The PLATO instrument control unit software: a model based SW architecture

Emanuele Galli, Giovanni Giusi, Andrea Russi, Fabrizio De Angelis, Maria Farina, et al.

Emanuele Galli, Giovanni Giusi, Andrea Russi, Fabrizio De Angelis, Maria Farina, Scige Liu, Anna Maria Di Giorgio, Stefano Pezzuto, Mauro Focardi, Marina Vela Nuñez, Rosario Cosentino, "The PLATO instrument control unit software: a model based SW architecture," Proc. SPIE 12180, Space Telescopes and Instrumentation 2022: Optical, Infrared, and Millimeter Wave, 121804P (27 August 2022); doi: 10.1117/12.2630056

SPIE.

Event: SPIE Astronomical Telescopes + Instrumentation, 2022, Montréal, Québec, Canada

The PLATO Instrument Control Unit Software: A Model Based SW architecture

Emanuele Galli[1*], Giovanni Giusi[1], Andrea Russi[1], Fabrizio De Angelis[1], Maria Farina[1], Scige J. Liu[1], Anna Maria Di Giorgio[1], Stefano Pezzuto[1], Mauro Focardi[2], Marina Vela Nunez[2], Rosario Cosentino[3]

[1] INAF-IAPS, Institute for Space Astrophysics and Planetology, Via del fosso del cavaliere 100, 00133, Rome, Italy,

[2] INAF-AAO, Arcetri Astrophysical Observatory, Largo Enrico Fermi 5, 50125, Florence, Italy,

[3] INAF – Catania astrophysical Observatory, TNG Galileo National Telescope

ABSTRACT

PLAnetary Transits and Oscillations of stars (PLATO) is a medium-class mission selected by ESA in the framework of the Cosmic Vision programme. The PLATO Instrument Control ICU is responsible for the management of the scientific payload, the communication with the SVM, and the lossless compression of scientific data before the download to the satellite Mass memory. The ICU requirements have been finalized for the Preliminary Design Review. The resulting technical specification has been used to design a Model Based Software architecture. The first two versions of the PLATO ICU SW have been released and fully validated on the target platform. This paper provides the details of the solutions adopted to cover all implemented services.

Keywords: Instrument Control Software, PLATO, lossless compression, PUS services

1. INTRODUCTION

PLAnetary Transits and Oscillations of stars (PLATO) is a medium-class mission selected by ESA in the framework of the Cosmic Vision programme to be launched in 2026 (Ref[1]). With the main aim of characterizing a large number of extrasolar planetary systems by detecting planetary transits and conducting asteroseismology of their parent stars, PLATO will allow to identify suitable targets for future, more detailed characterization.

The overall PLATO Data Processing System includes 6 N-DPUs (Normal Data Processing Units), 2 F-DPUs (Fast Data Processing Units) and the ICU (Instrument Control Unit). The ICU is responsible for the management of the scientific payload, the communication with the Service Module (SVM), and the lossless compression of scientific data before the download to the satellite Mass memory.

The ICU Application Software requirements have been defined during the phase B of the mission and finalized for the Preliminary Design Review (Ref[2]). The resulting technical specification has been used to design a Model Based Software architecture using the UML model standard to describe both the static and the dynamic behavior of each single task. A total of 15 tasks have been defined adopting an event-driven approach for inter-task communication protocol. The first two versions of the PLATO ICU SW have been released and fully validated on the target platform with a full characterization of the overall data compression chain (hardware and software) performances.

This paper provides the details of the solutions adopted to cover all implemented services.

[*]emanuele.galli@inaf.it; phone 0039 06 4548 8247; fax 0039 06 4548 8242; iaps.inaf.it

Space Telescopes and Instrumentation 2022: Optical, Infrared, and Millimeter Wave, edited by Laura E. Coyle, Shuji Matsuura, Marshall D. Perrin, Proc. of SPIE Vol. 12180, 121804P · © 2022 SPIE · 0277-786X · doi: 10.1117/12.2630056

2. ICU ASW SOFTWARE OVERVIEW

The ICU-ASW is the application software that is executed in the ICU-PLATO unit (see Figure 1)(Ref[3,4]), which is connected to the SVM through 4 Spacewire links per each module (A=Nominal, B=Redundant), and, by means of the two internal router and data compression units (RDCU-A and RDCU-B in cross-strapping configuration), it is also connected to:

- 2 Spacewires to MEU#1 and MEU#2 (each MEU has a Router (MEU-R), which is connected to 6 N-DPU)
- 2 Spacewires to F-DPU#1 and F-DPU#2
- 2 Spacewires connected to N-AEU#1 and N-AEU#2
- 2 Spacewires connected to F-AEU#1 and F-AEU#2

The ICU-RDCU is, in turn, connected to the ICU/MMU by two Spacewire link (Nominal) plus other two Spacewire links for the redundant RDCU. One Spacewire is used for the CCSDS Packet Transfer Protocol (called CPTP hereafter) to communicate with DPUs (Normal and Fast) and the other one is used to communicate to subunits that supports only the RMAP protocol (e.g. F-AEUs and MEUs).

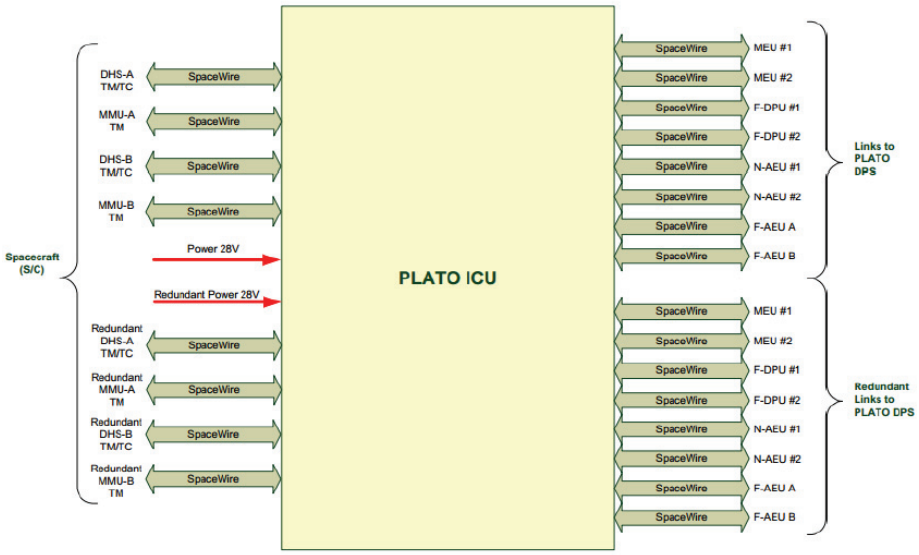

Figure 1. ICU Data & Power Interface

The ICU-ASW is the application software of the ICU instrument which has the following principal aims

- Acceptance of instrument commands from SVM.
- Execution of predefined instrument command sequences.
- Instrument health/status monitoring.
- Implementation of pre-defined procedures on detection of instrument anomalies: the instrument shall be able to adjust parameters and/or switch operating mode and/or activate subsystem redundancy when an anomaly occurs.
- HK data acquisition and packetization.

- Handling of data-pool (housekeeping and ICU configuration parameters)
- Science data acquisition, compression (Hardware and Software) and packetization.
- Transmission of data (HK, events and TC verification) from the instrument to SVM-DHS.
- Transmission of science data to the SVM-MMU.
- Execution of Onboard Control Procedures (OBCP)

In addition, the ICU-ASW shall provide the following SW oriented functions:

- The ability to load, via TCs, replacement and/or additional SW (patches, tables, etc..);
- Self-test and SW verification facilities;
- Possibility to load and dump part of ICU memory;
- Possibility to write and check NVM: possibility to inhibit these functions during flight operations.

2.1 ICU-ASW Software ports

Th ICU-ASW can be seen as a single component that encloses all expected functionalities and provides a set of ports to external actors/instruments in order to send and receive commands, data and events. In particular, the following logical ports are expected (see Figure 2):

- **dhs_port**: provides the service to send high-priority telemetry to the SVM-DHS and require the service to receive telecommands from the SVM-DHS, by using the PUS protocol over CPTP (Ref[7,8]). In this case the physical connection will be for both nominal and redundant sides.
- **mmu_port**: provides the service to send low-priority telemetry to the SVM-MMU (by using the PUS protocol over CPTP). In this case the physical connection will be for both nominal and redundant sides.
- **meu_psu_port**: provides the service to send Write and Read commands to the 2 MEU-PSU (by using the RMAP protocol). In this case the physical connection will be MEU#1 or MEU#2.
- **meu_router_port**: provides the service to send Write and Read commands to the 2 MEU-ROUTER (by using the RMAP protocol). In this case the physical connection will be MEU#1 or MEU#2.
- **meu_dpu_port**: provides the service to send TCs and requires services to receive TMs (by using the PUS protocol over CPTP) and Read/Write commands (by means of RMAP protocol) to the 12 N-DPU of MEU#1 and MEU#2. In this case the physical connection will be MEU#1 or MEU#2. RMAP protocol is used only with Boot-SW of NDPUs.
- **feu_port**: provides the service to send TCs and requires services to receive TMs (by using the PUS protocol over CPTP) and Read/Write commands (by means of RMAP protocol) to the 2 F-DPU of FEU. In this case the physical connection will be F-DPU#1 or F-DPU#2. RMAP protocol is used only with Boot-SW of FDPUs.
- **n_aeu_port:** provides the service to send TCs to the 2 N-AEU and requires services to receive data from N_AEU (by using the RMAP protocol). In this case the physical connection will be N-AEU#1 or N-AEU#2.
- **f_aeu_port:** provides the service to send TCs to the 2 F-AEU and requires services to receive data from F-AEU (by using the RMAP protocol). In this case the physical connection will be F-AEU#1 or F-AEU#2.
- **rdcu_port**: provides the service to read and write data, through the RMAP protocol, from the internal ICU RDCU router and HW compressor. In this case the physical connection is internal to the ICU by means of Spacewire links (4 link in cross-strapping configuration)
- **hw_port**: provides the interface to setup the ICU board, to read the internal status (e.g. ADC housekeeping), to access to the FPGA-MMU memory, and to power on/off the RDCU.

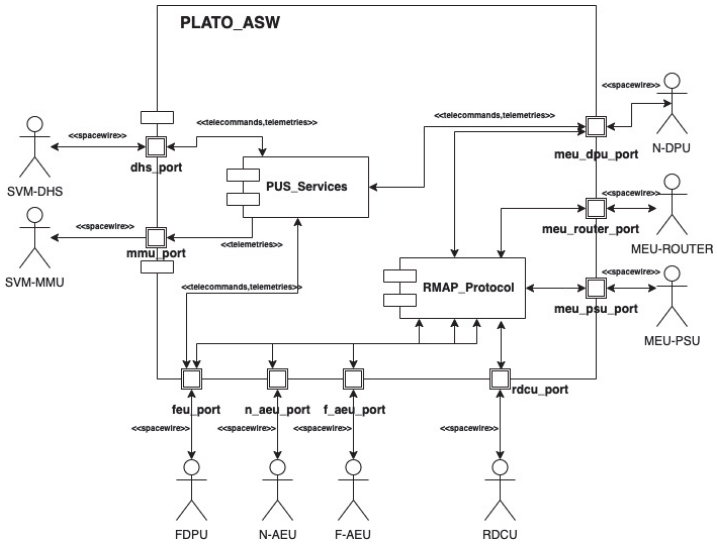

Figure 2. ICU-ASW Data Interfaces

3. ICU ASW SOFTWARE DEVELOPMENT PROCESS

The ICU ASW development process is based on an iterative V model, with 4 main deliveries of 4 different version of the ASW (see Figure 3). The first two versions of the ICU ASW have been already delivered to the consortium, and the third version will be delivered after the closeout of the Critical Design Review (CDR).

The Software System Requirement Specification (SSS) is the input of the ICU ASW requirements, and it is provided by the German Aerospace Center (DLR). The SSS contains software requirements specific to a unit of the Payload (i.e., ICU, N-DPU, F-DPU), as well as general requirements applicable to all units. The SSS requirements are provided both in PDF and in IBM DOORS module (Ref[10]).

The SSS DOORS module is imported in the INAF DOORS tool to simplify the tracing of each requirement. Each requirement applicable to ICU ASW, has been converted in one or more ICU ASW requirements that provide more details (e.g., if the CPU occupation must be reported, a new requirement will specify in which packet and which parameter reports the CPU load). The full process generates the ICU Software Requirement Specification (SRS)

The SRS is used to define the ICU ASW Verification & Validation test plan in order to define all test scenarios to be used during the test phase and the writing of test procedures.

The ICU SRS is the input document for the ICU ASW Design. In order to trace each specification, the SRS DOORS module is imported in the IBM Ration Rhapsody tool (Ref[9]). The IBM Ration Rhapsody is a tool to design and implement an application software by adopting principles of Model Driven approach. At the end of the process, all requirements that must be implemented shall be covered by either a module, a function or a variable.

Currently, more than 800 requirements are traced to the ICU ASW Design project and more than 90 modules are defined. A total 15 tasks have been identified and the behavior of each task has been designed by using UML statecharts and connected by using structure diagrams. Complex functions are modelled by using flow charts in order to verify quickly if the function is implementing the expected behavior. A test plan for each model is defined, and it is the input for the generation of ICU Unit and Integration Tests.

The code of language C is automictically generated by using the Rhapsody tool, and each unit is verified by using the VectorCAST Unit Test tool (Ref[11]).

Lastly, the verification and validation of the ICU is conducted by using the TSC5 from THERMA (Ref[12]). The TSC5 allows to write scripts in TCL language to perform automatic and continuous integration testing. Reports are then provided to system team at each milestone.

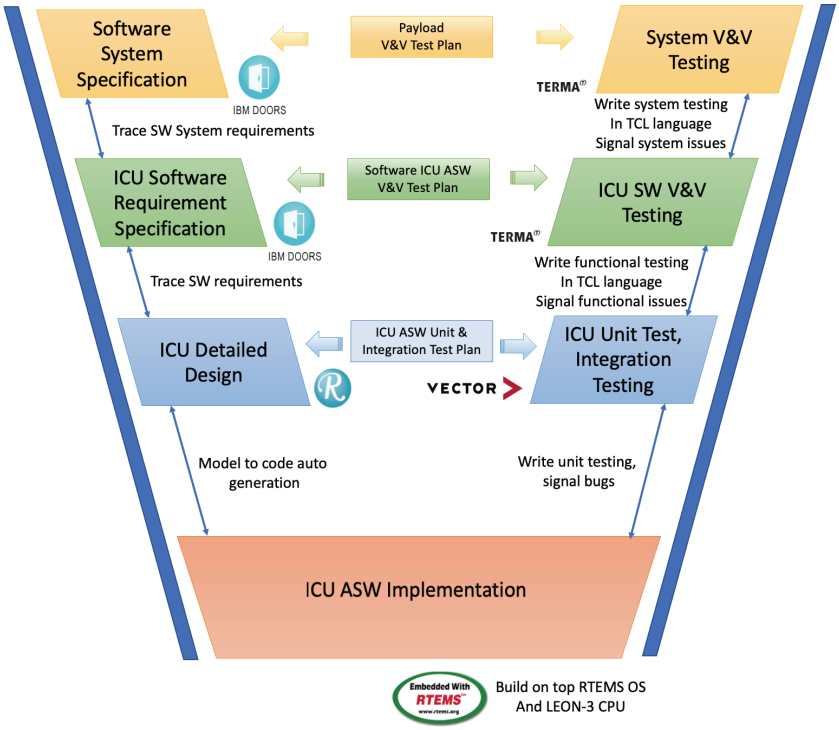

Figure 3. ICU-ASW V Model and adopted tools

4. ICU ASW MULTILAYER ARCHITETCURE

In order to simplify the ICU-ASW architecture, it has been decomposed in three different layers (see Figure 4):

1) Operating System Layer: this layer is strictly correlated to the adopted board and Operating System
2) Middleware Layer: this layer provides an abstraction of the Operating System layer as well as a set of adapters to hardware devices. Moreover, it provides a set of functions to implement an event-driven and time-driven application.
3) Application and Communication Layer: contains all modules to implement all use-cases that are expected for the ICU-ASW. It contains the module for the TM-TC handling, the PUS service Handling, the Subsystem Actuators, the Science Operation Handler and the PLATO mode handling.

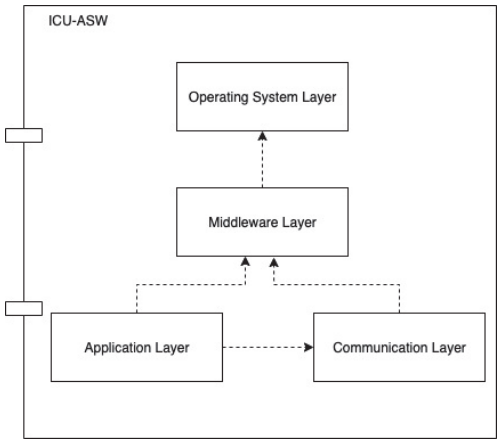

Figure 4. ICU-ASW Software Layers

4.1 Operating System Layer

The Operating System Layer is composed of the following major components (see Figure 5):

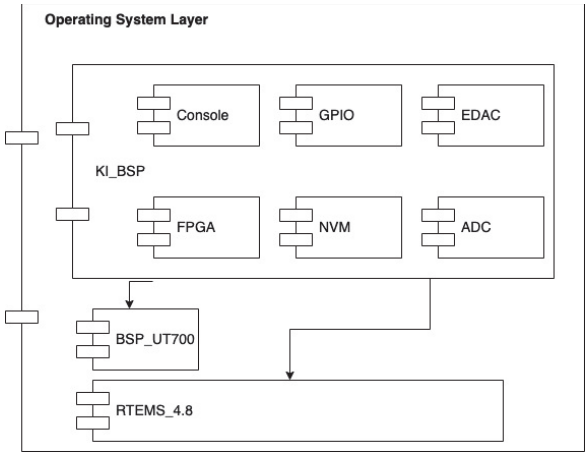

Figure 5. The ICU-ASW Operating System Layer

RTEMS: it includes all the library of the Real Time Operating System RTEMS® by EDISOFT and provides principally the following functionalities interfaces:

- Signal Handling: provides a signal interface for task
- Interrupts Handling: provides the interface for connecting interrupt service routines to interrupt hardware
- Task: provides the interface for the task management facilities (create, start, suspend/resume, delete, disable/enable scheduling, change priority, delay execution)
- Scheduling: manages the scheduling among tasks. Pre-emptive priority scheduling policy shall be adopted
- Semaphores: provides an interface for the management of binary, mutual and counting semaphores for the synchronization and mutual exclusion of shared resources

- Message Queue: provides the interface for the management of intertask communication within a single CPU
- Clock, Timers and Watchdog Timers: provides the interface for the management of interrupt timers and create watchdog timer to trigger a specific function of a task.

BSP_UT700 (UT699): it includes the library for the management of the UT700(UT699) Leon3FT® by Cobham Gaisler

KI_BSP: it provides the CPU/MMU FPGA kernel driver, and the thin wrapper interface (API) provided by Kayser Italia company. It is principally composed of:

- Console component: for debugging purpose, allows to print an output message on a console
- GPIO component: it is responsible of management GPIOs on the ICU board. It allows to power on/off the RDCU and Spacewire transceivers (foreseen from ICU EM#1 model)
- EDAC component: it allows to manage the EDAC on the ICU SRAM volatile memory. It allows to setup the memory scrubbing and the callback when correctable or uncorrectable errors are detected
- FPGA component: it oversees the communication on Spacewire links (PUS or RMAP) and to read Spacewire statistics, copy data through the DMA on the FPGA memory, and to handle the ICU HW watchdog
- NVM component: to store and load data on the NVM non-volatile memory
- ADC component: to read housekeeping the ICU ADC

The KI_BSP uses services provided by the RTEMS operating system and the BSP of the UT700. Of the RTEMS OS the following services are adopted:

- Interrupt Manager directives to enable/disable interrupts and to connect its own routine
- Clock Manager directives to handle local timer
- Task Manager directives to block a task for a certain period
- Semaphore Manager directives to create and handle OS semaphores

4.2 Middleware Layer

The Middleware Layer is composed of the following major components (see Figure 6):

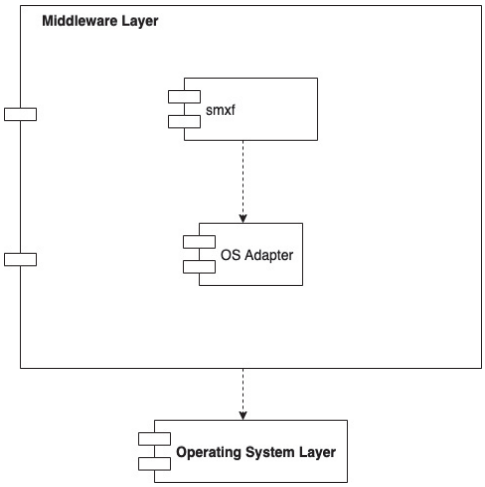

Figure 6. The ICU ASW Middleware Layer

SafetyMicroCframework (SMXF): it is part of the framework of the IBM Rational Rhapsody tool which is a UML compliant tool that allows generation of production-level code, for various real-time operating systems and hardware targets.

The SMXF is programmed in the C language. The SMXF is designed to be used by safety-critical applications generated using Rhapsody. Such applications are typically specified using statecharts, timing policies and threads. To provide the executable semantics for these abstractions, the SMXF includes logic to handle event-driven and concurrent execution, which is invoked by the generated code.

Therefore, the framework allows to develop a software by adopting a Model-Driven approach and provides all modules to build an event-driven and time-driven Real Time application. Once defined all states of each task (by using a state-diagram), it will generate automatically functions to handle events and timeout, and the transition among states.

The SMXF is an independent layer for behavioral generated code. It implements the required interfaces used by the generated code, like Reactive, Events and Timeouts. This set of classes specifies certain required interfaces, expected to exist on the generated code, e.g., call-back for event processing.

This layer uses the interfaces provided by the OS_Adapter layer to execute operating system functionality.

The SMXF contains a set of Core Services. This set of services specifies certain required interfaces, expected to exist on the generated code, e.g., call-back for event processing.

The set of core services communicates with OS Adapter layer and with the generated code.

The SMXF layer requires and relies on the correctness of the following functions on the Operating System Adaptors layer:

- Memory management – allocation and deallocation of memory segments. Though the SMXF does not contain dynamic allocations, it works on pre-allocated memory, which is assumed to be correctly allocated and accessible.
- Thread management – starting and terminating, scheduling, mutual exclusion mechanisms.
- Timing – clocking and time cycles, receiving time-tick notifications.

The set of core services relies on the existence and the correctness of the following services, in the generated code layer:

- Providing Event processing function, which is executed by the destination's thread.
- Providing rootState_entDef – the function to be executed when the behavior starts.
- Providing rootState_dispatchEvent – the function to be executed in order to dispatch events.
- Assigning appropriate operation to execute when TimedAction is triggered.

OS_Adapter: it provides a generic interface to access inter process communication interface in order to be independent from the underlying operating system layer. It provides the following major functionalities:

- Task Interface (*RiCOSTask*): provides the interface to create/start/suspend and destroy real time processes
- Mutex Interface (*RiCOSMutex*): provides the interface to the mutual exclusion OS functionality
- Semaphore Interface (*RiCOSSemaphore*): provides the interface to create counting semaphores.
- Timer clock Interface (*RiCOSTimer*): provides the interface to get the current time clock
- Event Interface (*RiCOSEventFlag*): provides the interface to exchange signals/events among tasks
- Message Queue Interface (*RiCOSMessageQueue*): provides the interface to exchange data among tasks

The OS_Adapter uses the Operating System Layer to implement each specific interface. The OS_Adapter has been implemented specifically for the RTEMS-4.8 by INAF/IAPS to create RTEMS-tasks, mutex, semaphores, events, and queues. So, for example, the RiCOTask is the interface for the RTEMS Task library, and the function RiCOSTask_init of the RiCOSTask module is the wrapper for the rtems_task_create function to create a new task object in the RTEMS system.

4.3 Communication and Application Layer

The communication and application layers contain the component to implement the protocol interface with subsystems and the SVM, and to implement the function software requirements of the ICU-ASW.

The two layers includes multiple components that can be summarized as in the following list:

Communication Layer:

- SVM component handles the PUS TC/TM (high and low priority) communication with the DHS. It oversees accepting PUS TC coming from the DHS and to upload PUS TM either to the DHS or to the MMU
- DPS component handles the PUS TC/TM with DPUs ASW (Fast and Normal DPUs) as well as the RMAP communication with DPUs BootSW (Fast and Normal DPUs). Moreover, provides the RMAP interface for F/N-AEU, F/N-CAM, MEU-PSU and MEU-Router

PUS and RMAP channels to subsystems (N/F DPUs, F/N AEUs, MEU Router and PSU) are mapped to a separated area of the FPGA memory. Specifically, the PCI FPGA SDRAM is subdivided in four different areas:

- RMAP Buffer Area: user managed, buffers here are used for transactions
- RMAP Descriptors Area: driver managed buffer for storing descriptors, headers and other data
- PUS Receive circular buffer: hardware managed buffer for incoming PUS packets
- PUS Transmit Area: user managed area for outgoing packets

Each area has a limited size, which can be fixed or variable (configurable at runtime)

Application Layer:

The application layer provides all functionalities that shall be implemented by the ICU ASW. It composed of multiple components that interact with the communication layer. In Figure 7 the interaction between all components is reported.

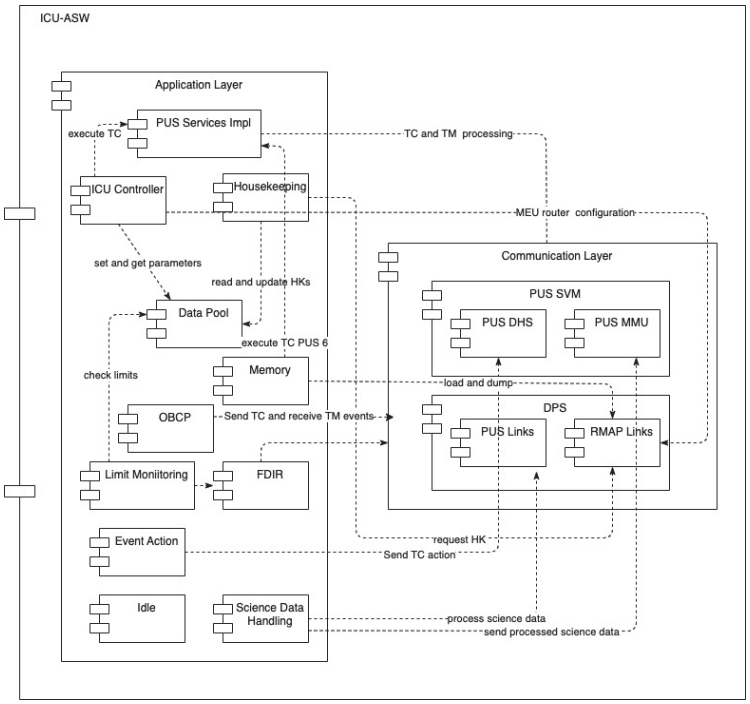

Figure 7. The ICU ASW Application and Communication Layer

The application layer is composed of the following main components:

- Data Pool component: to store and retrieve stored Housekeepings and ICU ASW parameters. Each parameter is associated with a single parameter-id (OPID) which is unique and will be the same for the full mission. The component provides functions to access to each parameter atomically.
- PUS services: provides the implementation of all foreseen PUS services
- ICU controller component: it oversees the execution of telecommands (except of memory service) and the management of ICU and Instruments operational modes
- Housekeeping component: provides the functionalities to read ICU and instruments housekeeping from F/N-AEUs and PUS and Router MEUs, to generate the relative Telemetries and to check limits of each housekeeping parameter
- Event Action component: provide the functionalities to manage actions that are activated due to an external event
- Science Data Handling component: it oversees the handling of scientific data coming from Normal and Fast DPU. It is in charge of compressing scientific data (by software algorithm or the RDCU hardware module), to packetize scientific data and to generate the relative PUS TM that will be stored in the DHS-MMU
- Memory component: it provides the module to process PUS Memory service and to generate the relative telemetries (dump or check)
- FDIR component: handles the ICU hardware watchdog and provides the set of functions to execute an FDIR when an error is detected
- Idle component: which oversees putting the CPU in idle and to SCRUB the CPU memory
- OBCP component: it executes and handles both the OBCP engine and the activation and execution of onboard procedures

Moreover an Utility component is foreseen that contains a set of functionalities that are common to all components (e.g. log functions for debugging, buffers, hash tables, etc.)

5. ICU ASW TASKS

The application software tasks are designed and implemented as <<singleton>>, therefore will exist one and only one instance of each task type. The behavior of each task is designed by using a statechart, which is automatically converted in a C module that uses the event-driven architecture described in section 4.2.
The following 15 tasks are implemented in the ICU ASW:

1. ICU Controller: it is in charge of:
    - To Manage the unit and instrument operating modes and the status of the relative subsystem
    - To Configure internal ICU devices and the subsystems
    - To Execute all the TCs from SVM
2. SVM TM Handler: The main functionality of this task is to non-scientific TM packets to the SVM. It checks periodically if a new TM has to be sent to the SVM (i.e. there is a message queued in the buffer), and if at least a message is found in the relative buffer then it prepares and sends the TM. Messages in the buffer are ordered by priority. Priority depends on the type of telemetry
3. SVM TC Handler: Receives telecommands by checking periodically if a new TC is in the buffer, and if a TC is found then checks and accepts the new telecommand. A new TC can be received from the SVM or can be a telecommand that has been queued locally (e.g., a telecommand generated by an onboard control procedure). If the command is accepted, it forwards the TC to the task that will execute the telecommand.
4. DPS TM Handler: This task handles the TMs coming from all the DPS systems and it oversees:
    - Checking the presence of new TM packets from DPS units
    - Transferring TM packets to the local memory
    - Checking the PUS packet type by unit and service. If it is a science packet then it is buffered in the science data buffer which contain all science data to be compressed, otherwise it is buffered in the SVM TM Handler buffer

5. DPS TC Handler: The main functionality of this task is to buffer telecommands that have to be sent to the N-DPUs or to F-DPUs, and to send queued telecommand the unit
6. Housekeeping Handler: The main functionalities of this task are
    - To read and update HK of ICU (HW and SW), RDCU, N-AEU, F-AEU, MEU-Router and MEU-PSU. Updated HKs are stored in the relative Data Pool
    - To report all HKs to be sent to the SVM periodically
    - To generate the hearthbeat to the SVM
7. Limit monitoring: The main functionalities of this task are:
    - To check the ICU internal HK (both HW and SW).
    - To check HKs of RMAP subsystems (RDCU, FAEUs, NAEUs, MEU-Router and MEU-PSU)
    - If the HK values are out-of-limit, then the task oversees executing the relative FDIR
8. Event Action: this task handles the event action service to react promptly in case autonomous actions are needed by the ICU when a particular event occurs. When the event happens the task will start the execution of a specific telecommand by queueing it in the SVM TC Handler buffer.
9. Memory Handler: it is in charge of executing telecommands that foreseen the load, dump or checking of a portion of a memory (volatile or non-volatile)
10. Watchdog Handler: resets the watchdog periodically and executes FDIR actions relative to EDAC errors
11. Idle: puts in idle the CPU and execute the scrubbing of the local memory
12. Science Data Handler: processes science data to be compressed by using the RDCU HW compressor. It is in charge of copying science data to the compressor and commanding the start of the HW compression.
13. Science Data SW compressor: processes science data to be compressed by using the SW compressor. Once the compression is completed, transfers the compressed data to the SVM MMU
14. Science Data transfer: waits for the interrupt of "compression completed" which is received by the RDCU HW compressor, when a HW compression is completed. When the interrupt is received, then the compressed data is copied in the local memory to be packetized and transferred to the SVM MMU
15. OBCP: handles all onboard control procedure to be executed. It checks periodically if there is at least an active onboard procedure to be executed.

In the following table is reported the priority, type (periodic or sporadic) and deadline of each task.

Components Name	Task Name	Priority	Type	Deadline
ErrWatchdog	IEWD	10	Periodic	0.05s (WD kick at 1Hz)
SVM_TM_Handler	STMH	17	Periodic	0.02s
SVM_TC_Handler	STCH	50	Periodic	0.1s
DPS_TC_Handler	DTCH	50	Periodic	0.1s
DPS_TM_Handler	DTMH	16	Periodic	0.016s
HousekeepingHandler	IHKH	55	Periodic	0.125s
LimitMonitoringHandler	ILMH	55	Periodic	0.125s
EventActionHandler	IEAH	90	Sporadic	1s
ICU_ASW_Controller	ICTR	95	Sporadic	Max TC execution
ScienceDataHandler	ISDH	99	Sporadic	2.5s
IdleTask	IDLE	250	Periodic	15m
ObcpHandler	OBDH	150	Periodic	1s
MemoryHadler	MEMH	200	Sporadic	10s
ScienceSwDataCompressor	SSWC	99	Periodic	2.5s
ScienceDataTransfer	SDTR	100	Sporadic	0.7

6. SCIENCE DATA ACQUISITION AND PROCESSING

One of the most challenging task of the ICU ASW is the compression of imagette science data. The ICU ASW receives scientific data from 24 cameras (science data cargo is copied in a PUS packet) periodically with a period of 25s. The total amount that is received from all cameras is around 48MB. This means that the ICU ASW shall compress 48MB in less than 25s.

The RDCU HW compressor uses an adaptive algorithm that takes in to account the data of the previous compressed chunk. Therefore, the HW compressor needs to have in memory the so called "model of compressed data", which is updated at the end of each compression. Since the RDCU compressor has a very limited buffer memory (8MB), a maximum of a camera data can be compressed at a time.

The ICU acquires uncompressed science data from cameras and the received data is copied in 4x24 CPU memory buffers. Each buffer contains the data received during a single frame transfer, that is 25s. and it can contain up to 2MB of science data (the expected maximum is 27500 imagettes, corresponding to 1.98MB).

For simplicity, the four buffer are named Buffer-A[1..24], Buffer-B[1..24], Buffer-C[1..24] and Buffer-D[1..24]. For example Buffer-A[1] is the buffer for NCAM-1. Each buffer contains the data received during a time frame/slot of 25s . The received data during the first frame is stored in Buffer-A, then during the second frame in Buffer-B, successively the acquired data during the third frame is stored in Buffer-C and the data received in the last frame is saved in Buffer-D. Then, cyclically, the data is stored in Buffer-A, Buffer-B and so on.

The ICU-ASW starts the compression activity only after the Buffer-A and Buffer-B are filled at the end of the two 25s slots (see Figure 8). During the compression activity, data of Buffer-A is compressed and soon after data contained in Buffer-B is compressed (data relative to a camera at a time).

The current implementation avoids retrieving the model each time a compression is completed, in order to minimize the number of large transfers between the ICU CPU memory and the RDCU SRAM compressor memory. By the way, it is necessary to store 2 complete sets of imagettes for each camera. First, the imagettes from the first camera in Buffer-A and its model is sent to the RDCU and processed. In the next step, the metadata and the compressed bitstream are downloaded from the RDCU but not the updated model. Now the same chunk but from the Buffer-B set is sent to the RDCU. An upload of the model is not necessary because it is already in the RDCU SRAM. Now the 2nd chunk can be compressed. After the compression the bitstream and now also the updated model will be downloaded from the RDCU. The updated model is needed to compress the same chunk from the Buffer-C set. Then the process starts from the beginning and the next part of the Buffer-C set can be compressed.

By this procedure, an upload and download of the model can be saved.

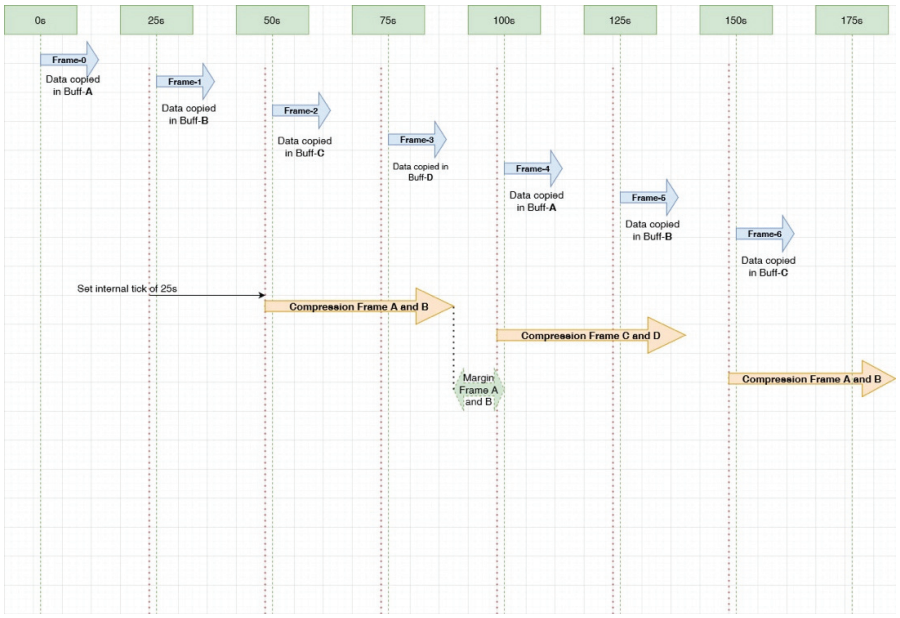

Figure 8. CPU Data Buffer Handling

7. SCHEDULABILITY ANALYSIS

In order to verify that all tasks complete their process before the deadline, a schedulability analysis is conducted for each milestone/version of the ICU ASW.
The analysis is conducted by using the TASTE tool (Ref[13]). TASTE is a set of freely-available tools dedicated to the development of embedded, real-time systems. It is developed by the European Space Agency together with a set of partners from the space industry.
TASTE promotes the combined use of formal description techniques (AADL, SDL, MSC, ASN.1) with strongly typed programming languages (Ada) and other system-level modelling tools (Simulink, VHDL) and coding languages (C, C++). TASTE addresses system architecture, data and behavior modelling and allows to generate low-level code for micro-controllers and distributed, communicating systems. Various platforms are supported, included Leon2/Leon3 with RTEMS as well as targets including FPGA components. It is open and extensible.
TASTE uses Cheddar as a modelling tool. Cheddar is an Ada framework that provides services to study the temporal behaviour of realtime applications and to perform a scheduling simulation analysis. In Cheddar, an application is defined by a set of processors, buffers, shared resources, messages, and tasks. In the simplest task model, each task periodically performs a treatment.

7.1 ICU ASW simulation by using the TASTE Interface View

The TASTE tool provides and Interface View to capture the system logical architecture, in terms of functions connected through their interfaces. The Interface View is used to generate all tasks and their interfaces of the ICU ASW as reported in Figure 9. The worst execution tie to be set for each interface is calculated on the real environment by using unit tests, logs and reported telemetries.

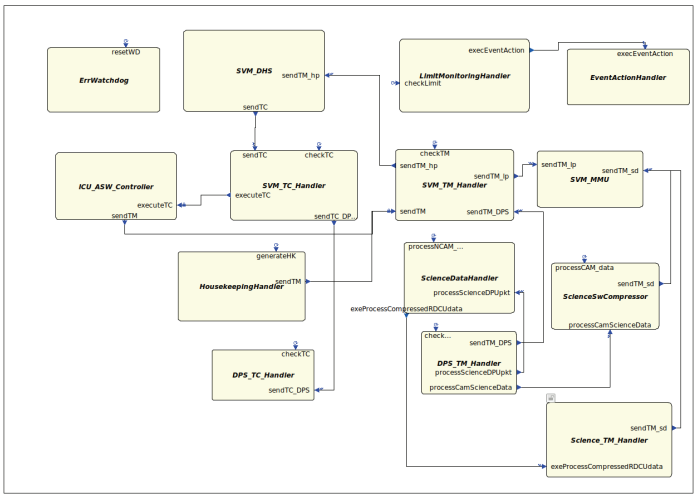

Figure 9. The ICU-ASW model in the TASTE Interface View editor. The yellow boxes represent the functions; the arrows model the provided interfaces (in) and the required interfaces (out).

7.2 ICU ASW deployment by using the TASTE Deployment View

The so-called “TASTE Deployment View” describes the hardware environment on which the software application defined by the Interface View can run. Each function defined in the Interface view can be connected to a specific hardware node, to define in which node is executed. If two or more nodes are connected by means a bus/link, it can also be simulated.

In the case of the ICU-ASW, two nodes are connected each other by means of Spacewire links, one for the MMU and one for the DHS, as shown in Figure 10.

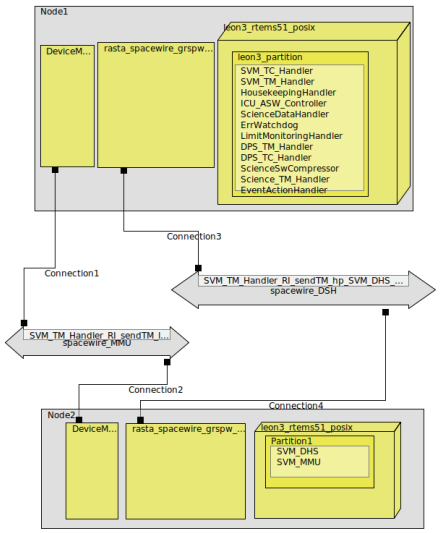

Figure 10. The ICU-ASW model in the TASTE Development View editor

7.3 ICU ASW schedulability analysis results

In the so-called “TASTE Concurrency View” is possible to starts the scheduling analysis by building and binding all the objects and interfaces that compose the interface and deployment models. For this analysis the Cheddar tool is used

The Cheddar analysis tool allows to obtain different analysis results:

- A synthesis table which displays the theoretical and simulation results.
- A simulation chronogram produced by Cheddar.
- A more complete theoretical results from Cheddar.
- A more complete simulation results from Cheddar.

The simulation has shown that all tasks are executed within the deadline. In the figure below the simulation result is reported, showing that all tasks are completed within the define timeout.

Figure 11. The Cheddar chronogram

8. RESULTS

The V model has been used to deliver the first version of the ICU ASW (0.4 and 0.8). The 0.8 is part of the Critical Design Review which is planned to be at the end of August 2022.

The 0.8 will be used also to validate the science data processing which is the most challenging task. Preliminary test results were conducted on 6/24 CAMs. The average CPU occupation when the ICU is in STANBY mode is 10%. A dummy task with same priority of tasks involved in the hardware compression activity load has been added to the ICU-ASW to simulate a CPU occupation of 30%: with 6 N-CAMS the maximum CPU occupation is around 45% resulting in a science load of not more of 5% (due basically to TM packetization of compressed science data and copy of received science packets in the local memory).

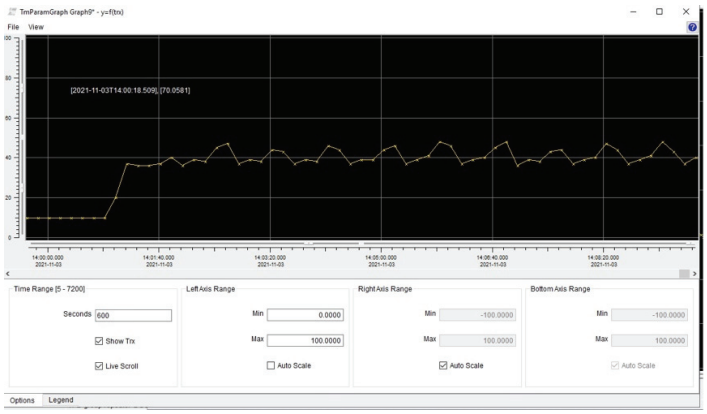

Figure 12. CPU occupation during 6 N-CAMs compression test

Since it was not possible to execute a test with 24 NCAMs due to some issues with the EGSE, a preliminary trend analysis has been conducted to verify that there is sufficient margin also for this case. Considering that 4 input buffers (each buffer stores data coming from all cameras received during a slot of 25s) have been configured, each buffer will be overwritten with new data every 100s (see also Figure 8). Since for the first 50s the processing of science data cannot start due to the optimization of model transfers between the ICU ASW and the RDCU, then only 50s per two frames are taken into account. Therefore, the data stored in the local memory must be, at least, copied in the RDCU memory before new data is received.

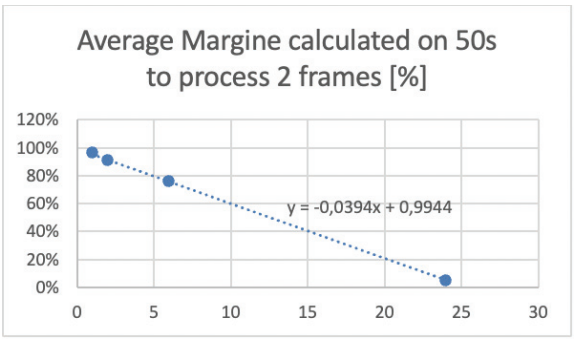

Figure 13. Average Margin calculated on 50s to process two frames (Y= % of Margin, X=#N-CAMs)

In Figure 13 the calculated average margin is reported. Of course, when the number of cameras is low, the margin is very high (>70%). The calculated trend is linear and can be fit with a linear function with a slope of -0.0394 and an intercept of 0.9944. It has been calculated that with 24 N-CAMs a margin of 4.9% is expected.

9. ACKNOWLEDGMENTS

A special acknowledgement to the European Space Agency (ESA) and DLR/DPS Team leading the PLATO Mission Consortium and supporting the ICU Team during each phase of the project.

This work has been funded thanks to the Italian Space Agency (ASI) support to the Project, as defined by the ASI-INAF agreement n. 2015-019-R.O. “Scientific activity for the PLATO Mission - B/C Phases”.

REFERENCES

[1] Rauer, H. et al., “The PLATO 2.0 Mission,” in [Experimental Astronomy, vol. 38, pp. 249–330], (2014).
[2] Giusi, G. et al., “The PLATO instrument control unit software: From unit requirements to sw architecture,” in [Proc. SPIE 11443-51], (2020).
[3] Cosentino, R. et al., “The instrument control unit of the plato payload: design consolidation following the preliminary design review by ESA,” in [Proc. SPIE 114434V], (2020).
[4] Focardi, M. et al., “The design of the instrument control unit and its role within the data processing system of the ESA PLATO mission,” in [Proc. SPIE AS101-61], (2018).
[5] ECSS-E-ST-40C, “Space Engineering-Software," (March 2009)
[6] ECSS-Q-ST-80C, “Software product assurance”, Rev. 1, 15 February 2017
[7] ECSS-E-ST-50-52C, “ SpaceWire – Remote memory access protocol”, 5 February 2010
[8] ECSS-E-ST-70-41C, “Telemetry and telecommand packet utilization”, 15 April 2016
[9] IBM “Ration Rhapsody family”, https://www.ibm.com/products/systems-design-rhapsody?mhsrc=ibmsearch_a&mhq=rational%20rhapsody
[10] IBM Ration DOORS, https://www.ibm.com/docs/it/ermd/9.5?topic=doors-overview-rational
[11] C and C++ Unit Testing Automation VectorCAST, https://www.vector.com/it/it/prodotti/products-a-z/software/vectorcast/vectorcast-c/
[12] TSC and CCS TERM tools, https://www.terma.com/markets/space/space-segment/central-checkout-systems
[13] TASTE tool, https://taste.tools/