module will get inputs from the reflective memory optical ring, coming from:

- ETS for the models parameters
- FMSC for the air load
- FCCs for electromechanical actuators (EMAs) position commands
- Electronic Control Unit (ECU) for EMAs commands

ASM output will be copied into the reflective memory optical ring toward FCCs, ECU and ETS as shown in figure 2.2.

Simulation module is designed to be able to work in two operating modes:

- An **Iron Bird Mode**, where all electromechanical actuators can operate. This corresponds to the main mode of operation, representative of a full aircraft

- A **standalone mode** in which ASM can be controlled by the Central Control Unit (CCU) to simulate one or several actuators' model without receiving signals from the others. This allows to check and evaluate the performance of a given EMA once set the model parameters.

2.2 Flight Mechanics Simulation Computer

The *Flight Mechanics Simulation Computer* (FMSC) is the part of the IB in charge of simulating the flight dynamics. Its main target is to offer real-time simulation, on both longitudinal and lateral directional plane, with the purpose of checking the equipment behaviour in a simulated real-time context. The model depends on the aircraft selected for experiments, which will be a turbo-prop with 90 seats (TP90).

The Simulink model consists of several subsystems, including one representing the rigid model mechanics; one for the flexible body mechanics; one for computation of hinge moments and one representing the sensors.

Inputs to FMSC are: elevator deflection δ_e, rudder deflection δ_r, right aileron deflection δ_{a_r}, left aileron deflection δ_{a_l}, and winglet deflection δ_w or wingtip deflection δ_{wtip}. A pecentage of throttle variation is also included.

Every block and subsystem are validated using a fixed-step integrator (Runge-Kutta) with a sample time $T_s = 0.01s$.

Aeroelastic Model

The aeroelastic model of the aircraft is also taken into account in FMSC. It is developed under the following assumptions:

- Inertial tensor in the body reference frame is considered diagonal
- The only deformable element for this 6 DoF aircraft model is the wing
- the flexible components and deformations are modelled considering only the 1^{st} and 2^{nd} symmetric bending modes and the 1^{st} symmetric torsional mode
- the A/C weight is constant in time

- the sweep angle $\Lambda = 0$

Strip theory is used for modelling aerodynamic forces and moments. It allows to describe them as a function of the direction of the local flow only, and to reduce complexity and computational cost.

Rigid Body Mechanics

To describe the rigid body mechanics nonlinear equations of motion are considered. The rigid model has as inputs the step variation of the formerly listed inputs together with initial trim conditions.

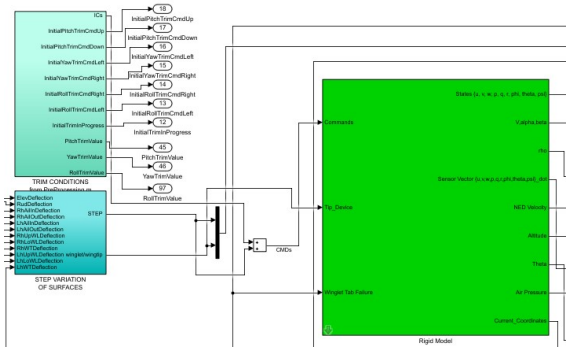


Figure 2.3: FMSC Rigid Model

The computed outputs are:

- the state vector $x = \{u, v, w, p, q, r, \phi, \theta, \psi\}$, where u,v,w are the linear velocity components of the A/C, p,q,r the angular rates and ϕ, θ, ψ the Euler angles.
- an auxiliary vector that collects V,α and β, where V is A/C speed, α the angle of attack and β the sideslip angle.
- the air density ρ
- the velocity vector in NED reference frame
- the accelerations of the state vector for the sensor model
- air pressure

- current coordinates, i.e. longitude and latitude

Flexible Body Mechanics

The successive block of the 6 DoF simulator implemented by the model, represents the flexible body mechanics.It is linked to the rigid model by means of the quantities listed before.

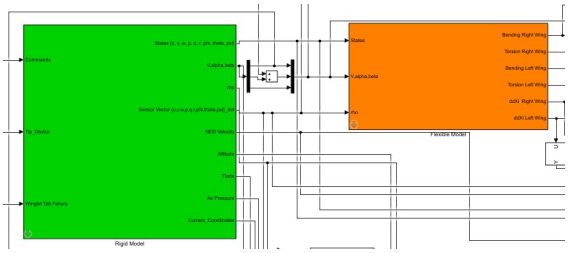


Figure 2.4: Rigid and flexible models

The flexible model is designed according to a Lagrangian approach and treats the wing as a deformable element, while tail and fuselage are taken as rigid parts. The deformation state of each flexible element is described following the Galërkin method, that allows to approximate the progress of the deformation with truncated series expansions.

Gust Simulation

A gust of wind produce a change in system aerodynamics, giving rise to an induced incidence on the wing for every node of the aerodynamic model. In the model, the gust effect is represent by a constant input that is true if considered, false otherwise.

Hinge Moments

Hinge moment is the technical name for the force required to deflect a control surface. It grows rapidly as control surface get larger and speed get higher (Garrison, 2018).

A dedicated subsystem, connected directly with the rigid model (green block), is in charge of calculating the hinge moments acting on control surfaces. This block takes as inputs velocity of the aircraft, wingtip, winglet and aileron deflections (CMDs in the model), and air density and returns the corresponding hinge moments.

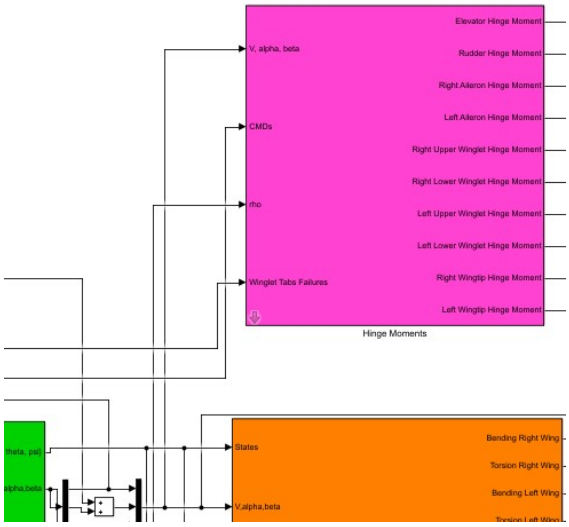


Figure 2.5: Hinge Moments block

Sensor Model

Finally, a subsystem models the sensors installed on the aircraft. There are three accelerometers on each half-wing, an inertial measurement unit (IMU) and a global positioning system (GPS).

Subsystems for the previously mentioned sensors are present inside the blue block. Every half-wing has a subsystem which contains three blocks, one for each simulated accelerometer.

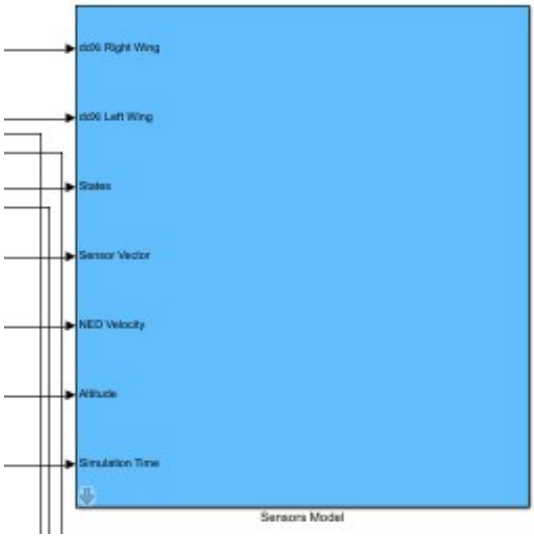


Figure 2.6: Sensor Model

2.3 Flight Control System

The *Flight Control System* (FCS) is the part of the IB in charge of controlling the airplane during the flight. It includes the Flight Control Computer (FCC), the Remote Interface Unit (RIU) and some other subsystems representing Winglet and Wingtip that refers to the Electronic Actuation Control Unit (EACU).

FCS is designed as a unique big Simulink model, simulated with a fixed-step discrete solver and a step time of $10ms$ set at model level.

The model provides five types of functionalities:

- *Interface Management*: subsystems representative of FCCs, RIUs and Actuation are connected each other through A429 ports. This functionality is used only in the Simulink model.
- *Simulation*: this provides the behavioral model of the real equipment
- *Stimulation - Type I*, modeled internally to FCCs to animate the model with the expected outputs
- *Stimulation - Type II*, used to feed the actuation subsystem through a kind of failure/event injection

- *Stimulation - Type III*, used to feed FCS model with data from the Cabin Dummy and from the A/C accelerometers.

It covers both the flying configurations of the Iron Bird, with either the winglet (WL) installed or the wingtip (WT) installed.

The Simulink representation is made up of several subsystems:

- 3 FCC subsystems, standing for FCC A, FCC B and FCC C
- 3 RIU subsytems, for RIU A, RIU B and RIU C.
- 4 Aileron subsystems (LH Outer, LH Inner, RH Inner, RH Outer), 2 Wingtip (LH and RH), and 4 Winglet (LH Upper, LH Lower, RH Upper, RH Lower).
- 3 subsystems generating the different types of stimuli.

Flight Control Computer

There are three different FCCs, FCC A, FCC B and FCC C. They are fed by type I stimuli to generate almost all the outputs. They interface also with trim and failure reset commands given by type III stimuli.

From a lower level point of view, the work done by these subsystems is to group the signals needed for WL/WT and Inner/Outer Aileron respectively and forward them to the subsystems representing that actuators.

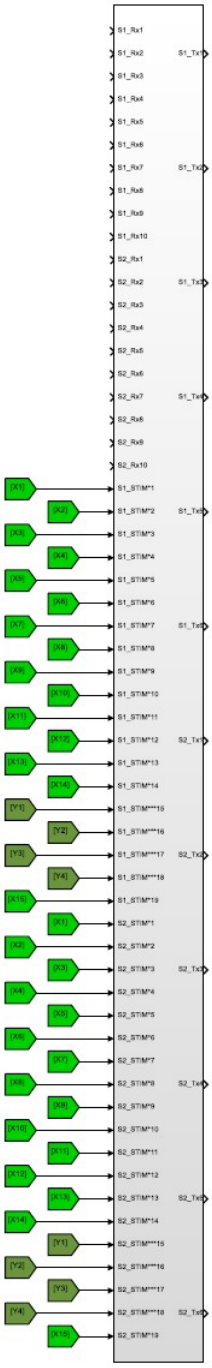


Figure 2.7: FCC subsystem

Remote Interface Unit

There are three blocks representing the Remote Interface Unit: RIU A, RIU B and RIU C.

They provide re-routing of messages from the input ports to the output ports, implementing so the Interface Management functionality.

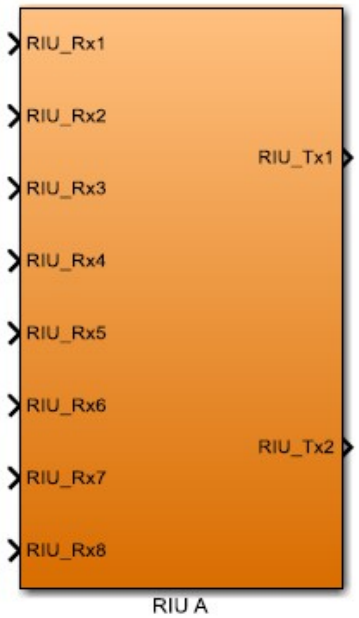


Figure 2.8: RIU subsystem

Aileron subsystem ports

This represents one of the actuation systems mounted on the A/C. In the model we have 4 subsystems referencing to the Aileron: LH Outer, LH Inner, RH Outer and RH Inner Aileron.

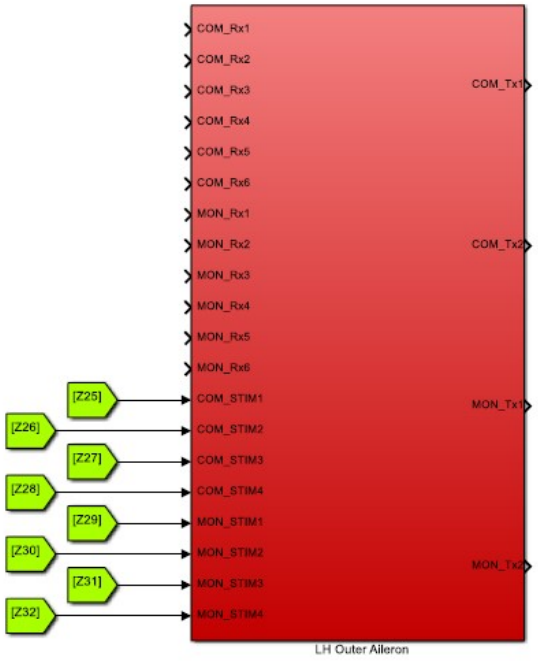


Figure 2.9: Aileron subsystem

Wingtip subsystem ports

Another actuation system. There are two wingtip subsystems: LH and RH wingtip.

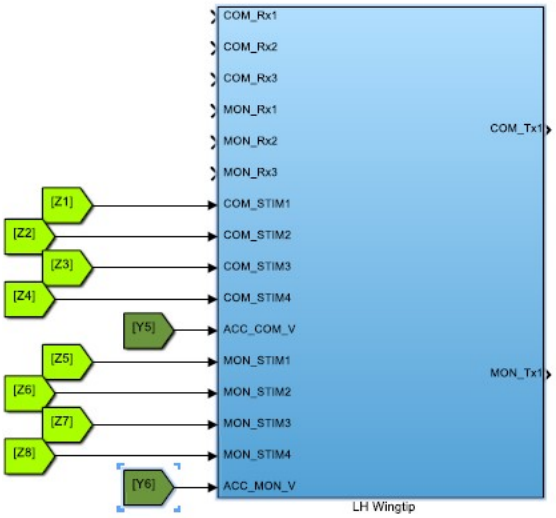


Figure 2.10: Wingtip subsystem

Winglet subsystem ports

Another actuation system. For the winglet there are 4 subsystems: LH Upper, LH Lower, RH Upper and RH Lower.

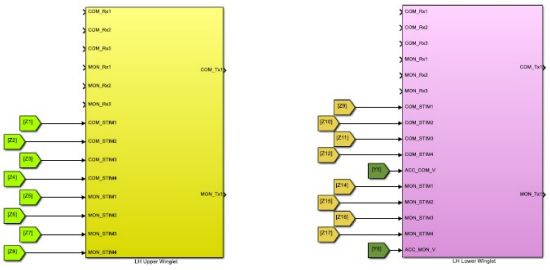


Figure 2.11: Winglet subsystem

2.4 Health Management System Module

ASTIB project will also contribute to the development of a health monitoring validation platform and the tools for the evaluation of the benefits of such integrated platform. Its goal is to collect and process data provided by the sensors mounted on the aircraft structure and systems such to evaluate the aircraft health status.

The **Health Management System Module** (HMSM) will give an advanced integrated reasoning toolset that incorporates justified levels of automated fault accommodation based on prognostic information for enhanced vehicle safety and decision support.

The status of the system is continually monitored with sensors. Data are collected and analysed. The first phase consists in a pre-processing of the data (outliers removal, transformation, ...). Then they are used within a diagnostic algorithm; anomalies are reported when there is a change from a healthy state and the root identified.

Finally, a prognostic algorithm is used to determine how much remaining life the component/system has. This time is also used for on-board tactical control or off-board strategic planning, giving the decision maker enough time to manage the health of the system and take the most appropriate action prior to the failure.

Health Management System Module will perform three key functions:

- Allow injection of simulated degradations and their progression to verify the merits of Prognostics and Health Management (PHM) algorithms
- Receive and store actuators measured data
- Implement prognostics and health management functions

As can be seen in figure 2.12, the user can choose which electromechanical actuator mathematical model consider and the fault to be implemented. This permits to perform HIL tests and studying growth of a specific fault for a single actuation system.

During normal operations of the Iron Bird, HMSM will receive data from both simulated and real EMAs and some additional variables useful for PHM analysis.

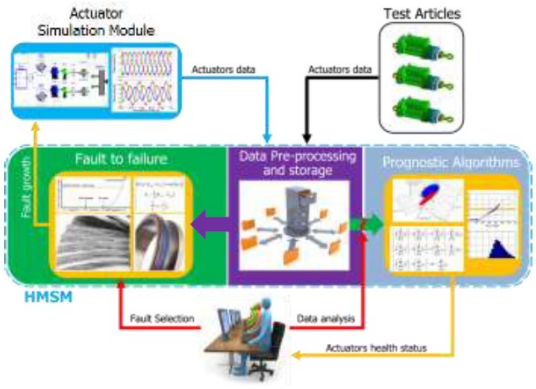


Figure 2.12: HMSM Architecture

These data can be saved by the user and retrieved to execute PHM algorithms that will use them to provide the EMAs health status.Data can be eventually pre-processed to reduce the dimension of the file containing them.

Since prognostic functions do not need to run in real-time, the prognostics functions will run on a common hardware and can be launched also in an external computers.

This module can, then, be divided into two part:

- An offline one, meaning it won't be running in real-time. It aims to run PHM algorithms and to show the pilot the system status. It consists, indeed, in a graphical user interface designed and developed in Matlab, shown below.

Figure 2.13: HMSM Graphical User Interface

It can be seen as divided in two main groups: *Main* and *Documentation*. The second gives a short guide about how to interact with the software. The 'Main' tab consists of two sections, one from which the user can select the inputs and another used by the software to graphically show the results. The 'DATA IMPORT' panel is used to load the data set from a file, of which the name must be specified. The 'RUN' button let PHM algorithm start.

The visual feedback is provided by three graphical tabs which are 'Features behaviour','Fault Detection' and 'Prognosis'. The first and the second show graphical informations about the selected features. In particular they show their trend over test duration and their pdfs as histograms, respectively. The last tab depict the estimated Remaing Useful Lifetime (RUL).

- An online one, that will interface with the optical ring and the ETS.

  This part is responsible for fault injection and data storage. It consists, like FMSC and FCS, in a Simulink model that will be converted in real time. The model can be described as an acquisition system that will propagate data from the Iron Bird to the offline part of HMSM. It has to be specified that only the right semi-wing is represented, since only this part is simulated by the ASM. Data from real actuators, also needed for PHM analysis, will be provided by a dedicated acquisition box.

Like other components in the Iron Bird, also HMSM is interfaced with them and with the ETS by means of a reflective memory card.

The described functionalities allow to identify three operating modes for this module:

1. *HIL tests*: in this mode the user can select an actuator model, a specific fault and a fault-to-failure trend to create a data set to be analysed by PHM algorithm

2. *Passive Mode*: in which data from tests are periodically stored and analysed to evaluate the status of real equipment

3. *Active Mode*: it is possible to inject a fault into mathematical model of the chosen EMA while normal IB tests are running

2.5 Reflective Memory and Synchronization

The Iron Bird can be correctly considered a multiprocessors system, since it consists of several PCs. Hence, a way to make them capable to communicate with each other is needed.

Two types of multiprocessors systems can be distinguished according to the way they share data and informations: shared-memory systems and distributed-memory systems. In the first case, we have multiple CPUs and a single global physical memory while in the latter each processing node has its local memory. A third approach exists, it is called Distributed-Shared-Memory (DSM) and combines the advantages of the previous. In words, a DSM system logically implements the shared-memory model on physically distributed-memory design (Baek, 2002).

A Reflective Memory network is a special type of shared memory system which enables multiple, separate computers to share a common set of data. Reflective Memory networks maintain an independent copy of the entire shared memory set in each attached system. Extremely low data latency is the key benefit of a high-speed, hardware-driven network like Reflective Memory. This low-latency performance is of paramount importance when building real-time systems such as simulators (ABACO Systems, n.d.).

The key feature of a Reflective Memory System (RMS) is, indeed, that each computer physically has its own local memory. Memory updates can occur over different types of interconnections. As example we cite the RT-CRM, Real-Time and Channel-based Reflective Memory that is based on memory channel, i.e. hardware assisted, virtual connection-based memory to memory transfer of data (Chia Shen, 2001). It represents an approach similar to the one used in this project. Indeed, as we will see later, the reflective memory used is organized in channels and each of them will be reserved for each input or output signal involved in the project.

The Iron Bird software simulations need to be synchronized, in order to achieve flight tests integrity and to ensure that all data have been produced during the same cycle. The used algorithm is based on reflective memory events, in particular an handshake between a master scheduler (the ETS) and all the involved software modules. The master scheduler receives commands from the test rig operator and, exploiting the Iron Bird optical ring, it pilots the following states:

- *Initialization*: executed once when 'Start' command is received and is reserved for the modules initialization and configuration
- *Execution*: executed iteratively until 'Stop' command is received from the rig operator. It is composed of the following sub-states:
  - Write cycle: software modules copy their output data into the optical ring, with the purpose of sharing them for the next execution cycle. In this phase ETS takes the needed data from the physical interfaces and copy them into reflective memory area
  - RT cycle: the software modules execute their specific algorithm and tasks
  - Read cycle: the software modules retrieve from reflective memory area the input data necessary for their execution. ETS converts those data into the physical interfaces.
- *Stop*: executed once when 'Stop' command is received to stop the current test and software modules execution.

For handshake communication between scheduler and nodes, two reflective memory events are used:

- Interrupt#1, sent by the scheduler to all the nodes to command the current state
- Interrupt#2, sent by every node as acknowledge

An enumerative value is used as parameter of the previous events:

- 1000: Inizialization
- 1001: Read
- 1002: Write
- 1003: Stop
- 2000: Acknowledge

Before going on with the next phase, the master scheduler have to collect all the acknowledge messages to ensure all the modules finished computations.

The master scheduler is able to check and notify in the ETS GUI LogSystem the following error conditions:

- any software module execution overrun detection, compared to real time cycle duration

- any missing acknowledge from modules

The ETS keeps track of number of consecutive overruns. If a single system runs late for N consecutive cycle the entire test sequence must be stopped.

The following image provides a sequence diagram of the described algorithm:

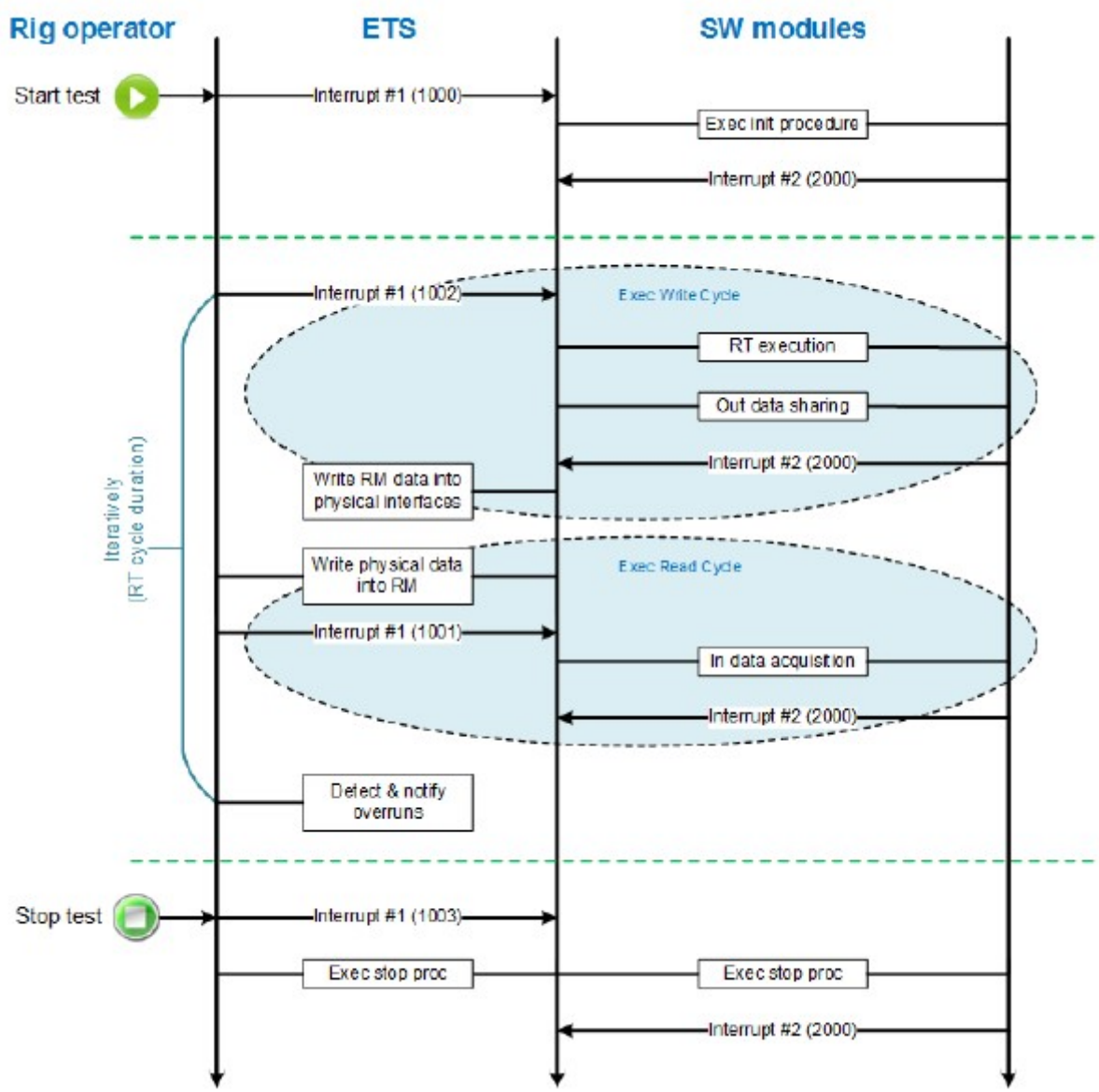


Figure 2.14: Synchronization algorithm

Chapter 3

Real-time conversion of Flight Control System

The task assigned to our team is about real-time conversion of the three models described in the previous chapter and then mapping their inputs and outputs on the reflective memory optical ring. This last operation is needed to make FMSC, FCS and HMSM able to communicate with each other and with the ETS.

In order to run a Simulink model in real-time, a common way is to deploy it on a different simulation environment capable of considering also the timing properties. NI VeriStand is a software environment with these features and the one chosen for our purposes. Importing a model in VeriStand requires a dll (dynamic linking library) file, in other words a file describing the model in a way understandable to NI software. Such a file is obtained as result of code generation, performed by mean of Simulink embedded coder.

3.1 Code Generation

To be translated into code, a model shall be suitable for code generation. By the way, in most cases this is not a verified condition and the model needs to be modified in its design such as in its settings.

Refactoring is needed if blocks such as "Step", "Ramp" or other from Source library, as well as "Scope" from Sinks library are present. These blocks must be removed and substituted with *NI VeriStand Inports* or standard Input Ports, and

NI VeriStand Outports or standard Output Ports respectively, otherwise VeriStand won't be able to correctly recognize inputs and outputs from the dll.

The following figure shows an example of the previous:

Figure 3.1: Refactoring example

Since the model we received from the designers was not compliant with the above statement, we firstly had to modify it and make it similar to the right side of figure 3.1.

From a practical point of view, our goal is to generate a dll file for each subsystem composing the model, load that file in VeriStand and map inputs and outputs on the reflective memory. This requires each subsystem in the model to be extracted from it and treated separately.

Let's consider as an example the subsystem representing the LH Inner Aileron. We put it in a single Simulink file and try to generate the dll file describing it.

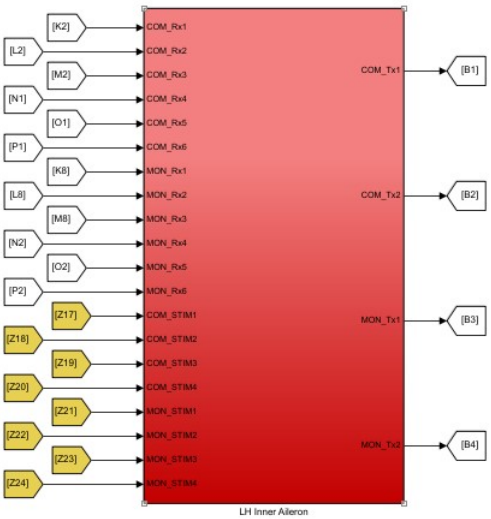


Figure 3.2: Original LH Inner Aileron

In order to perform code generation correctly we have to check some settings.

- In solver section, we usually put type=fixed-step and solver=discrete. The fixed-step size is a number chosen according to the time requirements. We read from requirements that is 10 ms. Since the application will run continuously, we also set *inf* as stop time.

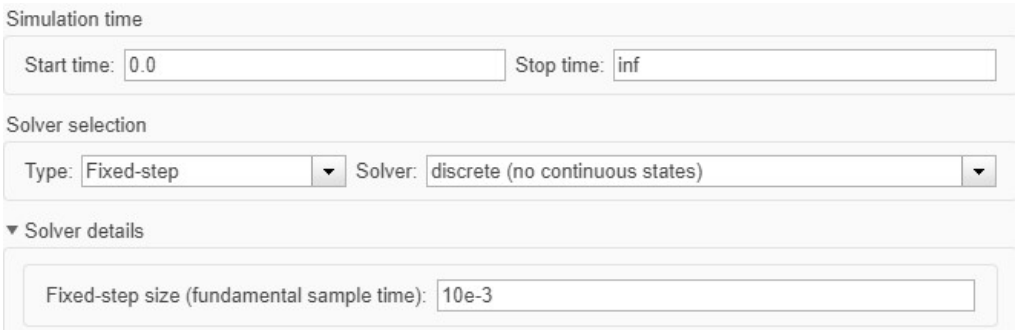


Figure 3.3: Solver settings

- In hardware implementation tab, we put Intel as device vendor and x86/Pentium as device type since we know our files will run on devices with these features.

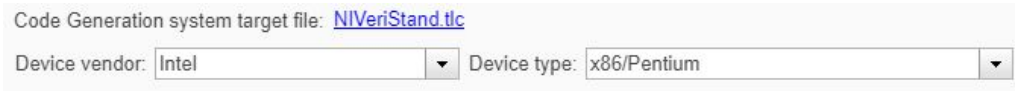


Figure 3.4: Hardware implementation settings

- In code generation tab, we have to select the needed Target Language Compiler, i.e. one of the tlc files that Simulink suggests. Since we'll use the dll inside NI VeriStand, we must choose NIVeriStand.tlc as system target file.

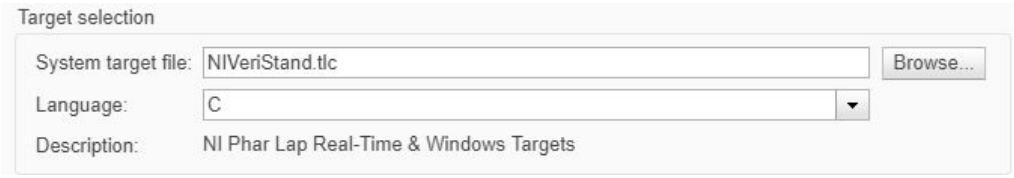


Figure 3.5: Code Generation settings

Now that settings are correctly fixed, we can build the model using Simulink Embedded Coder. It can be launched by clicking the corresponding button in the toolbar or with the keyboard shortcut CTRL+B.

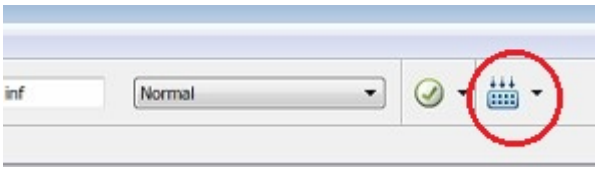


Figure 3.6: Simulink build button

The process is automatic and can be followed opening the Diagnostic Viewer window. If there are no errors, at the end of the process a folder will be created. It is named according to this pattern

modelName_targetFile_rtw

where:

- *modelName* is the name of the Simulink file without its extension (.slx)
- targetFile is the name of the tlc file we selected in code generation tab
- rtw stands for Real-Time Workshop and indicates that we want C code to be generated

Since we named the model *LH_Aileron_Inner_EACU_v0_2016b* and we chose *NIVeriStand.tlc* as system target file, the new folder will be named:

LH_Aileron_Inner_EACU_v0_2016b_niVeriStand_rtw

The just created folder contains the C files (*.c and *.h) relatives to the model together with a dll file. As mentioned before, this last file is the one we are interested in. Next step is to import the model in NI VeriStand.

3.2 Import in VeriStand

The project in VeriStand is described by several files. To add the model we have to open the system definition file, the one with extension *.nivssdf*. A window like this will open:

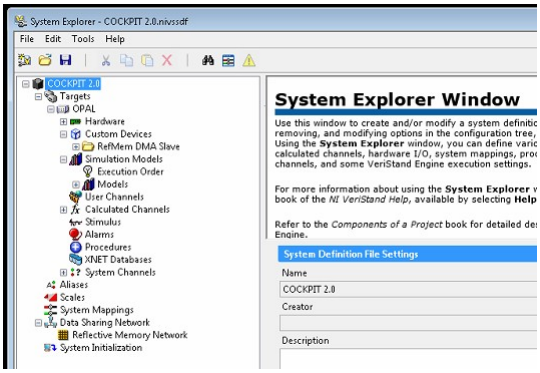


Figure 3.7: System definition explorer

To add the model, we have to copy it in the 'Model' folder of our NI VeriStand project. To achieve this, we click on 'Models' in the tree on the left side of previous window and then on 'Add a Simulation Model' button that appears on the toolbar.

Figure 3.8: NIVS Add Simulation Model button

On clicking it, this dialog will open:

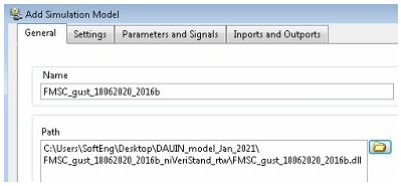


Figure 3.9: NIVS Add Simulation Model path

Now we have only to give a name for the model we want to add and specify the absolute path to the dll file in the file system. It is also possible to manage the decimation regards the main frequency and the core of processor we want to use.

After this step, we're able to run the project, with a complete access to inputs, outputs, and parameters of the model from NIVS interface. The first check we do is about the correct identification of inputs and outputs. We look at the Inports and Outports folders under the model we just added.

Figure 3.10: Empty Inports and Outports

In figure 3.10, we can see the above mentioned folders are empty. That's mean that something is wrong with the model.

As can be seen in figure 3.2, the model received from designers appears with a lot of "From" blocks as inputs and "Go to" blocks as outputs. Using them VeriStand is not able to recognize any input and/or output.

Hence, we have to remove all "From" and "Go to" blocks and replace them with NI VeriStand Inport and NI VeriStand Outport respectively. They can be found in the "NI VeriStand Blocks" folder of Simulink Library Browser, that will automatically appear once completed the installation of NI software.

An important aspect is naming of such elements in the model. In order to make it understandable for other people working on it and make it capable of communicating with the other models composing the Iron Bird, each port shall have a specific name indicated in the Interface Communication Document (ICD) given to us as an excel file.

Once added the NI ports and renamed them according to the ICD, we have to repeat the code generation process and import the new dll in VeriStand.

This is how all inputs and outputs of the subsystem should look before code generation:

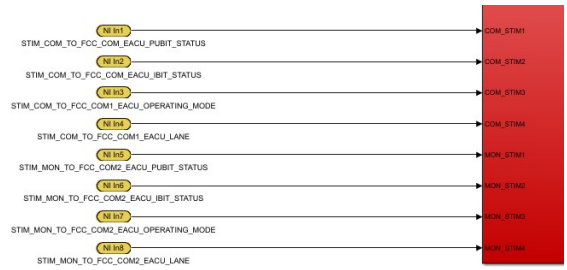


Figure 3.11: LH Inner Aileron Inputs

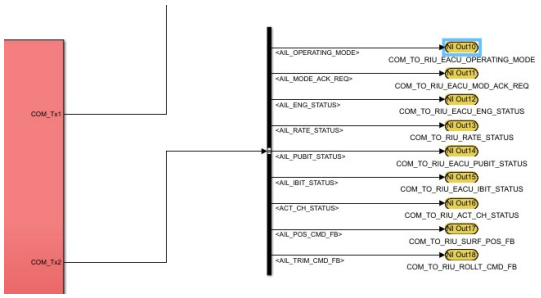


Figure 3.12: LH Inner Aileron Outputs

And this is a part of what we should see in Inports and outports folders in NIVS System Explorer Window:

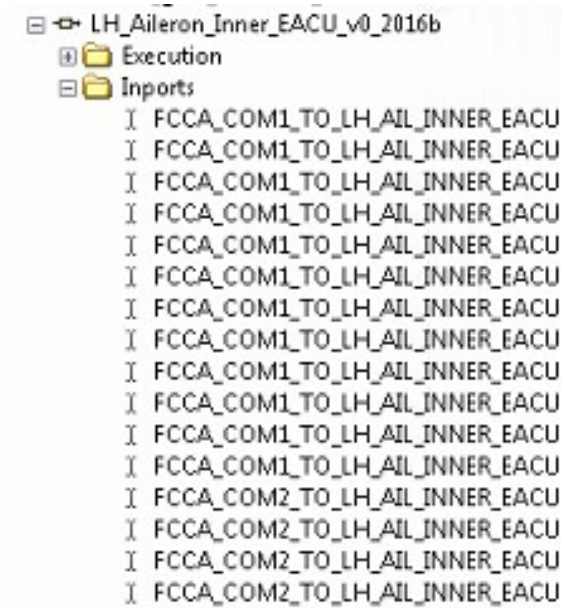


Figure 3.13: LH Inner Aileron NIVS inports

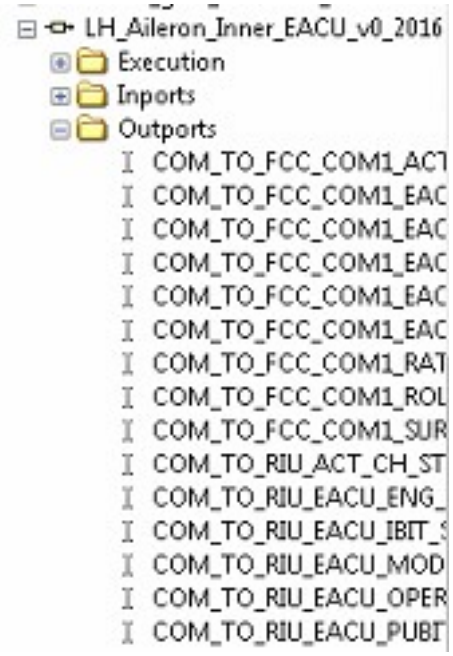


Figure 3.14: LH Inner Aileron NIVS outports

We repeated the same process for all the subsystems composing the model received from *Leonardo S.p.A* to obtain the following dll files:

- FCCA_Sim_v0_2016b.dll
- FCCB_Sim_v0_2016b.dll
- FCCC_Sim_v0_2016b.dll
- LH_Aileron_Inner_EACU_v0_2016b.dll
- LH_Aileron_Outer_EACU_v0_2016b.dll
- LH_Winglet_Lower_EACU_2016b.dll
- LH_Winglet_Upper_EACU_2016b.dll
- LH_Wingtip_EACU_2016b.dll
- RH_Aileron_Inner_EACU_v0_2016b.dll
- RH_Aileron_Outer_EACU_v0_2016b.dll
- RH_Winglet_Lower_EACU_2016b.dll
- RH_Winglet_Upper_EACU_2016b.dll
- RH_Wingtip_EACU_2016b.dll
- RIUA_Sim_v0_2016b.dll
- RIUB_Sim_v0_2016b.dll
- RIUC_Sim_v0_2016b.dll

where '2016b' indicates the version of MATLAB we used throughout the whole project.

Once imported all of them in VeriStand, this is how 'Models' folder in systen definition file looks like:

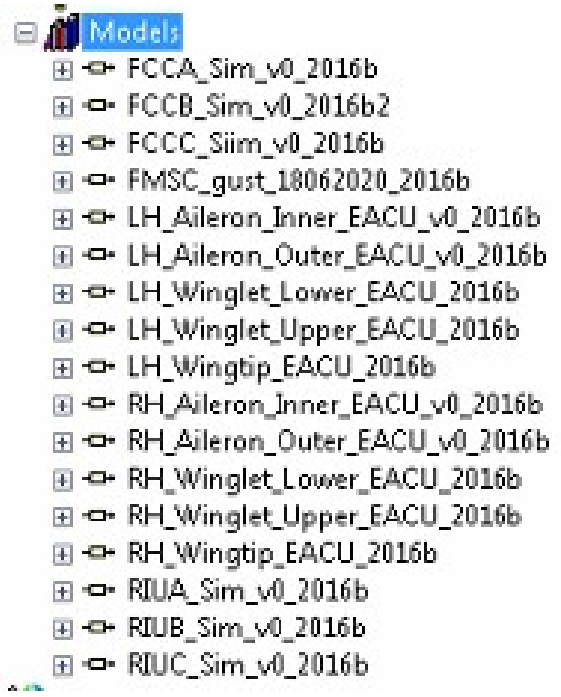


Figure 3.15: NIVS Models Folder

3.3 Mapping on Reflective Memory

When the names of all input ports and output ports are compliant with the names in ICD documents, the model has the right features to communicate with the other parts of the Iron Bird but it still is not able to do it. To communicate with the Actuator Simulation Model (ASM), Flight Mechanics Simulation Computer (FMSC), Health Management System Module and other parts of the project, each subsystem of the model shall know where to find the necessary data for its computations and where to write the produced results.

According to project requirements, these operations has to be done exploiting the reflective memory optical ring. Hence the subsequent step is to map each input and output on it.

First of all, for compatibility reasons we have to rename our controller in Target folder as **OPAL**. In this way our partners can easily merge our work with theirs.

Then, we need to add the reflective memory to our project in VeriStand System Explorer. We received the representation of the memory from CERTIA, the French company involved in this project. It consists of several files that we need to add all to the system definition file tree.

Figure 3.16: Reflective Memory files

We have to copy those files in a folder that we called *RefMem DMA Slave* and move it to the folder 'National Instruments\NI VeriStand 2017\Custom Devices'. Now we can add it to the project, right-clicking **Custom Devices** in the tree and selecting RefMem DMA Slave from the drop down menu.

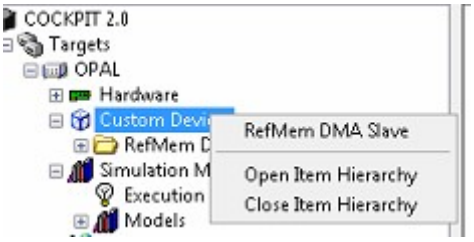


Figure 3.17: NIVS Custom Devices

In our custom device, we have to add 2 blocks. For adding blocks we have to select the reflective memory and click on 'Add DMA Block' button on the right-side of the system explorer that automatically add a folder in the custom device hierarchy.

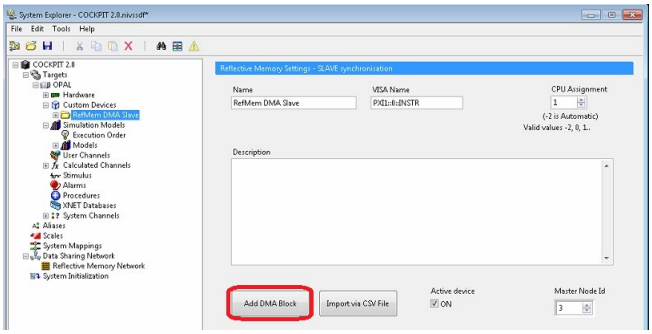


Figure 3.18: Custom Devices Add Blocks

In the screen that appears next, we can set the features of each block. We can give a name, specify an address and the type (i.e. read or write). Beside this we can add the needed number of data channels.

The first block, that we called Block1, should have these properties:

- "Read" mode
- 2116 data channels
- Block address = 1

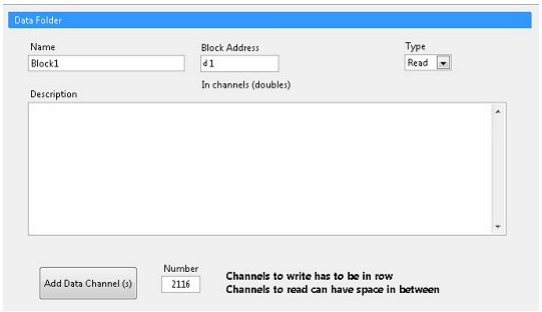


Figure 3.19: DMA Block 1

The other block, Block2, should have instead these properties:

- "Write" mode
- 1443 data channels
- Block address = 430

Figure 3.20: DMA Block 2

While Block1 will be used for mapping the inputs of each model, Block2 will be used for the outputs. At this point the reflective memory is properly added to the project and we can start mapping inputs and outputs.

Let's open the *Configure Mappings* utility from NI VeriStand System explorer, clicking the corresponding button in the toolbar:

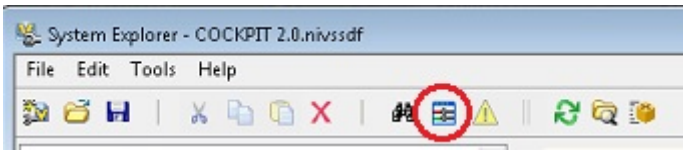


Figure 3.21: Configure Mappings button

In the dialog that will open we have to select our controller, OPAL, from the Network list

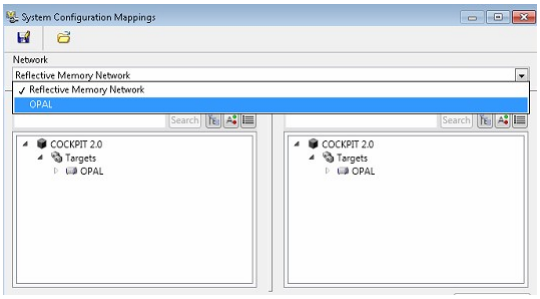


Figure 3.22: System Configure Mappings

Then for each signal, we select 'Source' and 'Destinations' navigating in the tree and confirm clicking on 'Connect' button.

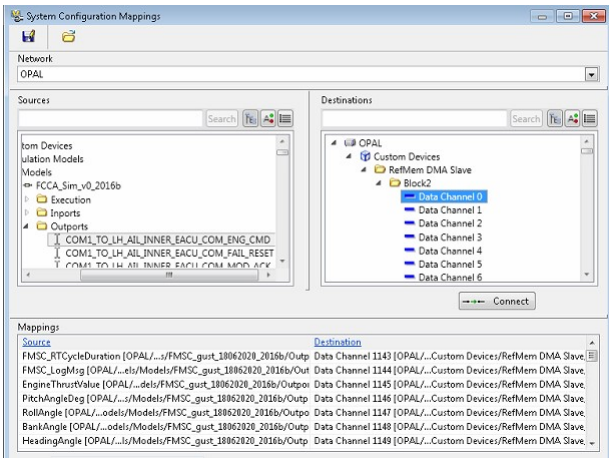


Figure 3.23: System Configuration Mappings

Confirmed mappings will be shown in the lower part of the above dialog.

As already said, all the signals in *Outports* folder shall be connected to Block2 of RefMem DMA Slave, and all channels of Block1 shall be connected to each signal in *Inports* folder.

To properly perform this operation, we need to know, for every variable, the address of the memory cell that will be written or be read. It is represented by an integer, indicating the channel in the corresponding block, and all of them are given to us into an excel file, that is **RFM affectation**, where RFM stands for *ReFlective Memory*.

Addresses of the RFM variables declarered in RFM affectation file, are calculated as

$$\mathbf{block\_address + variable\_address\_in\_block}$$

Let's make a couple of examples.
We consider the signal

$$\mathbf{COM\_TO\_FCC\_COM1\_EACU\_MOD\_ACK\_REQ},$$

that is an output. Since it is an output it should be connected to a channel in Block2. To select the correct channel, we look for COM_TO_FCC_COM1_EACU_MOD_ACK_F in the RFM Affectation file. It can be found at the line 1314.

1311		COM_TO_FCC_COM1_EACU_ID
1312		COM_TO_FCC_COM1_EACU_LANE
1313		COM_TO_FCC_COM1_EACU_OPERATING_MODE
1314		COM_TO_FCC_COM1_EACU_MOD_ACK_REQ
1315		COM_TO_FCC_COM1_EACU_ENG_STATUS
1316		COM_TO_FCC_COM1_RATE_STATUS

We know that the address of Block2 is 430, so the channel for the considered signal is calculated as

$$\text{line\_in\_excel - 430}$$

Applying this we obtain that the channel shall be 884. We select the considerd signal from 'Source' in 3.22 and Channel 884 from 'Destination' and click 'Connect' button.

On the other hand, inputs of each model have to be connected to Block1. Its address is 1, so to select the channel we have to modify the previous formula as:

$$\text{line\_in\_excel - 1}$$

After this modification, we proceed the very same way as for outputs.

Once mapping has been done, CERTIA performed some tests to better understand how to position FCCs, EACUs and RIUs in the hardware system. The main

result is that it is more efficient to run FCS components directly on the ETS calculator rather than passing through the Simulation Model (SM). This means that mapping is not needed for Flight Control System.

The reason is that the time required for writing the shared-memory rises with the number of variables to be written. Since on the real time system we have a fixed-time step, time needed to write the variables cannot exceed it. Otherwise a system overrun occurs.

Together with CERTIA partners, we noticed that FCCs, EACUs and RIUs subsystems are not very complicated for the ETS and they have to run only at 100Hz and most of the involved variables are computed as NI Inport = NI outport. For this reason we tried to run it directly on the ETS to see if this additional CPU load is tolerable. Since it still stays around 50% of CPU load, we considered it acceptable.

3.4 Mapping of FMSC and HMSM

The workflow described in the previous section has to be repeated in the very same way for both Flight Mechanics Simulation Computer and Health Management System Module. So, adding NI Inports and NI outports in the Simulink models, generate the dll and use NI VeriStand to map inputs and outputs on the reflective memory optical ring.

Also for the two cited modules, like for the FCS, addresses of the variables on the reflective memory can be found in the same excel file.

For the health module, to meet a requirement from the designer about the file generated from RT computer to record data, we had to add another custom device that is the Embedded Data Logger. Using it, we were able to put all the data collected into a single file and group them according to the actuator from which they came.

Chapter 4

Conclusions

This thesis is helpful to show the advantages of following a model-based approach in developing applications like control systems or, as in our case, in aerospace or in automotive fields.

In particular, we surely think about automated code generation and rapid control prototyping, referring with this to model simulation inside NI VeriStand.

The automated code generation permitted the team involved in the project to almost forget about the code. We only cared about the correct file extension that we needed. In other words, we checked only if a dll file were correctly generated.

Rapid control prototyping allows to analyze the performances and the integration of the various parts considering also the real time aspect in the simulation.

Considering the real-time aspect allowed us to shorten the testing time, and gave us the possibility to understand how the code for the three models we dealt with, behaves in a situation very similar to the final one, without having a physical prototype of the whole Iron Bird.

Our tests showed that the modules composing CERTIA's test bench are able to correctly communicate with each other. It is, then, possible to assert that Politecnico carried out his task in a proper way.

Chapter 5

Acknowledgements

The research work presented in this paper was performed within the ASTIB project, which has received funding from the Clean Sky 2 Joint Undertaking under the European Union's Horizon 2020 research and innovation programme under grant agreement CSJU – GAM REG 2014-2015.

Finally, I would like to thank some people that showed me their support and encouraged me to never give up.

First of all, I would like to express my sincere gratitude to prof. Luca Ardito, prof. Maurizio Morisio and prof. Riccardo Sisto, supervisors of this master's degree thesis, for their assistance at every stage of the research project. Their feedback pushed me to improve my thinking and helped to improve the quality of the work.

I would like to thank my colleague Chenguang Long, who collaborated in this project. Due to the pandemic we hadn't the chance to personally know each other, but I found a kind and helpful person. Without him, this experience wouldn't be the same.

I would like to thank people from CERTIA, LEONARDO S.p.A, Politecnico's DAUIN and DIMEAS, who designed the the three models on which we had the work, for their comments and suggestions.

A special thank to my mother, my father and my sister. With their precious support, also from an economical point of view, they allowed me to focus only on completing my studies and to become the person I am today.

Thanks to Simone, Thomas, Andrea, Sabatino, Gennaro, Miriam and Gianmario, my friends in Torino. They made these years happier and pushed me to give always my best.

Thanks to Max, Davide, Fede, Simo, Salvo, Alessandra, Alessio, Loris, Lorenzo, Emi, my longtime friends, for make me feel present despite the distance and for always believing in me.

Thanks to everybody!

Angelo Cordaro

Bibliography

ABACO Systems. (n.d.). *Reflective memory network.* Retrieved from `https://www.abaco.com/download/real-time-networking-reflective-memory`

Andrea Dellacasa, J.-C. M., Giovanni Jacazio. (2017). Astib: a research programme on innovative electro-mechanical and iron bird within the cleansky2 research initiative. , 27–36.

Baek, I. J. (2002). A survey on reflective memory systems. *15th CISL Winter Workshop*.

Chia Shen, I. M. (2001). Rt-crm: Real-time channel-based reflective memory. *Mitsubishi Electric Research Laboratories*.

Garrison, P. (2018). Hinge moments explained. *Flying*.

National Instruments. (n.d.). *What is veristand?* Retrieved from `https://www.ni.com/it-it/shop/data-acquisition-and-control/application-software-for-data-acquisition-and-control-category/what-is-veristand.html`

Wikipedia. (n.d.). *Simulink — Wikipedia, the free encyclopedia.* Retrieved from `https://en.wikipedia.org/wiki/Simulink`