ESTEC, 20th October 2016

SW autocoding for AOCS

10th ADCSS Workshop

J. Salvador Llorente, SENER Ingeniería y Sistemas, S.A.

Aeroespacial

www.sener.es

1 Contents

1. Introduction: Autocoding for AOCS:

- AOCS Design ► Prototype ► Detail models► Debug & test models ► ~~Models Ready ► Decision for Manual(initiate the process) versus autocoding (press the button).~~
- Decision for AUTOCODING has **implication early in the process**:
  - Design oriented towards autocoding SW. Hierarchy, Models definition, etc.
  - Systematic & controlled process for prototyping, implementation, **configuration**, …
  - Design and implementation to facilitate Verification, Maintainability and Usage
- **Design and coding Rules/guidelines: early application** for optimal tool usage
  - AOCS SW to guarantee quality of produced code, complexity, readability, etc.
  - Facilitate requirements traceability through design →implementation → verification
    - ECSS satisfaction: ECSS-E-ST-40C/ ECSS-Q-ST-80C.
- Those conditions imply an "initial" overhead, that will pay in return in later phases:
  - Ensures efficient, robust and solid SW(e.g. no coding error) and AOCS subsystem.
  - Provides a high level of flexibility and efficiency in changes and evolutions.
  - Design team involvement and continuity in the SW development process.
  - May simplify the SW development cycle and documentation, but room for improvement exist in this area.

2. Starting process

- High level modelling languages helping design-analysis process (90´s).
- Each modelling language SW introduced its own code-generation (mainly two).
- INITIAL autocoding tools: Simple and Rigid, but Uncertain.
- Space Market needs Guarantees ⇒ Reluctance of application in High Class (A-B-C) SW.
- However, it was applied in programs with
  - Hard constrains on schedule and budget
  - Flexibility in project cycle and receptiveness to “controlled” risks
  - Interest on innivations
- Autocoding has been introduced progressively in sequence:
  1. Low-cost, Technology acquisition/demonstration, National(or amateur) programs.
     - E.g.: MINISAT (see later)
  2. Technology evolution and improvement
  3. Application to “special” missions (technology, demonstration, academic…).
  4. Maturity and incorporation of highly demanding missions and applications.

2. Starting process: MINISAT

INTA mission (MOD)
- *Initiated 1994 → Launched 1997*
- *200kg, LEO, Sun-Pointing, Spin biased*
- *Primary Objective: Technology deployment-demonstration*
- *Secondary: Science*

- SPANISH NATIONAL SPACE PROGRAM:
- Ideal candidate for autocoding since:
  - Technology acquisition program
  - Limited complexity
  - Demanding Schedule and budget
  - SW development allowed deviation from strict Space SW Standards
- Start: 1994-Launch 1997. 14 months for ADCS design development and ADCS SW full functional validation, up to delivery for integration.
  - ADCS Design, development and prototyping based on emerging commercial tool.
  - Zero toolboxes available for guaranteeing SW results… (RTW version 1.0-1.1)
  - Autocoding in one single block …
    - Modifications necessary in autocoded SW for memory management: incorporated in later versions of the autocoding tool
- Excellent results, no ACS failures in the mission, significant lifetime extension.

USER STORY

MathWorks Tools Cut Satellite Software Development Time

The Challenge

To develop the attitude control system for Minisat-01, the first satellite to be completely built in Spain

The Solution

Use MATLAB®, Simulink® and Real-Time Workshop® to develop, test, and automatically generate code for the attitude control system

The Results

Accelerated simulation

Substantial time and cost savings

Problem-free performance, helping to extend the satellite's mission life

"We faced the challenge not only of developing the software for the attitude control system for Minisat-01 in less than one year, but also of completing exhaustive tests before the integration of the software with the other satellite systems, all within 14 months. It would not have been possible to develop, produce, and test the software within that time frame without MathWorks tools."

—Jose Ramon Villa, División Aeroespacial, SENER Ingeniería y Sistemas, S.A.

In June 1994, the Instituto Nacional de Técnica Aeroespacial (INTA), Spain's equivalent of NASA, contracted with the Spanish aerospace company CASA to design and manufacture the Minisat-01 satellite. CASA subcontracted the development of the attitude control system (ACS) to SENER Ingeniería y Sistemas.

The ACS controls the three-axis, sun-pointing attitude of the satellite in orbit to maintain solar power and telemetry control. The system also controls the spin and rotation of the satellite to accommodate three onboard low-earth orbit experiments.

SENER was able to complete the ACS within a very tight schedule and budget by using MATLAB®, Simulink®, and Real-Time Workshop®. "We firmly believe that this is the approach to be followed in the future for cost-effective development of real-time control systems," says SENER ACS Technical Manager, Jose Ramon Villa.

The Challenge

The ACS software had to be less than 100 KB in size, and it had to be written, tested, and integrated with other systems on a single, shared, onboard, 16-Mhz Intel 386 processor within 14 months on a budget of only $1.3 million for the whole ACS system.

The SENER team quickly realized how rest
the 14-month deadline was; if they followe
normal procedures, writing the real-time c
would take more than a year.

To produce the ACS system on sched
and within budget, SENER would need to
real-time flight code concurrently with de
and testing algorithms. They knew that Mat
control design products supported this ap
"We had been using MATLAB for other thi
learned about the diagramming and code
tools directly from The MathWorks," Villa
However, Simulink and Real-Time Worksh
new to SENER.

It was a risk, he admits, to stake the e
development effort on tools that SENER h
used before. But it was a risk the team wa
to take: developing the real-time flight cod
designing and tuning algorithms at the sar
was the only way to complete the project o
and within budget.

The Solution

The ACS software, also known as the onbo
attitude software (OBASW), was develope
six top-level software components. Three

Application Areas
Aerospace
Flight controls
Test and simulation

MathWorks Products Used
MATLAB®
Simulink®
Real-Time Workshop®

The MATHWORKS
3 Apple Hill Drive
Natick, MA 01760-2098 USA
Tel: 508-647-7000
Fax: 508-647-7101
E-mail: info@mathworks.com
www.mathworks.com

components were entirely hand-coded, while two are full Simulink subsystems. The remaining component comprises a complex Simulink subsystem and some handwritten code. In all, the OBASW uses up to seven Simulink subsystem levels, over 13,000 Simulink blocks, 11 hand-coded S-function blocks imported into Simulink diagrams with 6,000 lines of handwritten C code, and 25,000 lines of automatically generated code.

To comply with the size limitation of the ACS software, the SENER team developed a C utility that automatically rolled lines of Real-Time Workshop generated code into C loops, a feature included in later versions of the product. From this C source code the team produced an executable that reduced the size of the final OBASW to a little over 100 KB.

Once the OBASW was written, it had to be tested before being loaded onto the onboard computer. The intensive software testing had to be done quickly, but it had to include simulating the behavior of the satellite through three months of continuous satellite mission time.

The schedule did not allow for three months of testing. However, because the original attitude determination and control algorithms and logic had been developed in the MATLAB and Simulink environment, the limited testing time was not a problem for the team: they could use that same environment to accelerate testing.

First, the SENER engineers used Simulink and Real-Time Workshop to create a dynamic simulator that would test the code against an internal clock accelerated to cycle more than 20 times faster than real time. The different OBASW modules, including those handwritten in C, were integrated in Simulink as a fully coded Simulink model. Using this approach, the team extended testing for 8,000 hours, or half of the satellite's projected two-year mission life.

The Minisat-01 was launched in April 1997 using the airborne Pegasus launcher from Orbital Sciences Corporation. Since then, its ACS has operated without malfunction.

The Results

- **Accelerated simulation.** Using Simulink and Real-Time Workshop, SENER engineers built a simulator that ran more than 20 times faster than real time. This enabled them to simulate almost a full year in orbit—half the satellite's mission—before it was launched.
- **Substantial time and cost savings.** It took just over a year to design the ACS algorithms and develop and test the software for integration with other onboard software systems. "Using MathWorks tools shortened the development cycle, accelerated testing and required fewer people to document algorithms and write code," says Villa.
- **Problem-free performance, helping to extend the satellite's mission life.** The OBASW has performed flawlessly following the normal adjustments that were made after it was put into orbit. The satellite passed its original mission life of two years and was still operating correctly three years after its launch.

To find out more:
www.mathworks.com
www.sener.es

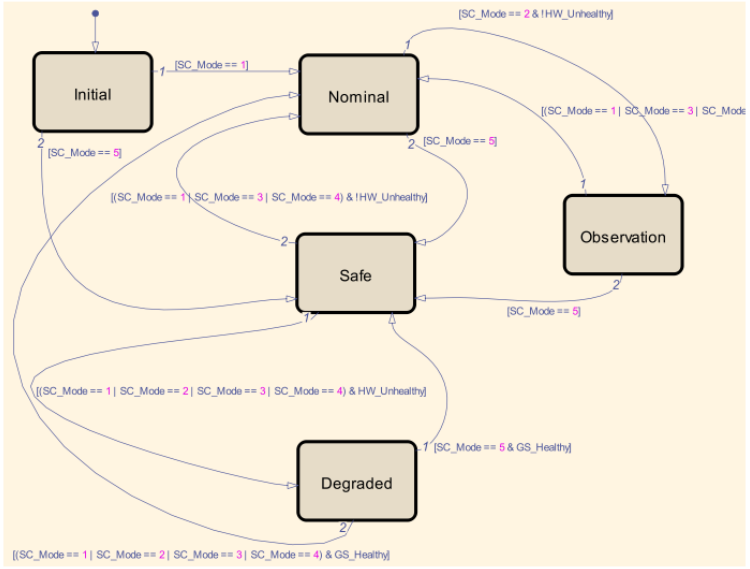
9B45v0

3. Getting Mature: Technology evolution

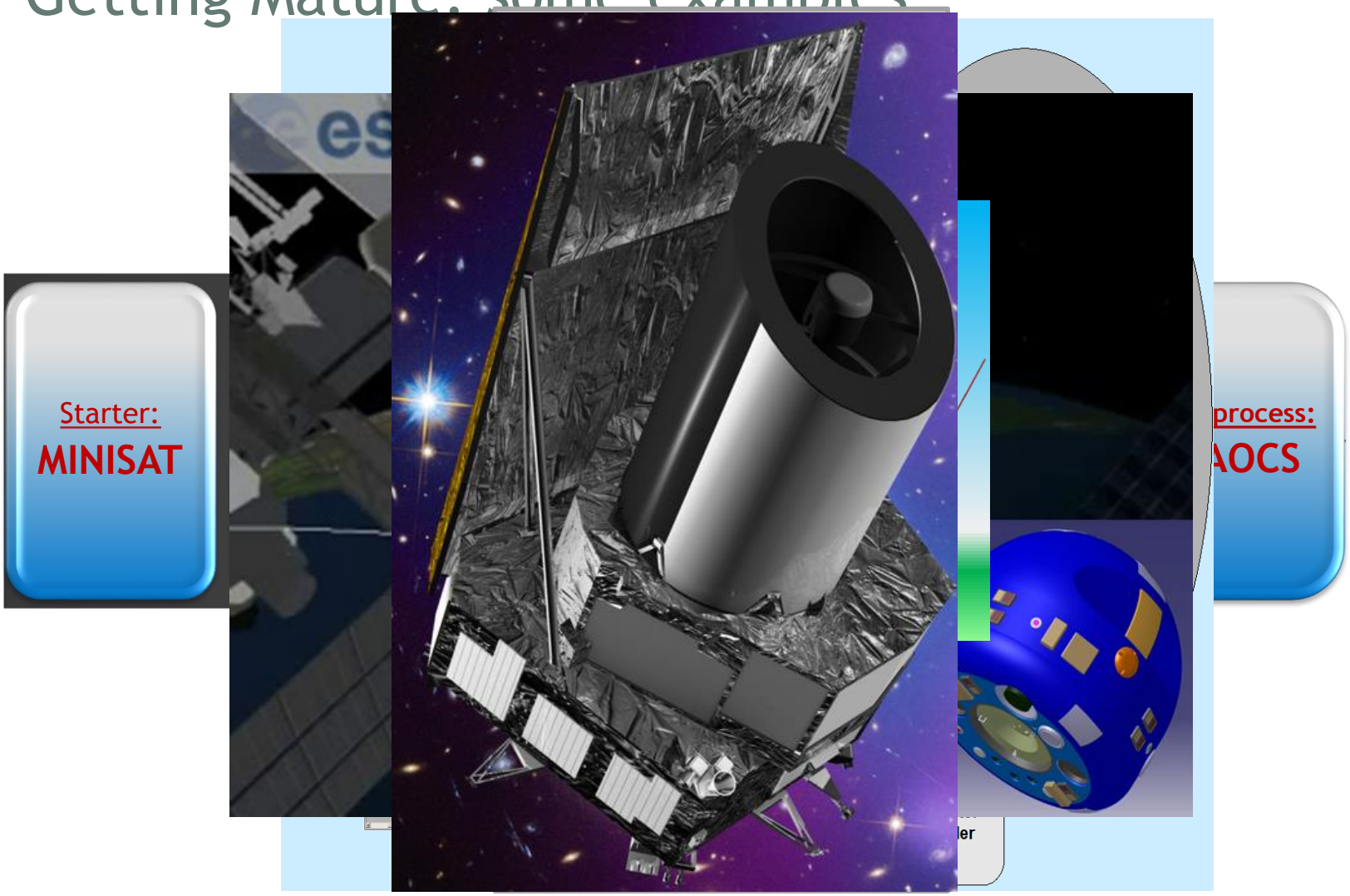
- TECHNOLOGY ► Tools Evolution, Extensions, Integration with SW dev. tools, etc.
- AUTOCODING PROCESS ► process Investigation and Improvement (each company)
  - GENERIC AOCS (GAOCS). ***Design to Autocoding preparation***. Design Tools, Environment, Simulators, AOCS/GNC building blocks and Verification tools
  - GENERIC AOCS Test Bench. Autocoding for the AOCS test benches (SIL+PIL+HIL).
  - ACODEG: Mechanization & cycle improvement for OBSW autocoding.
- Unconfident Customers ► deployment and demonstration
  - Autocoding process application in Ground Systems(EGSEs):
    - Classic OBSW for Herschel-Planck ACMS, but
      - Design, Specification & validation symbiosis with GENERIC AOCS.
      - AOCS-SCOE RTS (MOSAIC-EUROSIM).
    - IXV: GNC design and SCOEs: As for H-P extensive use in D.E. and SCOE (although more mature elements and tools, and different SCOE environment).
  - Demonstration and application in more flexible/risky receptive missions:
    - OPTOS ADCS, but also Eagle-Eye (Virtual SC AOCS), ASVIS, ...
    - Technology demonstration missions (just in ESA): PROBA-1,2,3,V, SMART, …

3. Getting Mature: OPTOS

- OPT
  - –
- ADC
  - –
  - –
  - –
  - –
  - –



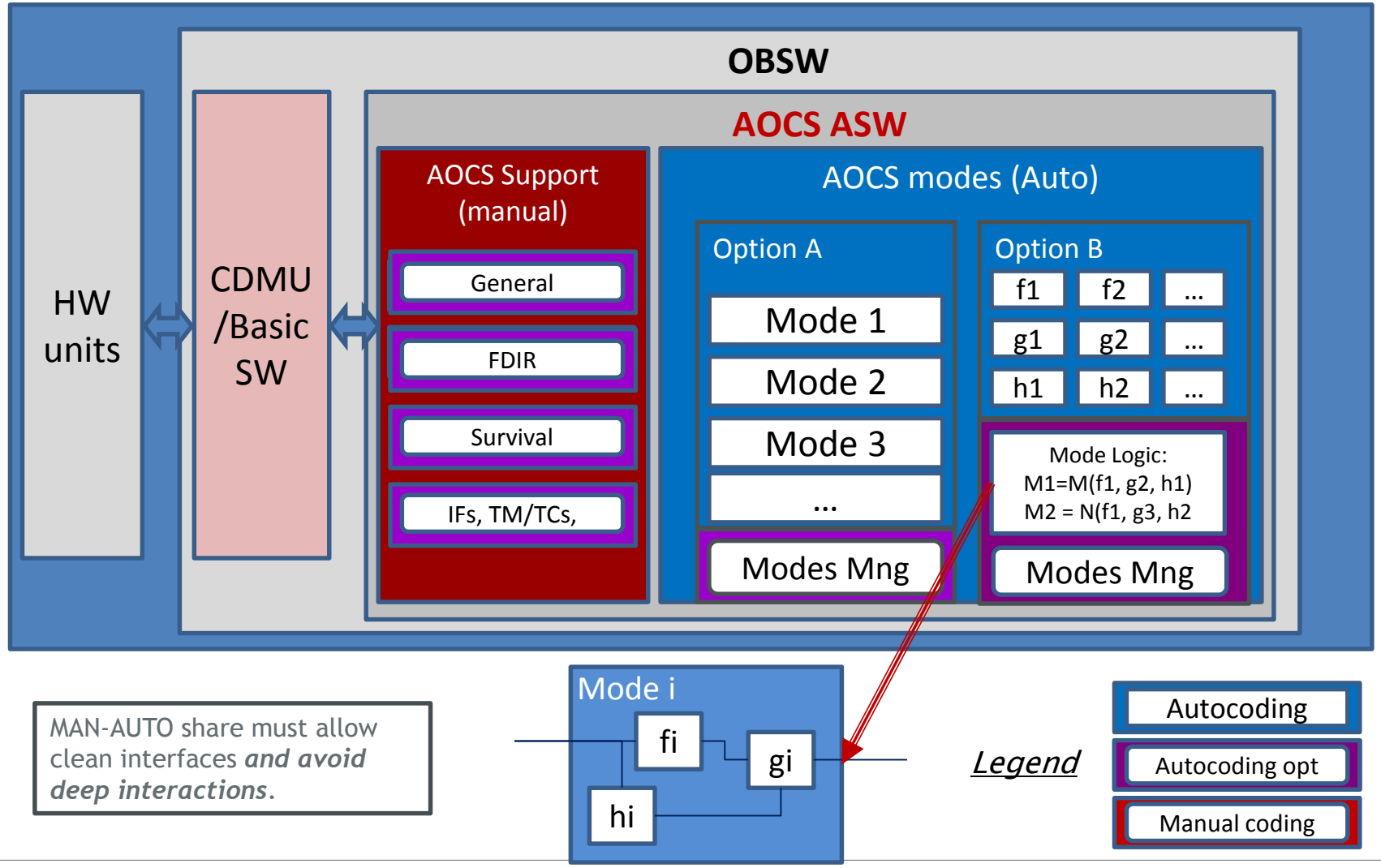
3. Getting Mature: some examples



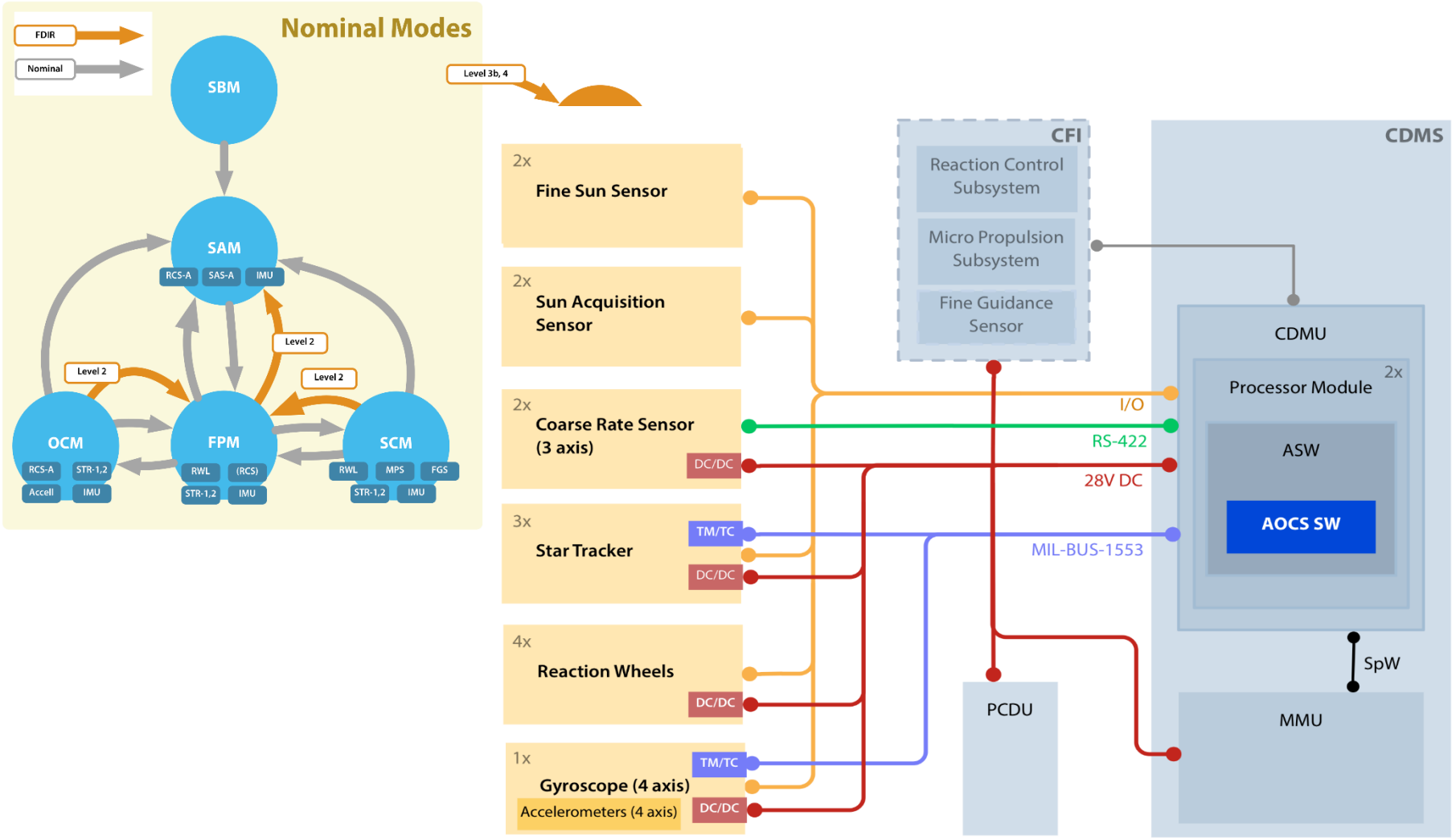
4. Application: SHARE Manual-Automatic

- Autocoding tools are NOT always the best for generation of SW. Depends on taste, but:
  - + Good for Architecture Implementation, blocks interconnection, data flow, combination of information, data processing, etc
  - – Not so good for Operation Logic Implementation, Modes Management (exc. Dedicated tools), TM/TC handling. Delicate configuration control issue.
- Based on those considerations, combination of classical and model based autocoded SW is a frequent solution:
  a) From only individual functions being autocoded, and integrated in a general manual SW which configures the modes.
  b) To the fully autocoded SW to be integrated in a dedicated computer/processor-board.
  c) Intermediate solutions are typically best, where some functionalities are implemented outside the autocoded SW, depending on convenience.
  d) Also manual code (or script language) is frequently used embedded in the models, for functions for which classical SW flow is simpler, and easier not only for implementation, maintenance, verification, etc.

4. Application: SHARE Manual-Automatic



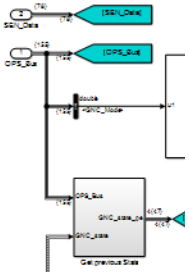
5. EUCLID AOCS

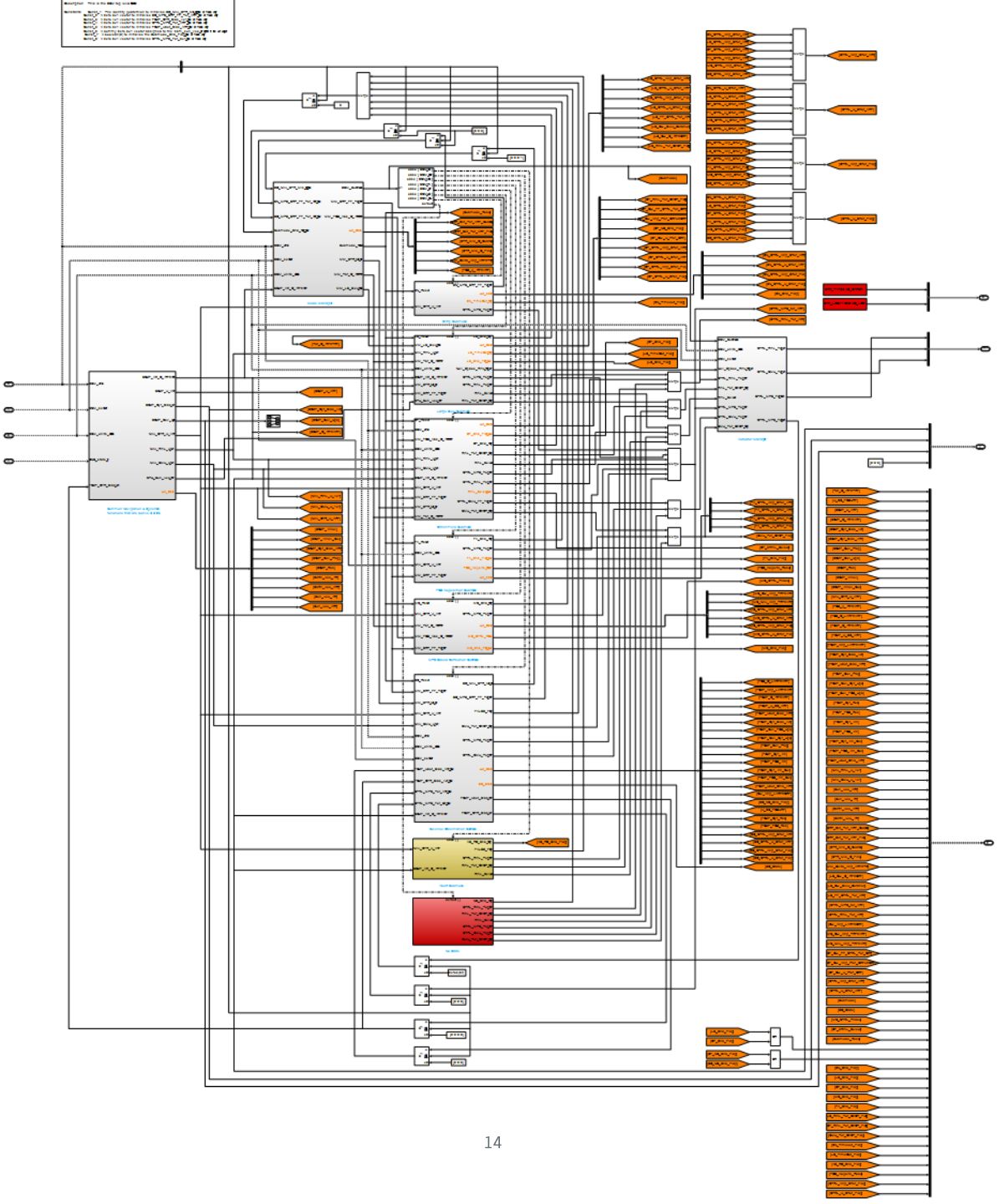


5. EUCLID: Dedicated share, process and organisation

- Dedicated organisation according to roles, shares, and processes:
  - BSW(System/TASI), provides the HW interactions, final implementation of TMs, and reception/transmission of TCs, and in general the central SW services of the CDMU.
  - AASW.
    - MANUAL code to handle direct interactions with BSW, FDIR, Safe Mode, AUTO management, Operations (TM/TCs), and implement modes activation.
      - In charge of ADSNL (in general ADSNL manages the overall AOCS SW)
    - AUTO implements all the AOCS modes, detailed algorithms, submodes and states, subset of FDIR functions(at the level of algorithms implementation), etc.
      - In charge of SENER (includes autocoding implementation)
- Development in versions: v0 (Architecture + IF), v1 (Functions + performance), v2(FDIR)
- Integration flow: IFs definition and checks
  - AUTO-MAN integration (in SDE under MAN company control).
  - BSW-AASW integration (in SVF under BSW company control).
- Development includes parallel versions and AUTO-MAN development (phasing issue).
- SW development based on general SW ECSS. Significant improvement might be obtained.

5. EUC





5. EUCLID: SW flow and versions

EUCLID AOCS STATUS

- AOCS SRR and PDR closed.

  In progress for CDR.

- AOCS models ready in design environment. Reference tests available for the SW production and verification.

  SW autocoding and UIT in progress

- Autocoding process already exercised for SAM and SCM

  MAN-AUTO integration and Equivalence checks performed.

- Results according to expectations.

  CPU Budget verified by test in target processor.

AASW

6. AOCS SW Autocoding Results

- Autocoding was initially of difficult application (not intended) for critical SW (e.g. AOCS)
- Reliable processes obtained, by means of dedicated developments and efforts.
- Tools have evolved, extended and grown providing a set of (still incomplete) tools.
- Maturity obtained by different means/projects: suitable for application in critical and demanding AOCS OBSW. Advantages:
  - ✓ + Direct involvement of design team in the SW production.
  - ✓ + minimisation of manual intervention and coding mistake
  - ✓ + Agility in modifications, and iterations. Much shorter SW cycle & cost reduction.
  - ✓ + Simplification in documentation and specification process may be exploited
  - ✓ + Flexibility in the generation of SW test data and SW test cases
- Autocoding advantages require some challenges in processes, and alleviation in conventional SW habits (criticism on the advantages). *To be improved:*
  - ? – Early application of rules & processes to exploit the potential advantages.
  - ? – Phasing adaptation: Intense initial effort in AOCS development (models and D.E.).
  - ? – SW docs simplification (specification, design description: models should replace it)
  - ? – ECSS adaptation to those autocoding conditions looks necessary (not just tailoring)

The way to see the future

Thanks