HERSCHEL

SPIRE On Board Software Software Specification Document

Document Ref.: SPIRE-IFS-PRJ-001036

Issue: 1.1

Prepared by: Sergio Molinari
Riccardo Cerulli-Irelli

Approved by: Renato Orfei
Ken J. King
Bruce M. Swinyard

1 Introduction

1.1 Purpose of the Document

This document describes the Architecture Design that led to the generation of the SPIRE On-Board Software. The OBS runs under the VIRTUOSO Operating System, which is designed for Real-Time DSP applications. We will first describe the main features of VIRTUOSO kernel services that are used in the OBS: Tasks, Semaphores, FIFO Message Queues, Events and Memory Pools. We will then describe the implementation of the on-board memory management. Finally, we will describe the OBS applicative by a series of Architecture Diagrams where the OBS is broken down into the individual tasks; each task is then decomposed into modules. Each diagram module maps one, or a group, of modules in the OBS code. Blocks and modules will be described in detail, enhancing the design features that implement the various requirements in the URD AD7.

The DPU Switch-on and Boot procedure is not implemented as part of the OBS, but it is implemented as a separate entity stored on a PROM. See RD8 for details.

1.2 Acronyms

ACE	1553 Advance Computing Engine
AOT	Astronomical Observation Template
APID	Application Identifier
CASE	Computer Aided Software Engineering
CDMS	Command and Data Management System
CNR	Consiglio Nazionale delle Ricerche
CPU	Control Processing Unit
DPU	Digital Processing Unit
DRCU	Detector Readout and Control Unit
EEPROM	Electrically Erasable Programmable Read Only Memory
FCU	FPU Control Unit
HERSCHEL	Far InfraRed and Submillimeter Telescope
FOV	Field Of View
FPU	Focal Plane Unit
FTS	Fourier Transform Spectrometer
HIFI	Heterodyne Instrument for HERSCHEL
HK	HouseKeeping
HS	High Speed
HW	HardWare
ICC	Instrument Control Centre
ICS	Instrument Command Sequence
IFSI	Istituto di Fisica dello Spazio Interplanetario
MCU	Mechanical Control unit
MOC	Mission Operations Centre
OBS	On Board Software
OIRD	Operations Interface Requirements Document
PACS	Photoconductor Array Camera and Spectrometer

PROM	Programmable Read Only Memory
RAM	Random Access Memory
ROM	Read Only Memory
SA	1553 DPRAM SubAddress
SPIRE	Spectral and Photometric Imaging Receiver
SW	SoftWare
TAI	Temps Atomique International
TBC	To Be Confirmed
TBD	To Be Defined
TBW	To Be Written
TC	TeleCommand
TM	TeleMetry
UR	User Requirement
URD	UR Document
WE	Warm Electronics

1.3 References

1.3.1 Applicable Documents

Document Reference	Name	Number
AD1	FIRST/Planck Instrument Interface Document Part A	PT-IIDA-04624
AD2	FIRST/Planck Instrument Interface Document Part B Instrument "SPIRE"	SCI-PT-IIDB
AD3	FIRST/PLANCK Operations Interface Requirements Document	SCI-PT-RS-07360
AD4	FIRST/PLANCK Packet Structure Interface Control Document	SCI-PT-IF-07527
AD5	FIRST Instrument Commanding Concepts	
AD6	Operating Modes for the SPIRE Instruments	SPIRE-RAL-DOC-000320
AD7	SPIRE OBS User Requirement Document	SPIRE-IFS-PRJ-000444
AD8	FIRST SPIRE Electrical Interface Control Document	SPIRE-Sap-Cca-24-00
AD9	SPIRE Data Interface Control Document	SPIRE-RAL-DOC-001078
AD10	SPIRE DRCU/DPU Interface Control Document	SPIRE-SAp-PRJ-001324

1.3.2 Reference Documents

Document Reference	Name	Number
RD1	Guide to applying the ESA software engineering standards to small software projects	BSSC(96)2
RD2	FIRST SPIRE DPU subsystem specification document	
RD3	FIRST SPIRE DPU-DRCU Interfaces	SP-RCI-5.7.00

RD4	Telemetry and Telecommand Packet Utilisation Standard	ECSS-E-70/41
RD5	Herschel/Planck Instrument Data Rates	H-P-1-ASPI-TN-0204
RD6	SPIRE DPU Virtual Machine	
RD7	SPIRE OBS User Manual	
RD8	DPU Boot Software Architectural Design	DPU-AD-CGS-001
RD9	VIRTUOSO User's Guide for ADSP-21020	

1.4 Document Change Record

Issue	Revision	Date	Reason for Change
0	2	18/05/2001	First draft. The document consists of the Software specifications that are common to the three instruments.
0	9	17/04/2002	Added a quite general version of the OBS Logical Model, mostly mutuated from HIFI. Also added a first draft of a SPIRE-specific architecture design and module description
1	0	18/05/2003	Complete rewrite. Logical Model and Software specifications removed. Architecture design description has been updated and greatly enhanced.
1	1	15/08/2004	Added Software Requirements section aligned with version 1.2.j of the OBS. Design description aligned with OBS version 1.2.j; it also includes features (monitoring, autonomy) that are not in 1.2.j but that will be implemented according to this design.

2 Software Requirements

2.1 Initialization and Configuration Requirements

ID	Requirement	Related UR	Design
SP-SR-IN1	SPIRE will act as a Remote Terminal		INIT Task (§3.2.6.1)
SP-SR-IN2	The OBS shall support Mode Commands as shown in Table~3.2.4-1 of AD4 (3045-DLL)		
SP-SR-IN3	The OBS shall support the subaddress (SA) allocation shown in Table 3.2.3-1 of AD4 (3050-DLL)		
SP-SR-IN4	The OBS shall implement the SA utilization Table~3.2.3-1 of AD4 (3135-DLL-R,T, 3140-DLL)		
SP-SR-IN5	The OBS shall use SA 0R for mode command (3145-DLL)		
SP-SR-IN6	The OBS shall support the mode commands listed in Table~3.2.4-1 of AD4 (3250-DLL)		
SP-SR-IN7	The OBS shall use SA 1T to transmit instrument status (3150-DLL)		
SP-SR-IN8	The OBS shall use SA 8R to receive spacecraft time (3180-DLL)		
SP-SR-IN9	The OBS shall use SA 10T to inform spacecraft that a new telemetry packet is ready (3185-DLL-T)		
SP-SR-IN10	The OBS shall use SA 11--26T to transfer telemetry packets from instrument to spacecraft (3195-DLL)		
SP-SR-IN11	The OBS shall use SA 11--14R to transfer telecommand packets from spacecraft (3200-DLL)		
SP-SR-IN12	The OBS shall use SA 27R to prepare instrument for telecommand transfer (3205-DLL)		
SP-SR-IN13	The OBS shall place in SA 27T, after reading the telecommand packet, the confirmation message (3210-DLL)		
SP-SR-IN14	The OBS shall use SA 30T (Data Wrap read) for test purposes (3235-DLL)		
SP-SR-IN15	The OBS shall use SA 30R (Data Wrap write) for test purposes (3240-DLL)		
SP-SR-IN16	The OBS will read the 1553 configuration register and store in memory the value of the RT address		
SP-SR-IN17	The OBS will configure SA11-27T as circular buffers with a size of 128 words		
SP-SR-IN18	The 1553 I/F will be configured to issue an interrupt signal upon reception of the Synchronize with and without data word mode command		

2.2 Spacecraft Interface Requirements

ID	Requirement	Related UR	Design
SP-SR-SC1	The OBS shall support a cyclic satellite Data Bus Protocol based on a 1 second period called Frame, divided into 64 subframes, each containing a number of Mil Std 1553B messages (4105-TFL, 4120-TFL)	UR-TC3	TMTC (§3.2.6.3) ISR1553 (§3.2.5.1)
SP-SR-SC2	Each packet transfer shall be controlled by the exchange of a Packet Transfer Request/Descriptor and a Packet Transfer Confirmation, providing the necessary (handshake) information about the transfer (4195-TFL)		ISR1553 (§3.2.5.1)
SP-SR-SC3	The OBS shall check the status of the packet transfer that has taken place in the previous Subframe, within the receiving of the next Subframe Sync Message, at the latest (4200-TFL)		ISR1553 (§3.2.5.1)
SP-SR-SC4	If a packet transfer has been performed then the RT shall update the TM packet data buffer within 2 msec, and shall update the TM packet Packet Transfer Request Words within 2 msec (4205-TFL, 4210-TFL)		TMTC (§3.2.6.3) ISR1553 (§3.2.5.1)
SP-SR-SC5	Only one TM packet transfer from each RT at a time is allowed. If there is more than one packet to be sent the RT shall queue the TM packets (4220-TFL)		TMTC (§3.2.6.3)
SP-SR-SC6	TM packets shall be transferred within one Subframe (4230-TFL)		TMTC (§3.2.6.3)
SP-SR-SC7	The OBS shall support Packet Transfer Requests via SA 10T and Packet Transfer Descriptors via SA 27R (4240-TFL)		ISR1553 (§3.2.5.1)
SP-SR-SC8	In case of TM packets the OBS shall provide the following parameters with these words: i)The number of needed messages, and ii) The number of words in the last message (4245-TFL)		TMTC (§3.2.6.3) ISR1553 (§3.2.5.1)
SP-SR-SC9	For TC packets the above parameters are provided by the BC. The OBS shall utilize these parameters to re-assemble the TC packets (4250-TFL)		TMTC (§3.2.6.3) ISR1553 (§3.2.5.1)
SP-SR-SC10	The OBS shall read the RT address in the Subframe User field (see Figure~4.2-1 of AD4) (4275-TFL)		ISR1553 (§3.2.5.1)
SP-SR-SC11	The OBS shall support an internal Subframe Counter and shall provide the value for BC access (4295-TFL)		ISR1553 (§3.2.5.1)
SP-SR-SC12	When receiving the first Subframe each second the Subframe Counter shall be set to 0 and the OBS shall increment this value by one with every received Sync with Data Word command (4300-TFL)		ISR1553 (§3.2.5.1)
SP-SR-SC13	The OBS shall copy the Time Message to SA 8T immediately after receiving the Mode Command Synchronize at the beginning of a frame. At initialization, before receiving any valid Time		TIME (§3.2.6.2)

	Distribution Message, the OBS shall set the buffer at SA 8T to zero (4345-TFL)		
SP-SR-SC14	The RT status information shall be available via SA using the layout shown in Figure~4.4-1 of AD4 (4355-TFL, 4360-TFL)		ISR1553 (§3.2.5.1)
SP-SR-SC15	The TC packet shall be received by OBS in the TC Data receive SAs, beginning with SA 11R (4415-TFL)		TMTC (§3.2.6.3)
SP-SR-SC16	The Packet Transfer Descriptor shall be received by OBS in the SA 27R, according to the layout shown in Table~4.5.1-1 of AD4 (4420-TFL, 4421-TFL)		ISR1553 (§3.2.5.1)
SP-SR-SC17	The OBS shall evaluate the TC Packet Transfer Descriptor after the reception of the next Subframe Sync, within one Subframe (4425-TFL)	UR-TC12	ISR1553 (§3.2.5.1)
SP-SR-SC18	The OBS shall store the new TC packet immediately and copy the associated words of the Packet Transfer Descriptor to SA 27T, to become the TC Packet Confirmation, according to the layout shown in Table~4.5.1-2 of AD4 (4430-TFL)	UR-TC12	TMTC (§3.2.6.3)
SP-SR-SC19	To receive a TC packet the RT shall adopt the procedure shown in Figure~4.5.1-1 of AD4 (4435-TFL)		TMTC (§3.2.6.3) ISR1553 (§3.2.5.1)
SP-SR-SC20	If the packet counter contained in a valid TC PTD will be different from the counter of the PTD of the previous TC +1, a TM (5,1) event will be generated to signal the anomaly.	UR-TC21	TMTC (§3.2.6.3)
SP-SR-SC21	The layout of the Packet Transfer Request shall be in accordance with Table~4.6.1.1-1 of AD4 where: i. reserved bits shall be set to zero (4505-TFL) ii. No. of messages for next packet shall indicate the number of messages needed for the packet the OBS is intending to send in the next Subframe. The first message of a TM packet shall always stored at SA 11T (4510-TFL) iii. No. of Data Words shall indicate the number of data words transmitted in the last message. In case of 32 words this field shall be set to 00000B (4515-TFL) iv. since data packets have always a size of n times 16 bit, with n an even number, no filling area shall be foreseen (4520-TFL) v. Event fields A and B shall be set to 0 (no Event message pending) (4525-TFL, 4535-TFL) vi. the Burst Mode field shall be set to 0 (Nominal Mode) (4545-TFL, 4550-TFL) vii. Flow Control field shall be set by RT according to the status of TM transfer immediately. Its value shall be: 00B (No transfer pending), 01B (Transfer		TMTC (§3.2.6.3) ISR1553 (§3.2.5.1)

	is pending) (4555-TFL, 4560-TFL, 4565-TFL, 4570-TFL. 4572-TFL) viii. Packet Count field shall be used to support a OBS-generated counter. To avoid that after an OBS initialisation or reset an identical packet number is used, there shall be one number foreseen for that case. This number shall never appear in the cyclical transmission (4575-TFL)		
SP-SR-SC22	The OBS shall support a circular Packet Counter in the range 1 to 255 decimal (4585-TFL, 4590-TFL)		ISR1553 (§3.2.5.1)
SP-SR-SC23	After initialization or restart the OBS shall set the counter value to 0 for the first TM Packet Transfer (4595-TFL)		ISR1553 (§3.2.5.1)
SP-SR-SC24	The OBS shall not use this counter for any other purpose than defined in Chapter~4.6 of AD4 (4600-TFL)		TMTC (§3.2.6.3) ISR1553 (§3.2.5.1)
SP-SR-SC25	After requesting a TM packet transfer, the RT shall determine if the packet transfer was performed via the handshake signal (TM Packet Confirmation) sent by BC (4265-TFL)		ISR1553 (§3.2.5.1)
SP-SR-SC26	The OBS shall receive the Packet Confirmation on SA 10R according to the layout shown in Figure~4.6.1.2-1 of AD4 (4635-TFL, 4640-TFL)		ISR1553 (§3.2.5.1)
SP-SR-SC27	The OBS shall request a TM packet transfer (RT to BC) by setting its TM Packet transfer control words (SA 10T) (4685-TFL)		TMTC (§3.2.6.3) ISR1553 (§3.2.5.1)
SP-SR-SC28	For the exchange of TM packets in normal data bus mode, the RT shall support the logic shown in Figure~4.6.1.3-1 of AD4 (4690-TFL)	UR-TM2	TMTC (§3.2.6.3) ISR1553 (§3.2.5.1)
SP-SR-SC29	After a TM packet transfer, in case there is no new TM packet pending the RT shall set the first word of the TM Packet Transfer Request to 0000 0000B, and the Packet Count value of the second word shall stay unchanged. The Flow Control field bits shall be set to 00B (4695-TFL)		ISR1553 (§3.2.5.1)
SP-SR-SC30	The OBS shall generate TM-packets with a maximum size of 1024 octets (4020-TFL)		INIT (§3.2.6.1) TMTC (§3.2.6.3) HS (§3.2.6.7) HK_ASK (§3.2.6.6)
SP-SR-SC31	The OBS shall support the exchange of variable length packets (4030-TFL)		TMTC (§3.2.6.3)

			ISR1553 (§3.2.5.1)
SP-SR-SC32	A new TM packet will be loaded on the 1553 DPRAM only if there is free space available		TMTC (§3.2.6.3)
SP-SR-SC33	The OBS will maintain a circular buffer containing the TM Packet Transfer Requests.		ISR1553 (§3.2.5.1)
SP-SR-SC34	A new TM PTR will be added to the TM PTR circular buffer when a new TM packet has been copied from the memory pools into SA11-27T of the 1553 DPRAM		ISR1553 (§3.2.5.1)

2.3 Telecommand Requirements

ID	Requirement	Related UR	Design
SP-SR-TC1	The OBS shall accept TC-packets with a maximum size of 248 octets (4025-TFL)	UR-TC3	INIT (§3.2.6.1)
SP-SR-TC2	There will be only two immediate commands: "Abort VM" and "Abort Memory Dump"	UR-TC6 UR-TC18	TMTC (§3.2.6.3) CMD_SEQ (§3.2.6.4)
SP-SR-TC3	The interpretation and execution of immediate commands will take precedence over normal commands.	UR-TC4 UR-TC12 UR-TC6 UR-TC7	CMD_SEQ (§3.2.6.4)
SP-SR-TC4	The OBS will check that the APID of the TC is legal. In case of failure a TM (1,2) will be sent with codes and parameters as per PSICD.	UR-TC8 UR-TC11	CMD_SEQ (§3.2.6.4)
SP-SR-TC5	The OBS will check that the packet length computed from the number of 1553 data words contained in the TC PTD is consistent with the length parameter in the TC packet header. In case of failure a TM (1,2) will be sent with codes and parameters as per PSICD.	UR-TC8 UR-TC11	CMD_SEQ (§3.2.6.4)
SP-SR-TC6	The OBS will implement the CRC checksum algorithm to compute the CRC on the incoming TC and compare it to the CRC word at the end of the TC packet. In case of failure a TM (1,2) will be sent with codes and parameters as per PSICD.	UR-TC8 UR-TC11	CMD_SEQ (§3.2.6.4)
SP-SR-TC7	The OBS will check that the packet type is a valid one. In case of failure a TM (1,2) will be sent with codes and parameters as per PSICD.	UR-TC8 UR-TC11	CMD_SEQ (§3.2.6.4)
SP-SR-TC8	The OBS will check that the packet subtype is among the valid subtypes for that type. In case of failure a TM (1,2) will be sent with codes and parameters as per PSICD.	UR-TC8 UR-TC11	CMD_SEQ (§3.2.6.4)
SP-SR-TC9	The OBS will support all the services of AD4, with the exceptions listed in AD9.	UR-TC20	CMD_SEQ (§3.2.6.4)
SP-SR-TC10	The OBS will accept and execute all commands	UR-TC1	CMD_SEQ

	specified in AD9	UR-GE3-5	(§3.2.6.4)
SP-SR-TC11	Once a TC has been successfully verified its contents will be checked for executability. In case the TC contains inconsistent or incorrect parameters for that particular packet type and subtype, a TM (1,8) TC execution failure will be generated containing all the information needed to identify the occurred problem.	UR-TC10 UR-TC14 UR-TC16	CMD_SEQ (§3.2.6.4)

2.4 Telemetry Requirements

ID	Requirement	Related UR	Design
SP-SR-TM1	The OBS shall be able to generate all TM packets specified in AD9.	UR-TM1	TMTC (§3.2.6.3) HS (§3.2.6.7) HK_ASK (§3.2.6.6)
SP-SR-TM2	The generation of TM (1,1) (1,3) (1,5) (1,7) will be carried out according to the “ACK” bits contained in the header of the related telecommand.	UR-TC5 UR-TC15	CMD_SEQ (§3.2.6.4)
SP-SR-TM3	It will be possible to simultaneously run 4 independent HK collection tasks in the OBS	UR-TM14 UR-TM15 UR-TM17 UR-GE3-5 UR-GE13	HK_ASK (§3.2.6.6)
SP-SR-TM4	The nominal HK collection task will record the time of start HK collection at each periodic activation, and include this time as a parameter in the HK packet.	UR-TM11	HK_ASK (§3.2.6.6)
SP-SR-TM5	The list of HK parameters to be collected will be contained in an on-board table	UR-TM9 UR-TM12	HK_ASK (§3.2.6.6)
SP-SR-TM6	Each running HK packet collection task will be associated to one and only one packet definition table	UR-TM12	HK_ASK (§3.2.6.6)
SP-SR-TM7	The HK packet definition will contain the list of commands needed to get the parameters	UR-TM13	HK_ASK (§3.2.6.6)
SP-SR-TM8	DPU internal parameters will be collected using commands that implement the same syntax as for the S/S commands	UR-TM9 UR-TM10	HK_ASK (§3.2.6.6)
SP-SR-TM9	The location of an HK parameter in an HK packet will be defined by the ordinal location of the related command in the HK packet definition table	UR-TM13	HK_ASK (§3.2.6.6)
SP-SR-TM10	The HK collection will be periodic, with a sampling interval that can be configured via TC (3,1) as per AD4.	UR-TM16	HK_ASK (§3.2.6.6)
SP-SR-TM11	The HK collection will be handled by tasks that will be activated by Events triggered by VIRTUOSO timers preset to the periodicity equal to the chosen sampling	UR-TM16	HK_ASK §3.2.6.6

	rate.		
SP-SR-TM12	It will not be possible to change the HK packet definition table for a running HK collection task.		CMD_SEQ (§3.2.6.4)
SP-SR-TM13	The HK task activation/deactivation will be done via TC as specified in AD4.	UR-SM9	CMD_SEQ (§3.2.6.4)
SP-SR-TM14	The OBS will use separate buffers in the memory pools to hold science frames for each different frame ID	UR-GE8-10	(§3.2.4.2)
SP-SR-TM15	Each science buffer in the memory pools will contain as many science frames (of a specific frame ID) as a TM packet can hold	UR-TM18	HS (§3.2.6.7)
SP-SR-TM16	A science buffer that cannot contain any more frames for that specific ID will be considered complete, and a new science buffer will be created in the memory pools		HS (§3.2.6.7)
SP-SR-TM17	After a FIFO flush, all science buffers containing frames with IDs pertinent to the flushed FIFO will be considered complete.		HS (§3.2.6.7)
SP-SR-TM18	A HK buffer or complete science buffer will be integrated with a packet header compliant to AD9 and AD4 to become a TM packet	UR-TM4	TMTC (§3.2.6.3)
SP-SR-TM19	It will be possible to select, via TC, subsets of science frames to be copied into the science buffers	UR-TM7 UR-GE13	CMD_SEQ (§3.2.6.4) HS (§3.2.6.7)
SP-SR-TM20	The OBS shall be able to support a total output telemetry rate of 100 Kbps averaged on 24 hours		TMTC (§3.2.6.3) ISR1553 (§3.2.5.1)

2.5 Functional and Operational Requirements

ID	Requirement	Related UR	Design
SP-SR-FU1	The OBS will implement an on-board command interpreter called the Virtual Machine (VM) that will be able to execute commands at a predefined time.	UR-TC23 UR-SM10 UR-SM11 UR-FU1-9 UR-GE12	Hard_VM (§3.2.5.2) Soft_VM (§3.2.6.10) VM_SVC (§3.2.6.11)
SP-SR-FU2	The VM will be able to interpret and execute commands in the form of 32-bit words stored in tables on-board		Hard_VM (§3.2.5.2) Soft_VM (§3.2.6.10)
SP-SR-FU3	The VM will be able to send commands to the S/S via the LS interface	UR-FU10 UR-GE11	Hard_VM (§3.2.5.2) Soft_VM (§3.2.6.10)
SP-SR-FU4	The VM will be able to read the reply word sent by the	UR-FU10	Hard_VM

	S/S via the LS interface	UR-GE11	(§3.2.5.2) Soft_VM (§3.2.6.10)
SP-SR-FU5	The VM will be able to perform **for** cycles	UR-GE11	Hard_VM (§3.2.5.2) Soft_VM (§3.2.6.10)
SP-SR-FU6	The VM shall be able to perform conditional decisions based on the values of some parameters	UR-GE11	Hard_VM (§3.2.5.2) Soft_VM (§3.2.6.10)
SP-SR-FU7	The VM shall be able to read from/write to OBT	UR-GE11	Hard_VM (§3.2.5.2) Soft_VM (§3.2.6.10)
SP-SR-FU8	The VM will be able to generate TM (1,x) packets		Hard_VM (§3.2.5.2) Soft_VM (§3.2.6.10) VM_SVC (§3.2.6.11)
SP-SR-FU9	The VM will be able to generate TM (5,x) packets		Hard_VM (§3.2.5.2) Soft_VM (§3.2.6.10) VM_SVC (§3.2.6.11)
SP-SR-FU10	The VM will be able to generate TM (21,4) packets limited to SID TBD		Hard_VM (§3.2.5.2) Soft_VM (§3.2.6.10) VM_SVC (§3.2.6.11)
SP-SR-FU11	There will be 1 VMs where the command timing will be implemented by an HW interrupt line connected to a DPU timer. This VM will be called Hard_VM.		Hard_VM (§3.2.5.2)
SP-SR-FU12	There will be 3 VMs where the command timing will be implemented using task activation regulated by VIRTUOSO SW timers. These VMs will be called Soft_VMs.		Soft_VM (§3.2.6.10)
SP-SR-FU13	The execution of an observing procedure running as VM code on the Hard VM will be stoppable by disabling the DPU interrupt associated with the DPU HW timer. It will be possible to do this via TC (Abort Measurement)	UR-TC18	CMD_SEQ (§3.2.6.4)

2.6 Memory Management Requirements

ID	Requirement	Related UR	Design
SP-SR-MM1	The OBS will provide a protected DM memory area where tables of data can be defined (called On-Bard Tables – OBT)		§3.2.4.1
SP-SR-MM2	It will be possible to create, update and delete an OBT via TC	UR-TC19 UR-TM9 UR-TM12 UR-SM10 UR-SM11	CMD_SEQ (§3.2.6.4)
SP-SR-MM3	OBT will be used to store HK packet definitions and VM codes.	UR-TM12 UR-SM10 UR-SM11	CMD_SEQ (§3.2.6.4)
SP-SR-MM4	Each OBT will be characterized by an ordinal ID number and a length		§3.2.4.1
SP-SR-MM5	The OBS shall maintain an updated list of IDs and lengths for all currently defined OBTs		§3.2.4.1
SP-SR-MM6	The OBS will allow the relative allocation and de-allocation of OBTs	UR-TM12	CMD_SEQ (§3.2.6.4) + §3.2.4.1
SP-SR-MM7	The OBS will dynamically re-allocate OBTs to optimize memory occupation on board	UR-TM12	CMD_SEQ (§3.2.6.4) + §3.2.4.1
SP-SR-MM8	Dynamic OBT re-allocation will have no impact on the HK data collection or execution of VM code on the Soft_VM	UR-TM12	CMD_SEQ (§3.2.6.4) + §3.2.4.1
SP-SR-MM9	It will not be possible to dynamically re-allocate OBTs while the Hard_VM is running	UR-TM12	CMD_SEQ (§3.2.6.4) + §3.2.4.1
SP-SR-MM10	It will not be possible to modify or delete an OBT currently associated to a running HK collection task	UR-TM12	CMD_SEQ (§3.2.6.4)
SP-SR-MM11	It will not be possible to modify or delete an OBT that contains VM code currently being executed		CMD_SEQ (§3.2.6.4)
SP-SR-MM12	The OBS will use internal protected fixed-size memory areas configured as circular buffers (Memory Pools) to build TM packets.	UR-TM5	§3.2.4.2
SP-SR-MM13	There will be separate memory pools for events and report packets, housekeeping packets and science packets		§3.2.4.2
SP-SR-MM14	Memory blocks will be allocated in the memory pools only if there is sufficient space available in the pool	UR-TM5	CMD_SEQ (§3.2.6.4) HS (§3.2.6.7) HK_ASK

			(§3.2.6.6)
SP-SR-MM15	In case a memory block is requested and there is insufficient space in the memory pool a TM (5,1) will be generated to signal the anomaly	UR-TM5	CMD_SEQ (§3.2.6.4) HS (§3.2.6.7) HK_ASK (§3.2.6.6)
SP-SR-MM16	The exchange of TM packets between OBS tasks will be done by passing the pointer to the packet and not the packet itself.		
SP-SR-MM17	All information concerning TM packets within the DPU memory, including its location, will be passed from one task to the other using VIRTUOSO FIFO queues.		§3.2.3.3
SP-SR-MM18	After a TM packet has been copied from the relevant buffer of the memory pools into the 1553 DPRAM, the buffer will be released		TMTC (§3.2.6.3)
SP-SR-MM19	The OBS will implement two memory pools for Telecommands (TCs): one for normal commands and a higher priority one for immediate commands	UR-TC6 UR-TC7	§3.2.4.2
SP-SR-MM20	The exchange of TC packets between OBS tasks will be done by passing the pointer to the packet and not the packet itself.		
SP-SR-MM21	All information concerning TC packets within the DPU memory, including its location, will be passed from one task to the other using VIRTUOSO FIFO queues.		§3.2.3.3
SP-SR-MM22	The message enqueued on the VIRTUOSO FIFO with the pointer of a new TC in the memory pools will also contain the length of the packet as computed from the TC PTD		TMTC (§3.2.6.3)

2.7 Subsystem Interface Requirements

ID	Requirement	Related UR	Design
SP-SR-SS1	The OBS will support the syntax specified in AD10 to send commands to the S/S	UR-TC14	LS (§3.2.6.5)
SP-SR-SS2	In case of a “Sync DRCU Timers” S/S command, the time when the command is actually written onto the LS port is recorded and made available in a global variable so that it can be used as an HK parameter.	UR-SY3	LS (§3.2.6.5)
SP-SR-SS3	Commands will be sent by the OBS to the S/Ss using a single SW interface. The only exception is the VM activated by the HW DPU timer that will send the command directly writing onto the HW interface		LS (§3.2.6.5)
SP-SR-SS4	The OBS will wait 2 msec after sending a command through the LS port before reading the response word		LS (§3.2.6.5)

SP-SR-SS5	The OBS will check that the command ID in the response word is identical to the command ID of the command word sent. In case of failure a TM (5,1) event will be generated to signal the anomaly and the OBS will assume that the command was not correctly received or executed by the S/S		LS (§3.2.6.5)
SP-SR-SS6	The OBS will check that the "Ack bits" in the reply word are "00". In case of failure that a TM (5,1) event will be generated to signal the anomaly and the OBS will assume that the command was not correctly received or executed by the S/S		LS (§3.2.6.5)
SP-SR-SS7	It will not be possible to send a command via the LS port unless the response to the previous command has been read and processed. The only exception is for the commands sent by the Hard VM		LS (§3.2.6.5) Hard_VM (§3.2.5.2)
SP-SR-SS8	In case the Hard VM sends a command via the LS port, a copy of the data currently present on the receive registers will be saved in memory		Hard_VM (§3.2.5.2)
SP-SR-SS9	The OBS will be able to read the 3 FIFOs where the data sent by the S/S via the 3 High-Speed data links are stored	UR-FU11	HS (§3.2.6.7)
SP-SR-SS10	The OBS will be able to interpret the format with which science data are sent by the S/Ss as per AD9 and AD10	UR-GE8-10	HS (§3.2.6.7)
SP-SR-SS11	The OBS will read the frame ID and check that it a valid ID for that FIFO. In case of failure a TM (5,1) shall be generated with specific error codes and parameters to signal the error. The related FIFO channel is then considered out-of-sync.		HS (§3.2.6.7)
SP-SR-SS12	The OBS will read the frame length and check that it is a valid length for that frame ID. In case of failure a TM (5,1) shall be generated with specific error codes and parameters to signal the error. The related FIFO channel is then considered out-of-sync.		HS (§3.2.6.7)
SP-SR-SS13	The OBS will read a number of words (from the FIFO) consistent to the read frame length, compute the resulting XOR and compare the resulting XOR with the XOR checkword present at the end of the frame. In case of failure a TM (5,1) shall be generated with specific error codes and parameters to signal the error. The related FIFO channel is then considered out-of-sync.		HS (§3.2.6.7)
SP-SR-SS14	No attempt will be made to recover the data from a FIFO is in an out-of–sync state. Instead the FIFO will be reset.		HS (§3.2.6.7)
SP-SR-SS15	It will be possible to flush the FIFOs by TC		CMD_SEQ (§3.2.6.4) HS

			(§3.2.6.7)

2.8 Synchronization Requirements

ID	Requirement	Related UR	Design
SP-SR-SY1	The time synchronization activities will be performed at the earliest possible time after reception of the "sync without data word" Mode command from the CDMS	UR-SY1	ISR1553 (§3.2.5.1) TIME (§3.2.6.2)
SP-SR-SY2	The OBS will use the timing information available on the 1553 bus to synchronize the DPU timers to the spacecraft time	UR-SY1	ISR1553 (§3.2.5.1) TIME (§3.2.6.2)
SP-SR-SY3	The OBS will check that the timing information has been regularly updated before using it	UR-SY1	TIME (§3.2.6.2)
SP-SR-SY4	The OBS will compute the difference between the DPU time and the spacecraft time within 100 μsec from each start of frame	UR-SY1	TIME (§3.2.6.2)
SP-SR-SY5	TheD**t** parameter will be available to all OBS tasks that will need to get a time stamp	UR-SY4	TIME (§3.2.6.2)
SP-SR-SY6	The OBS will provide a time stamp for all needs in the code by reading the operating system time and adding the currently valid D**t**.	UR-TM3	§3.2.3
SP-SR-SY7	Whenever the time has not yet been synchronised (e.g., after switch on or reset), the OBS shall set to 1 the MSB of the time field in the header of TM packets.	UR-SY2	TBD

3 Architectural Design

3.1 The DPU/VIRTUOSO/OBS System

The DPU OBS will run under VIRTUOSO, an operating system designed for use in DSP environments, where speed of response to interrupts is usually critical. This environment allows the implementation of a multitasking application: a VIRTUOSO task in the OBS is an independent module consisting of one or more C routines, with its own thread of execution and set of system resources. It performs a well-defined function or set of functions and communicates information to other tasks. Tasks can be assigned priorities depending on their criticality: VIRTUOSO will assign CPU resources accordingly. Task intercommunication and synchronization is accomplished through a set of services like semaphores, events, FIFO messages, that are entirely managed by VIRTUOSO.

3.2 The SPIRE OBS

The SPIRE OBS implements a parametric concept where a relatively limited set of services coded in the software on-board can be invoked with different sets of parameters to provide all the functionalities required. The goal is to build a flexible tool that can be configured and used to execute all the required instrument functionalities by simply uploading tables of parameters, without the need to add/change software modules. This approach has the advantage to allow the development of the OBS application at an early time in the project, where the observing procedures are in a poor state of definition and specification; likewise, it makes the patching of the OBS a simple matter of uplinking tables of parameters rather than pieces of executable code.

3.2.1 The Virtual Machine

This concept is implemented in the SPIRE OBS via the Virtual Machines, a set of state machines able to interpret and execute at a precise timing a set of so-called OPCODEs (32-bit words) that provide basic functionalities like reading and writing into memory locations, register operations (shift, add), reading and writing to the S/S interface. The set of OPCODEs currently available can provide the typical functionalities of high-level programming languages like: variable definition, ‘if’ statements, conditional loops (e.g. ‘do while’) etc. The results in an “ad-hoc” programming language in which a complex observing procedure can be reduced to a series of OPCODEs and thus simply loaded as a series of 32-bits words.

The complete description of the VM implementation is found in RD6. Five different and independent VMs are implemented in the SPIRE OBS. The main VM is also called HARD_VM because the timing of the OPCODE execution is implemented via one of the DPU interrupt lines that is attached to a HW timer (§3.2.2.2.1). The other four VMs are called Soft_VMs because the timing is implemented using software timers handled by VIRTUOSO. The HARD_VMs, thanks to its excellent timing performances (10 μsec jitter) will be used to run observing procedures. The other four VMs will be used to run (also simultaneously to and observation running on the HARD_VM) batch procedures like PID temperature controls. Autonomy recovery functions can be un on any VM depending on its criticality.

3.2.2 Hardware/Software Interactions

3.2.2.1 Interfaces

The DPU interfaces with the Herschel spacecraft computer on one side, and with instrument subsystems on the other side. The spacecraft interface is implemented via a MIL-STB-1553B interface according to specifications contained in AD4; the packet-level protocol is handled by the interrupt-driven OBS task TMTC (§3.2.6.3). The subsystems interface is implemented via slow and fast serial links to the three SPIRE S/S, as described in RD3. The slow bi-directional links used to send commands and receive HK parameters from the from/to the S/S are handled by the OBS task LS (§3.2.6.5). The fast mono-directional links used to receive science data from the S/S are handled by the OBS task HS (§3.2.6.7).

3.2.2.2 Interrupts

There are three interrupt lines available on the SPIRE DPU. In ascending order of priority, they are dedicated to the DPU FIFOs (where the science data on the fast data links from the subsystems are received), the MIL-STD-1553B interface to the CDMS, and the DPU internal timer. The low-level interaction of the interrupt lines with the VIRTUOSO kernel is done through small standard assembler Interrupt Service Routines, called **ISRi_Handler** in the main OBS Architecture Diagram. The only function of these assembler ISRs is to transfer control to a C module by raising a VIRTUOSO Event; the target C module can either be directly associated to the interrupt via this event (using the VIRTUOSO call KS_SetEventHandler) or it can be put in a wait state on the VIRTUOSO Event. We briefly describe below the three interrupt lines available on the SPIRE DPU; the tasks and modules mentioned are described in detail in the rest of the document.

3.2.2.2.1 The TIMER Interrupt

This is the highest priority interrupt. The DPU timer is used by the Virtual Machine **Hard_VM** task to implement the SubSystem commanding at exact times with a less than 10 microseconds jitter. The DPU timer is basically a down-counter starting from a programmable number (in microseconds); when the down-counter reaches 0 it sends the Interrupt signal. This interrupt is served by the irq3.s routine, which transfers directly, not via an event, but via a direct call to the vm.c C routine, the control to the Hard_VM task.

3.2.2.2.2 The 1553 Interrupt

This is the second highest priority interrupt. This interrupt line is utilized by the MIL-STD-1553B Advanced Computing Engine (ACE) chip that interfaces the DPU to the CDMS. The ACE is software programmable to associate the interrupt line to any 1553B event (like reception of messages on particular SAs, reception of Mode Codes, etc.). This interrupt line is served by the irq2.s routine that raises the ISR_1553_Event; this event is associated to the ISR_1553 C module which is configured as a VIRTUOSO Event Handler, that is the real Interrupt Service Routine for this interrupt. Once the Event Handler has completed execution it can decide if the control has to pass to other tasks waiting on that same event.

3.2.2.2.3 The FIFO Interrupt

This is the lowest priority interrupt. This interrupt is dedicated to the FIFOs on which the science data coming on the fast data links from the SubSystems are received. This interrupt line can be programmed to any of the empty/half-full/full states of the three SPIRE DPU FIFOs (it is a single physical line that is multiplexed and managed by an FPGA). The adopted setting is to trigger the interrupt at Half-FIFO-Full. This interrupt is managed by the irq0.s routine that raises the IRQ0_Event that in turn triggers the HS task.

3.2.3 OBS Tasks

The OBS is divided into a set of VIRTUSOS tasks. The following table lists the task together with a short description of their functions and the associated priorities (the lower is the number, the higher is the priority:

Task Name	Function	Priority
INIT	It performs the OBS and 1553 interface initialization. It is the first task to start and dies upon completion.	4
TIME	Keeps up-to-date the relationship between the internal DPU clock and the S/C clock	4
TMTC	It manages the TC and TM packet exchange with the CDMS	5
VM_1	This is the first of the Virtual Machines managed via the VIRTUOSO Task_Sleep directive	5
VM_2	This is the second of the Virtual Machines managed via the VIRTUOSO Task_Sleep directive	5
VM_3	This is the third of the Virtual Machines managed via the VIRTUOSO Task_Sleep directive	5
VM_AFX	Additional Virtual Machine managed via the VIRTUOSO Task_Sleep directive, to be used for Autonomy recovery procedures.	5
HS	Task responsible for reading the DPU FIFOs, check consistency of science frames and pack them into standard TM packets	6
VM_SVC	This task generates events, reports and other TM packets upon command from VM code	7
LS	It manages the dispatch of commands to the subsystems and the reception of parameters to/from the subsystems.	7
CMD_SEQ	Checks the header of the received TC packets, issues appropriate TC verification reports and, upon positive verification, interprets and executes them.	8
HK_ASK_0	First task that collects DPU and instrument parameters and generates HK packets.	9
HK_ASK_1	Second task that collects DPU and instrument parameters and generates HK packets.	9
HK_ASK_2	Third task that collects DPU and instrument parameters and generates HK packets.	9
HK_ASK_3	Fourth task that collects DPU and instrument parameters and generates HK packets.	9
HK_MONITOR	It monitors the HK parameter and, in case of critical values, invokes	9

	the appropriate Autonomy Function	
AUTONOMY	Task that handles Event Packet generation and recovery procedures upon reception of anomaly messages received from HK_MONITOR	10
IDLE	Performs TBD memory checks	11

Table 3-1 OBS Task list

Control exchange between tasks is implemented using **Events**, **Semaphores** and VIRTUOSO **FIFO message Queues**. These VIRTUOSO System Objects are described in some detail below; here we also mention that they can be, and are, also used in the OBS to transfer data between tasks.

Whenever a parameter or a group of parameters computed by a task is to be made available to other tasks, without the need to transfer control at the same time, we will use global variables. This because parameters cannot be passed from one task to another just as one would do with routine calls.

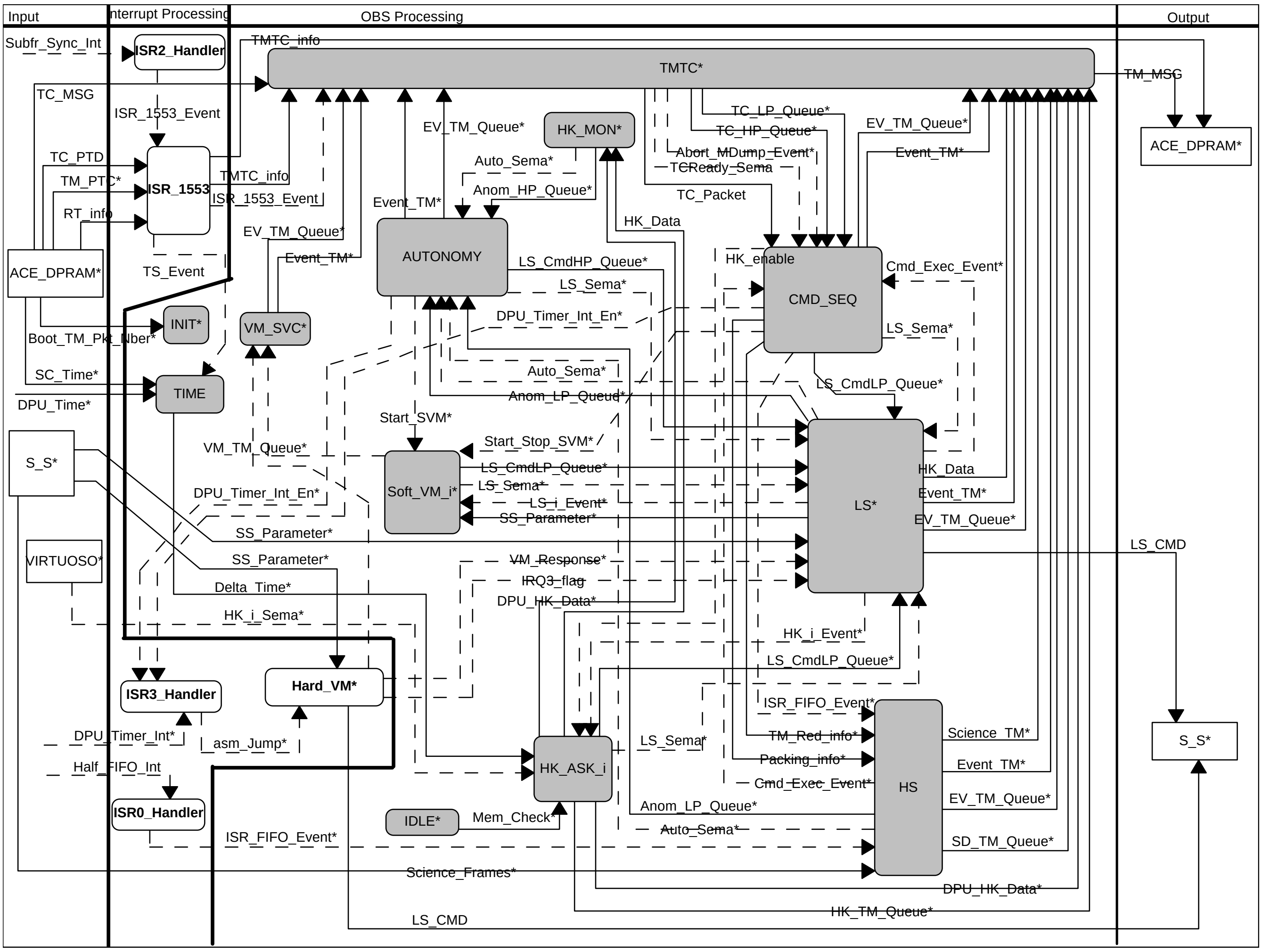


Figure 3-1 OBS Tasks Interconnection Diagram

3.2.3.1 Events

Events are the highest priority VIRTUOSO objects, after the Interrupts, to modify the schedule of task execution. Tasks can be set on a wait state until a particular event defined in the VIRTUOSO Project File is raised. At that point the tasks that are on wait, start to execute. The following events are used in the SPIRE OBS:

Event Name	Raised by:	Triggers task:
ISR_1553_Event	ISR2_Handler	ISR_1553, TMTC
ISR_FIFO_Event	ISR0_Handler	HS
TS_Event	ISR_1553	TIME
HK_i_Event	LS	HK_ASK_i
LS_i_Event	LS	Soft_VM (i+AFX)
Cmd_Exec_Event	LS, HS	CMD_SEQ

Table 3-2 List of VIRTUOSO Events used in the OBS

VIRTUOSO overhead to signal an event should be less than 15 μsec (RD9, §A.12).

3.2.3.2 Semaphores

While events only have two possible states, semaphores are counters. They are used when a condition for triggering a certain task can be set by multiple sources, or can be set many times before the waiting task starts execution; each time the waiting task serves the semaphore its counter is decreased by 1, until it gets down to 0. An example is the semaphore that signals that a new Telecommand has been received from the CDMS; if the OBS is busy executing some process, the TCs can be buffered and the related semaphore is signalled a correspondent number of times; the TC interpreter that is waiting on that semaphore will serve it until the semaphore counter is decreased to 0.

Another occurrence when the use of semaphores is to be preferred is in conjunction with cyclic operations. VIRTUOSO provides a number of system timers that can be configured to automatically signal semaphores. A typical example for semaphores usage is the periodic HK packet collection.

The semaphores used in the OBS are:

Semaphore Name	Function	Raised by	Triggers:
HK_i_Sema	Starts the periodic HK packet collection	VIRTUOSO timers	HK_ASK_i
LS_Sema	Signals LS that a command has to be sent to the SubSystems	CMD_SEQ, HK_ASK_i, Soft_VM_i,	LS
TCReady_Sema	Signals that a new TC has been downloaded from the CDMS and is ready to be verified and executed	TMTC	CMD_SEQ
Auto_Sema	Signals an anomaly or an out-of-limit conditions in the HK parameters	HK_MON, LS, HS, VM_SVC, HK_ASK_i	AUTONOMY

Table 3-3 List of VIRTUOSO Semaphores used in the OBS

VIRTUOSO overhead to signal a semaphore to another task that is on a wait state on that semaphore is of the order 50 μsec (RD9, §A.12)

3.2.3.3 FIFO Queues

VIRTUOSO FIFOs are system objects used to transfer control and data to other tasks. FIFOs (First-In-First-Out) are queues entirely managed by VIRTUOSO. Tasks can be put on a wait state on the reception of messages on FIFO queues. Contrary to events and semaphores, FIFO messages can bring along parameters (max 10). The FIFO queues can be specified in the VIRTUOSO Project File with the length of the associated message and the maximum number of messages that the queue can handle. The FIFO queues in the OBS are:

FIFO Queue	Function	Sent by:	Received by:	# MSG	# Words
TC_HP_Queue	Notifies that an immediate command is ready for execution	TMTC	CMD_SEQ	8	10
TC_LP_Queue	Notifies that a normal command is ready for execution	TMTC	CMD_SEQ	8	10
EV_TM_Queue	Notifies that a new event TM packet is ready on the EV_POOL	AUTONOMY, VM_SVC	TMTC	80	10
HK_TM_Queue	Notifies that a new HK TM packet is ready on the HK_POOL	HK_ASK_i	TMTC	32	10
SD_TM_Queue	Notifies that a new science TM packet is ready on the SD_POOL	HS	TMTC	128	10
LS_HP_Queue	Notifies that a high-priority command has to be sent to the SubSystem	Soft_VM_i	LS	64	5
LS_LP_Queue	Notifies that a low-priority command has to be sent to the SubSystem	CMD_SEQ, HK_ASK_i	LS	1024	5
VM_TM_Queue	Notifies that an event/report packet is to be sent	Hard_VM, Soft_VM_i	VM_SVC	64	3
Anom_LP_Queue	Notifies a low-priority anomaly		AUTONOMY	512	10
Anom_HP_Queue	Notifies a high-priority anomaly		AUTONOMY	512	10

Table 3-4 List of VIRTUOSO FIFO Queues used in the OBS

VIRTUOSO overhead involved in sending a FIFO message to a waiting task and reading it, is ~70 μsec (see RD9, §A.12)

3.2.4 Data Memory Management On-Board

The DPU memory is structured according to the DPU Memory Architecture File that will be delivered together with the OBS code. In particular the Data Memory consists of 512 kW (32-bits words) is divided into two blocks. The first one is SEG_DMDA and contains the static variables used by the OBS; the two most important sections of this segment are those hosting the Tables and the Memory Pools. These will be described below in detail. The second segment is SEG_CHEAP and is used by VIRTUOSO to handle semaphores, events, FIFOs.

3.2.4.1 On-Board Table Management

Implementing a parametric approach in the SPIRE OBS requires that all variables governing the code functionalities (e.g., packet structure definitions, VM codes, monitoring limits, etc.) are made available in tables that can be easily loadable/updatable/downloadable via standard routes using the services provided by AD4, without having to re-compile and re-load the entire image of the OBS code. The SPIRE OBS implements a table management system where tables can be dynamically allocated on-board and can be addressed simply using ID numbers that are internally resolved into absolute addresses by the OBS. The memory area used to store the on-board tables, called **tabellone,** resides in the SEG_DMDA block and its size is 128 kW.

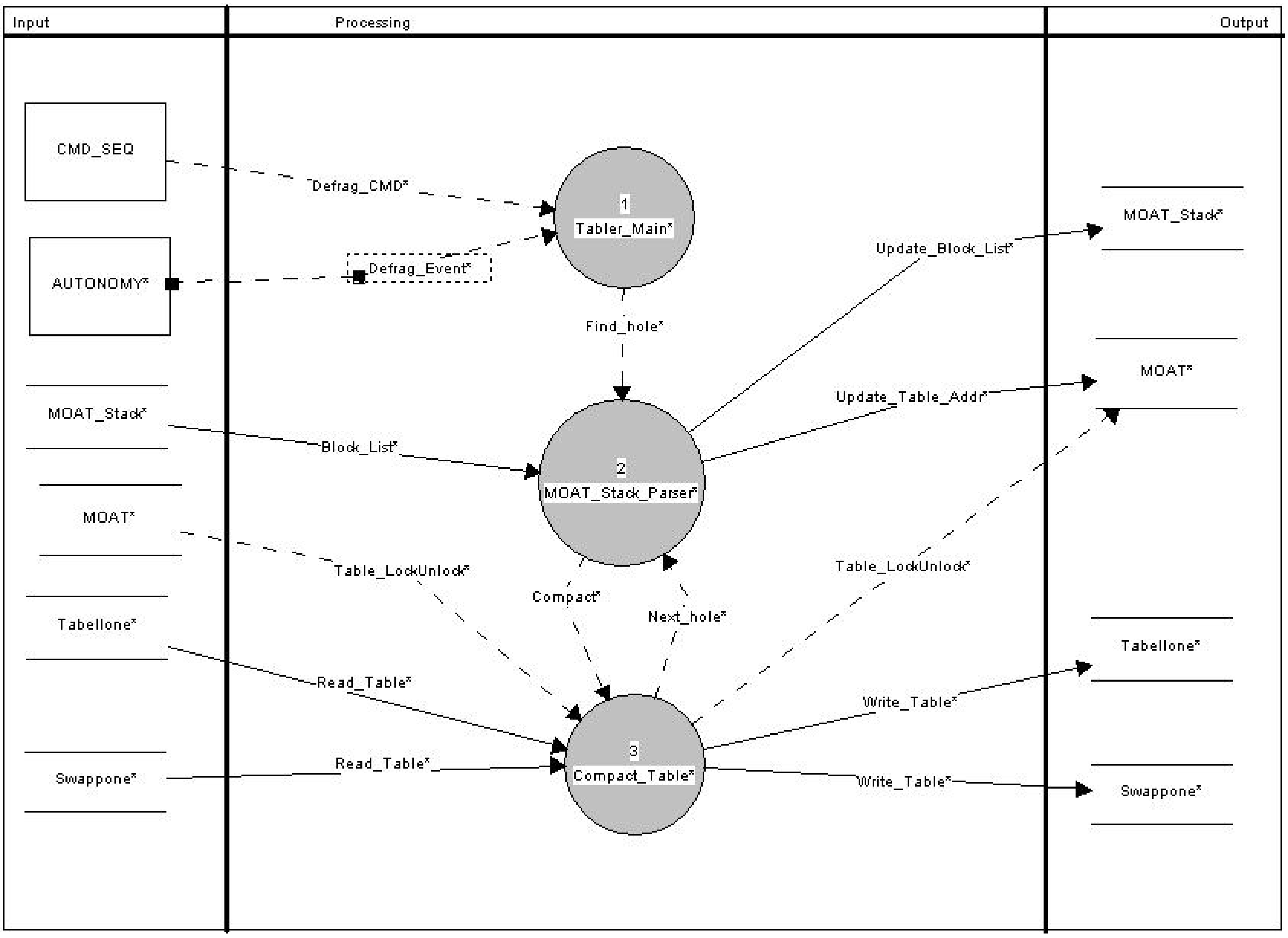
A table is characterised on-board by an ID number, a starting memory location, a length and a series of flags indicating their usage status. Critical tables (HK definition tables, VM code) can be locked while they are being used; this prevents access by other tasks that could modify the table contents while the table is being used by another task. As an example, a table containing an HK packet definition that is currently being used to collect HK parameters cannot be modified/deleted.
Similarly if a VM program is executing, its table ID will be locked as well as all the tables containing VM code called from within the master program.

The set of parameters that characterizes each table is stored and constantly kept up-to-date in a master table called the **MOAT** (Mother Of All Tables), which is also contained in tabellone**.** The exact position and size of all tables (but the MOAT) within tabellone is not fixed to allow full flexibility in the table management (create/modify/delete). When a new table with the required ID number is to be created, the OBS looks into the MOAT to identify the location of a free contiguous block of the required size within tabellone. The corresponding entry in the MOAT is updated accordingly. Thanks to the MOAT, the tables in tabellone do not need to be created in order of Table ID; i.e., the start address of table 46 may be higher than the start address of table 117. This quite flexible table management scheme will lead in time to a certain degree of fragmentation in tabellone (holes are left when tables are deleted), that can be removed via a dedicated TC.

The reallocation of tables to optimize memory on-board is only performed on tables that are not currently locked and is accomplished through the following mechanism. The reception of a dedicated TC triggers the Tabler_main function that calls the MOAT_Stack_Parser function; the latter starts parsing the MOAT_Stack records in the MOAT_Stack table (that contains the list, sorted by address, of occupied blocks in tabellone) until it finds a hole, i.e. when a table starts does not start immediately after the end of the previous table. The MOAT itself cannot be used for this purpose because the MOAT is sorted by Table ID and not by address.

When MOAT_Parser finds a hole, it calls the Compact_Table function. This function first checks the MOAT to see if the table is locked by another task (it could be in use for HK collection or VM code execution); if the table is free, it locks it by raising the lock flag for that Table ID in the MOAT; this prevents tasks that use that table to access it while it is being moved. It copies the table from tabellone to swappone (which is an 8 kW reserved area used for swap in SEG_DMDA), and deletes it from tabellone. Then it reads it back from swappone and writes into tabellone immediately following the end address of the last table in the contiguous area of tabellone (i.e., where before there was the hole).

Finally, it updates the start address for that Table ID in the MOAT and the block list in MOAT_Stack, and unlocks the table. The control is passed back to function MOAT_Parser that finds the next hole.



3.2.4.2 Memory Pools

Memory pools are DM areas where fixed-size 512W blocks of memory can be dynamically allocated/deallocated to host TCs received or TM packets that are being built for dispatch to the satellite. These areas are managed as circular buffers where read/write pointers are held and updated in static structures that also keep track of usage status of each block (e.g. which task reserved a particular block) and of the whole pool (number of used blocks); when the block usage is greater than 80% an event TM packet is generated. Each pool is specified with the maximum number of blocks that can be allocated, while the size of each block is fixed. The Memory Pools defined in the SPIRE OBS are defined as follows:

Pool Name	Usage	# of Blocks	Priority
TC_POOL	Telecommand Packets	16	
EV_POOL	Event Telemetry Packets	64	

HK_POOL	HouseKeeping TM Packets	32	
SD_POOL	Science Data TM Packets	128	

Table 3-5 List of Memory Pools used in the OBS

3.2.5 C Interrupt Service Routines

3.2.5.1 ISR_1553

This is not a VIRTUOSO task, but it is the Interrupt Service Routine for the IRQ2 interrupt used by the MIL-STD-1553B interface. Formally, ISR_1553 is a VIRTUOSO Event Handler. The routine is immediately triggered on the event **ISR_1553_Event**, raised by the assembler routine isr2.s. ISR_1553_main first updates the instrument status by writing in SA1T of the ACE DPRAM the required information, and then parses the Mode Code to understand the type of interrupt. It then passes control to another function Transfer_Handler. If the Mode Code is a **synchronize without data word** command, it: i) resets to 0 the internal DPU SubFrame Counter, ii) raises the TS_Event to wake-up the TIME_task. If it is a **synchronize with data word command** it: i) increments the internal SubFrame counter, ii) decode the data word to understand the address of the RT allowed for TM transfer in the current SubFrame.

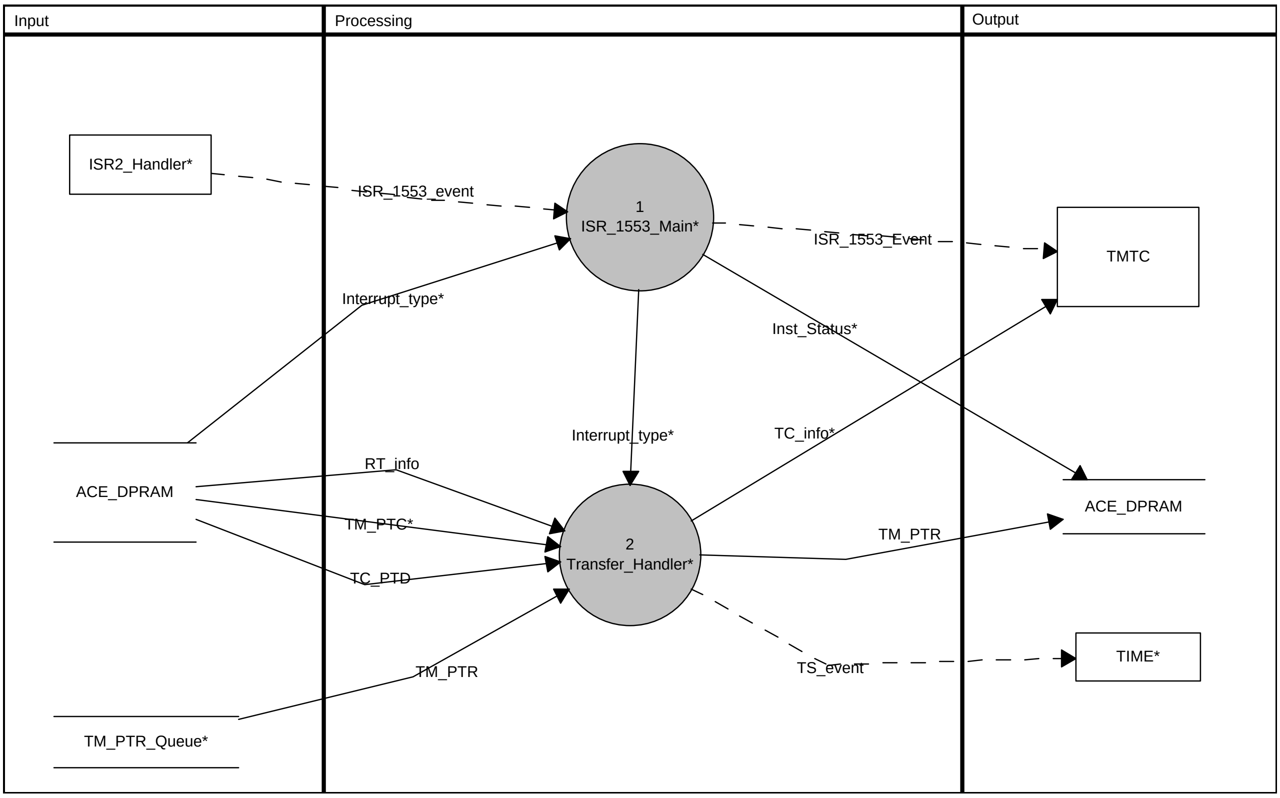


Figure 3-2 ISR_1553 Module Functional Decomposition

After these interrupt-dependent actions, the Transfer_Handler function checks if a new TC packet is available from the CDMS, by reading the TC Packet Transfer Descriptor (TC_PTD) from SA27R on the ACE DPRAM and transfers this information to the TMTC task. It then checks if the previously sent TM packet has been successfully received by the CDMS by checking the TM Packet Transfer Confirmation (TM_PTC) from SA10R in the ACE DPRAM. Finally

Transfer_Handler checks if there are new TM packets waiting to be sent to the CDMS by checking the status of the TM_PTR_Queue (that holds the list of TM Packet Transfer Requests for pending TM packets) and, if the check is positive, transfers the PTR for the next available TM packet on SA10T of the ACE DPRAM.

Once the ISR_1553 Event Handler has completed execution, its returned value tells VIRTUOSO if the control should be passed or not to the other tasks (TMTC) waiting on ISR_1553_EVENT. A TRUE returned value is used if the previous TM packet was downloaded and confirmed by the CDMS. In this case the ISR_1553_EVENT is passed on to TMTC to load a new TM packet in to the 1553 DPRAM. (see later).

3.2.5.2 Hard_VM

This is not properly a VIRTUOSO task, but rather an Interrupt Service Routine triggered by the isr3.s assembler ISR which in turn is activated by the TIMER interrupt.

This task allows for the execution of operations (including commands to the Sub-Systems) at a fixed time with a maximum jitter of 10 microseconds. The task, interrupt driven, is started/terminated by a DPU internal command which enables/disables the DSP highest priority interrupt (IRQ3) driven by a 1 MHz clocked HW timer. For each IRQ3 request, the task reads from a preloaded table (the VM code) the commands to be executed/ transmitted. A VM code is actually a one column 32 bit word vector containing commands to be sent to the Sub-Systems, timer setting (IRQ3), mutex (i.e. Sub-system interface locking), loop and other Virtual Machine "assembler" instruction, operating as an absolute program. See RD6 for a complete description.

A number of baseline VM programs, with functionality for the foreseen observation modes, will be resident on the DPU. These programs, stored in tabellone, will be modified/reloaded via TC, thus easing the need for OBS patching. A program can be as simple as a loop calling a preloaded subroutine.

In order to avoid collision on the low speed I/F with the LS task, a special (internal) command is foreseen to lock/unlock (setting the IRQ3_flag) the low speed I/F. The locking command will precede the SS commands of at least 2ms in order to allow for the possible contemporary (just started) transmission of a command via the LS task. As a safety measure, the Hard_VM stores in a back-up memory location the contents of the low-speed “receive” register in order to preserve the integrity of the parameters requested by LS task; this is notified to the LS task using the VM_Response task (see §3.2.6.5).

The VM task aborts itself when the END (end of program) opcode in the VM code is reached. VM is a state machine running into the whole system in a quite autonomous way.

A VM compiler will be provided (see RD6) to resolve all the mnemonic labels and constant in a VM program and produce the absolute VM code. A VM simulator will also be provided (see RD6): it will be a modified version of the OBS VM section, to control any "unprotected" CMD/RCMD instruction and output (on the out list file) a timeline of the SS commands.

Event/Report TM packets can be generated during the execution of VM code by using specific opcodes which cause the dispatch of FIFO messages (containing all relevant info) to the VM_TM_QUEUE.

A much more detailed description of the Virtual Machine is given in RD6.

3.2.6 Tasks Description

3.2.6.1 INIT Task

The INIT task has the highest priority and runs as soon as the PROM switch-on procedure is completed and the control is passed to the OBS application. This task makes all the initializations that aren't made automatically by the OS using the application configuration files. A most important part of the INIT sequence is the configuration of the 1553 interface ACE.

The 1553 SubAddresses (SAs) will be configured according to specifications in AD4. The SAs dedicated to reception of TM packets will be configured as circular buffers in order to be able to enqueue TM packets with the necessary speed in case faster-than-nominal telemetry transfer rates are needed. The ACE will be configured to issue an interrupt request upon reception of sync mode codes.

The OBS will be started after completion of the PROM-resident Boot Software; since this software generates Event TM packets to the CDMS, the OBS will have to check how many packets have already been sent in order to avoid sending TM packets with the same sequence number in the “TM Packet Transfer Request” (see AD4). This will be done by the INIT task by checking the 1553 DPRAM area corresponding to SubAddress 10 in reception (SA10R), before reconfiguring the 1553 Interface memory.

3.2.6.2 TIME Task

This task is activated each second after reception of the TS_EVENT from ISR_1553. It is responsible for the time synchronization between the DPU and the Spacecraft. It i) checks that the Spacecraft time fields (SA8R) have been updated by the CDMS and reads them, ii) reads the VIRTUOSO time and it computes the difference D**t**. Each time the OBS is required to provide the current DPU time (e.g., to put the time stamp on TM packets), the VIRTUOSO time will be read and the D**t** computed by TIME_task will be added. D**t** will be also made available to HK_ASK_i task to include it as a DPU HK parameter.

3.2.6.3 TMTC task

This task, together with the ISR_1553 interrupt service routine (see §3.2.5.1), handles the interface with the spacecraft CDMS. It is enabled by the ISR_1553_EVENT raised by ISR_1553.

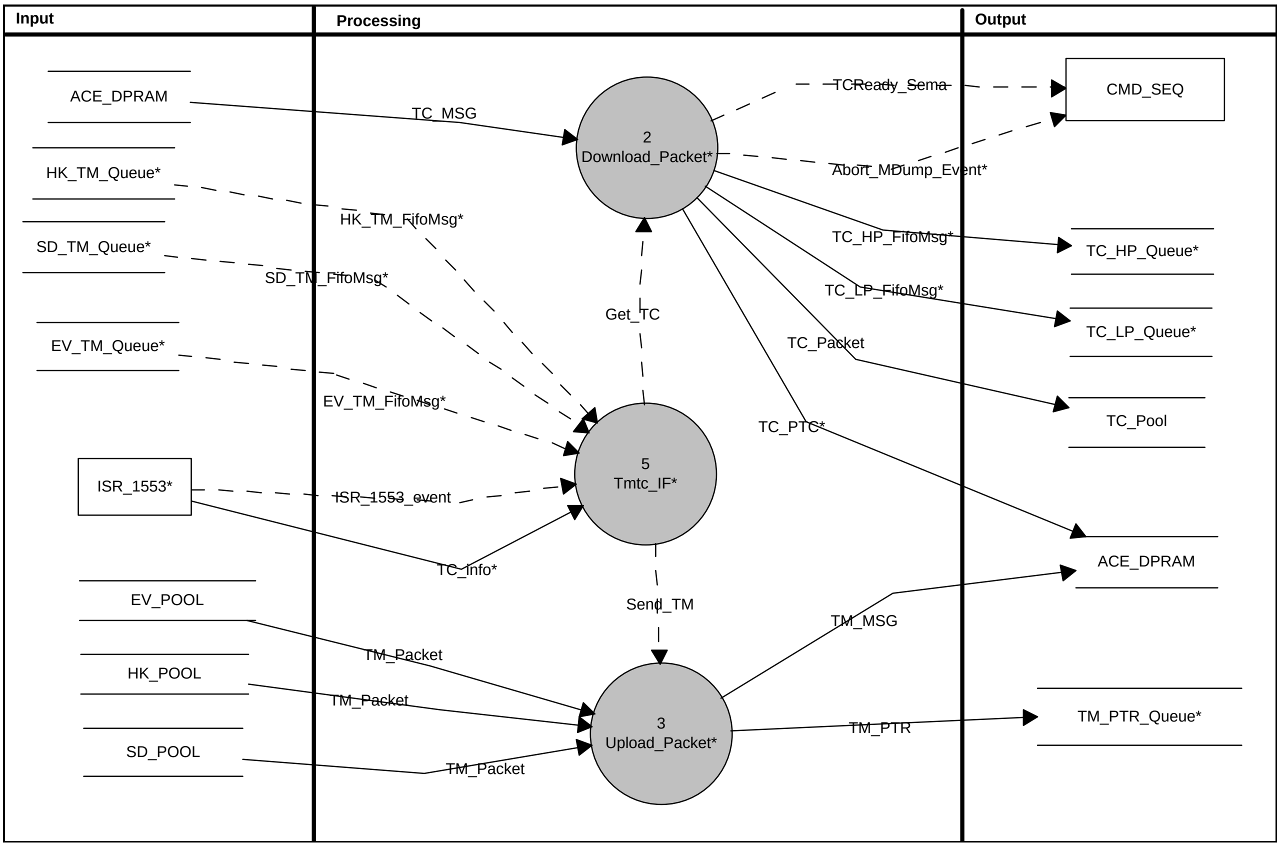


Figure 3-3 TMTC Task Functional Decomposition

If there are TM packets ready in the various memory pools (written there by a variety of other tasks and signalled to TMTC using the FIFO Queues EV_TM_QUEUE, HK_TM_QUEUE and SD_TM_QUEUE), and if there is space available on the transmission buffers SA11T-SA26T of the ACE DPRAM, the function Tmtc_IF transfers control to the function Upload_Packet. This function, using the information passed along with the FIFO Queue messages (see §3.2.3.3), copies the TM packet from the relevant memory pool into the proper SAs on the ACE DPRAM, compiles the appropriate TM PTR and writes it in the TM_PTR_Queue (where it will be read by ISR_1553, see §3.2.5.1).

If Tmtc_IF is notified by ISR_1553 (with the TC_info data flow) that there is a new TC packet sent by the CDMS, it calls the Download_Packet function. It reads the relevant SAs from the ACE DPRAM, builds the TC packet directly in the TC_POOL memory pool and writes the pointer to the TC into the FIFO queues TC_HP_queue or TC_LP_queue depending on TC priority. The high-priority TC are the so-called immediate commands that have to be executed as soon as they are received; they are the “Abort VM” and “Abort Memory Dump” commands. All others are low (standard) priority commands.

Finally it raises the TC_READY semaphore to CMD_SEQ, and acknowledges TC reception to the CDMS by copying the TC Packet Transfer Descriptor into the TC Packet Transfer Confirmation on SA27T.

3.2.6.4 CMD_SEQ Task

This is the main task of the OBS. It is in charge to check, interpret and execute all the received TCs. CMD_SEQ is in a wait state until the "TC_Ready" semaphore is signaled from task TMTC, notifying the availability of a new TC. When this happens, CMD_SEQ reads in succession from the FIFO queues TC_HP_QUEUE and TC_LP_QUEUE the message containing the pointer to the TC in the TC_Pool. These actions are done in the cmd_seq_main function. All functions in this task (see below) will act based on the contents of the TC; the only parameter passed among the various functions is the pointer to the TC in the TC_POOL, and not the TC packet itself. This avoids multiple copies of the TC packet flowing around between functions, optimizing memory usage and maximizing speed of execution. We will maintain on board a list of indexes to relevant TC fields for every TC packet type and subtype; in this way there will always be only one copy of a TC packet for use by all functions.

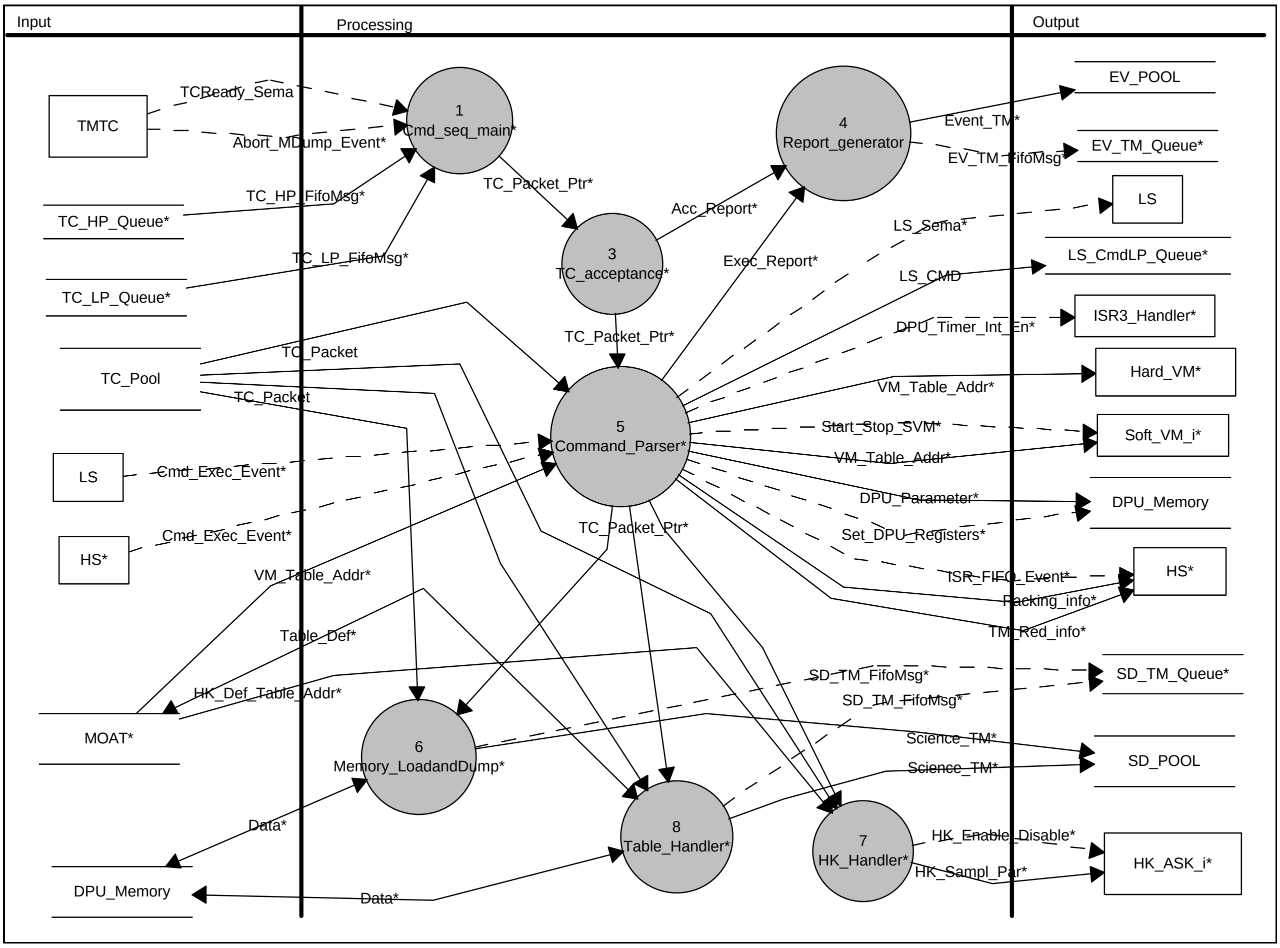


Figure 3-4 CMD_SEQ Task Functional Decomposition

The TC packet pointer is then passed to function tc_acceptance that performs the complete sequence of TCs verification steps, down to their "executability" (i.e. the validity of the Application data in the TCs). The acceptance information (TC accepted or refused) is then passed to the report_generator function. This function is not properly a separated task, but rather a group of routines compiling the appropriate report into standard TM packets, writing them into the EV_POOL and signalling TMTC, via a message to the EV_TM_QUEUE FIFO queue, that a new packet is ready to be transmitted to the CDMS.

The function command_parser parses the TC packet type/subtype combination and takes appropriate actions. In case of TC (8,4) it also parses the Function_ID/Activity_ID combination.

Commands can be divided into two groups: atomic and complex. Atomic commands can consist either of simple setting of a parameter stored in the DPU memory (like the OBSID), or in resetting some DPU registers (like FIFO_Reset), sending a single command to the S/S (like the Reset_DRCU_Counters) or starting/stopping the VMs. These atomic commands are executed in the body of the command_parser function; the generation of the related execution reports (if required) is also initiated in this function.

Complex commands are those that involve a series of actions; this is the case of HK collection management (service 3), memory management (service 6) and many of the functions activity (service 8).

The HK_Handler function manages the activation/deactivation of the four independent housekeeping collection tasks HK_ASK_i. The relevant parameters (HK Packet definition tables, sampling, etc.) are modified in this function only, and made available to HK_ASK_i as global structures. The activation/deactivation is performed by starting/stopping the VIRTUOSO timers that triggers the HK_i_Sema semaphores (see §3.2.3.2) on which the HK_ASK_i tasks are on a wait state.

The Table_Handler function manages the creation/modification/deletion of tables in tabellone (see §3.2.4.1). This function uses the parameters passed from the ground via the TC to update the data for the relevant table ID and modify accordingly the MOAT entries for that table ID. In case of Table dump, the TM packets are created in this function and written into the SD_POOL and a corresponding FIFO message is written to the SD_TM_QUEUE to signal TMTC that a new packet is ready to be sent to the CDMS.

The Memory_LoadandDump function manages the loading/dumping of DPU memory using absolute memory addresses. In this case the TC packet contains all needed info to load/dump memory without having to resolve addresses via the MOAT. In case of memory dump or memory report, the relevant TM packets are created in this function and written into the SD_POOL and a corresponding FIFO message is written to the SD_TM_QUEUE to signal TMTC that a new packet is ready to be sent to the CDMS.

In all cases (e.g., configuring HK housekeeping, running VMs, etc.) where it is necessary to identify the relevant on-board table stored in tabellone, its address is always resolved from the MOAT.

3.2.6.5 LS task

The LS Task is in charge of transmitting commands to the subsystems, although it can be used to also retrieve certain DPU housekeeping parameters. The only exception is the Hard_VM task that can send commands directly to the SubSystems by writing directly to the Low-Speed interface. The task is triggered by the LS_Sema semaphore (see §3.2.3.2); function LS_main checks the LS_HP_Queue and LS_LP_Queue FIFO queues in this order and reads the FIFO message which contains three parameters: the actual command to be sent to the subsystem, the address in the DPU memory where to store the parameter returned in reply by the Sub-Systems, and an event number that LS has to raise upon completion.

There are two types of commands that can be sent to LS: DPU commands and Sub-System commands. DPU commands are a specific set of commands defined in RD7 that mimic the syntax of the Sub-Systems commands. The HK packet defined in AD9 contains both DPU and Sub-System parameters; since the HK packet definition table is organised as a series of 32-bit words containing the command needed to get that particular HK parameter, we find convenient to retrieve the needed

DPU parameters by means of Sub-Systems-like command syntax in order to have an homogeneous HK packet definition table. Each DPU Command ID is associated with a unique DPU parameter memory address.

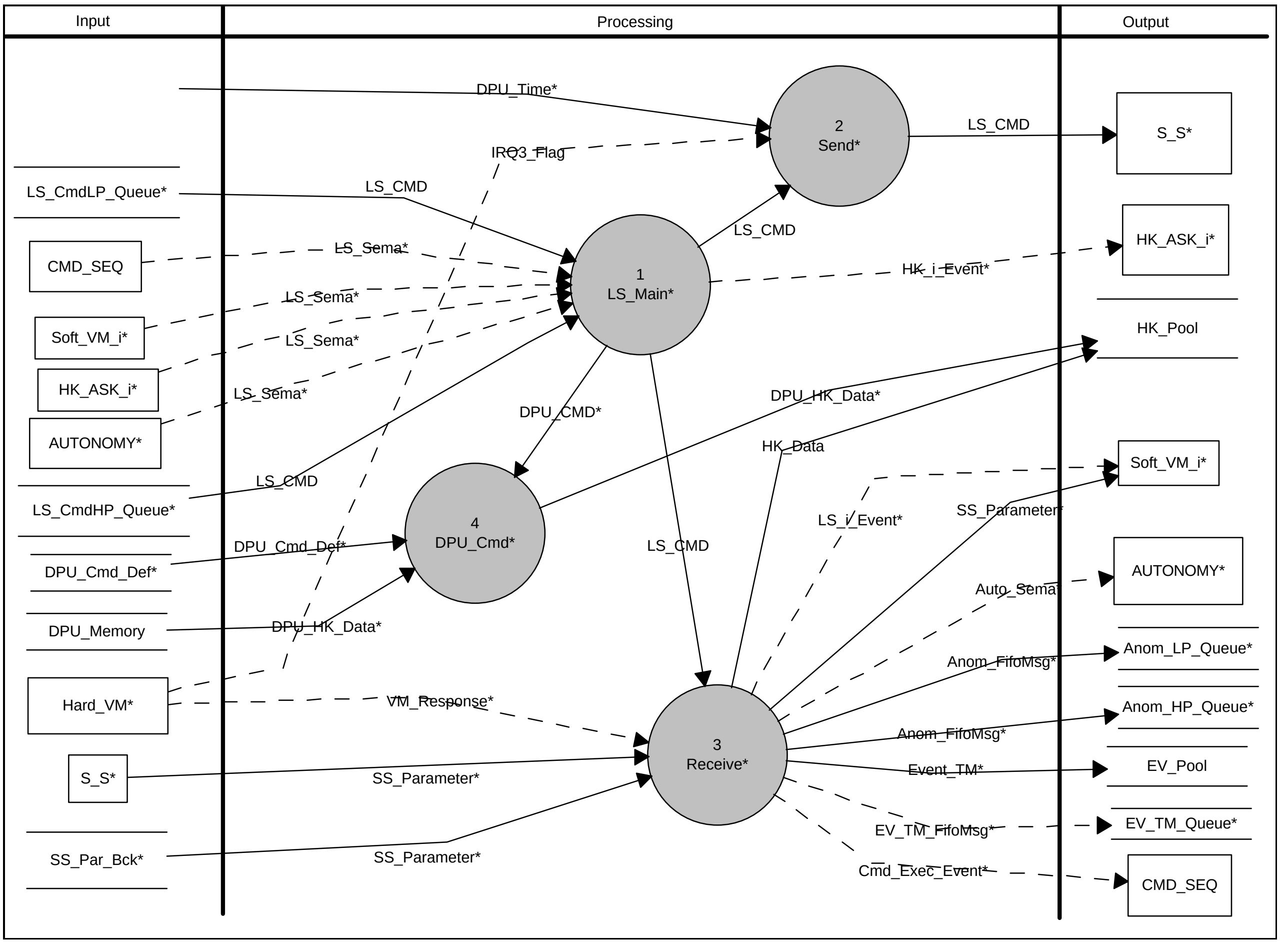


Figure 3-5 LS Task Functional Decomposition

If the MSb of the command word is 0, then it a DPU command. The function DPU_Cmd parses the command ID and put the corresponding parameter into the return address specified in the relevant FIFO message (which in most cases will be within an HK packet).

If the MSb of the command word is 1, then it is a Sub-Systems command. The Send function checks for the availability (IRQ3_flag set) of the low speed I/F (it might be in use by VM Task) and if not available suspends itself for 2 msec until the port is no longer busy. The function then writes the command word on the DPU register that maps the write port of the Sub-System interface and then the LS task is put to sleep for 2 milliseconds. The reason for this particular wait time is the following. In principle the Sub-Systems should respond within few hundreds of microseconds; in reality the LS task could be interrupted by interrupts, events, semaphores and FIFO messages that trigger tasks with priority higher than LS, so the wait time needs to be longer. Another aspect to be taken into account is that when a task goes to sleep VIRTUOSO transfer control to other tasks; this task switch has an overhead of about 100 microseconds so doing fast task switches is not very efficient in terms of CPU usage. A wait time of 2 milliseconds is an acceptable compromise between speed of response and CPU usage efficiency.

After the above mentioned wait time VIRTUOSO gives control back to LS. The Receive function first checks if the Low-Speed port is being accessed by the Hard_VM task. As explained in §3.2.5.2, the Hard_VM task gets control when the highest priority IRQ3 interrupt is triggered; this task is the only one to send Sub-Systems commands directly via the Low-Speed port without passing via the LS task. In particular, it may take control after LS has sent a command, but before LS has read the Sub-System response. To preserve the integrity of the Sub-System response to LS, the Hard_VM task reads the DPU memory locations where the Sub-System interface “receive” register is mapped, stores its contents in a back-up memory location and raises the VM_Response flag. The Receive function, based on the value of the VM_Response flag, will read the Sub-System replied parameter from the “receive” register of the Low-Speed port, or from the back-up location where the Hard_VM task stored it.

The Sub-Systems reply word to a command sent by the DPU contains ancillary information to diagnose possible interface or command format errors. If the command was correctly interpreted and executed by the SubSystems, they will echo the exact copy of the command ID (see AD10). In addition, the response word will also contain a 2-bits "Ack" field in place of the Sub-System address bits, indicating the result of the command (OK, Interface Time-out, Command Forbidden or Command unknown). If the "Ack" field will return OK then LS will assume the returned parameter is a valid one; if the “Ack” fields report an error condition or the echo of the command ID is not equal to the command ID sent, Event TM packets messages will be generated and sent to the satellite (containing error codes that specifically identify the anomaly condition). Anomaly Reports are also sent to the AUTONOMY task via the proper FIFO queues and the Auto_Sema semaphore will be raised to trigger the appropriate recovery procedures.

Receive will put the read parameter in the memory location specified in the FIFO message (see above) originally read by LS_main.

LS_main concludes its actions raising the event number specified in the FIFO message originally read by LS_main; presently the only foreseen event is the one signalling HK_ASK_i that the HK packet collection sequence is finished.

3.2.6.6 HK_ASK_i task

The OBS provides the ability to collect four independent HK packets at different sampling rates. In all figures the reference is always made to the i^{th} of these tasks. The tasks are enabled/disabled with KS_TaskSuspend/Restart VIRTUOSO kernel calls (the HK_Enable control flow).

The periodic activation of this task is via the HK_i_Sema semaphore that is raised by the associated VIRTUOSO timer (one per HK_ASK_i task) in the CMD_SEQ task. The HK_i_main function first resets the relevant VIRTUOSO timer to the sampling interval currently valid for that HK_ASK_i task; this parameter, together with the other ones characterizing the HK sampling (see AD9) are update and made available by CMD_SEQ task. Then HK_i_Main allocates a block in HK_POOL and passes its address to the Cmd_Enqueue function, which starts parsing the relevant HK Packet definition table (whose absolute address is resolved via the MOAT). In case a memory block could not be allocated an anomaly report is enqueued on the Auton_LP_Queue FIFO and Auto_Sema is raised to trigger the AUTONOMY task.

For each command word read from this table, Cmd_Enqueue sends a message on the LS_LP_Queue FIFO and raises the LS_Sema semaphore to LS task. The FIFO message to LS task contains the command word, the address where to store the parameter returned by the Sub-System or the DPU, and an event to be raised by LS (see §3.2.6.5); this event is always 0 (i.e., no event) except in case of the last HK collection FIFO message, for which the event ID is

HK_i_Event. As Cmd_Enqueue sends FIFO messages to LS, LS puts its replied parameter into the proper location of the HK packet in HK_POOL.

When LS has finished processing the last Sub-System parameter request it will raise the HK_i_Event, triggering the HK_Pkt_Build function. This function writes the header of the TM HK packet in HK_POOL and enqueues a message in HK_TM_Queue containing the address of the packet in HK_POOL. At that point a copy of the full HK packet is made on the DPU memory; this will be used by the HK_MON task to monitor the HK parameters.

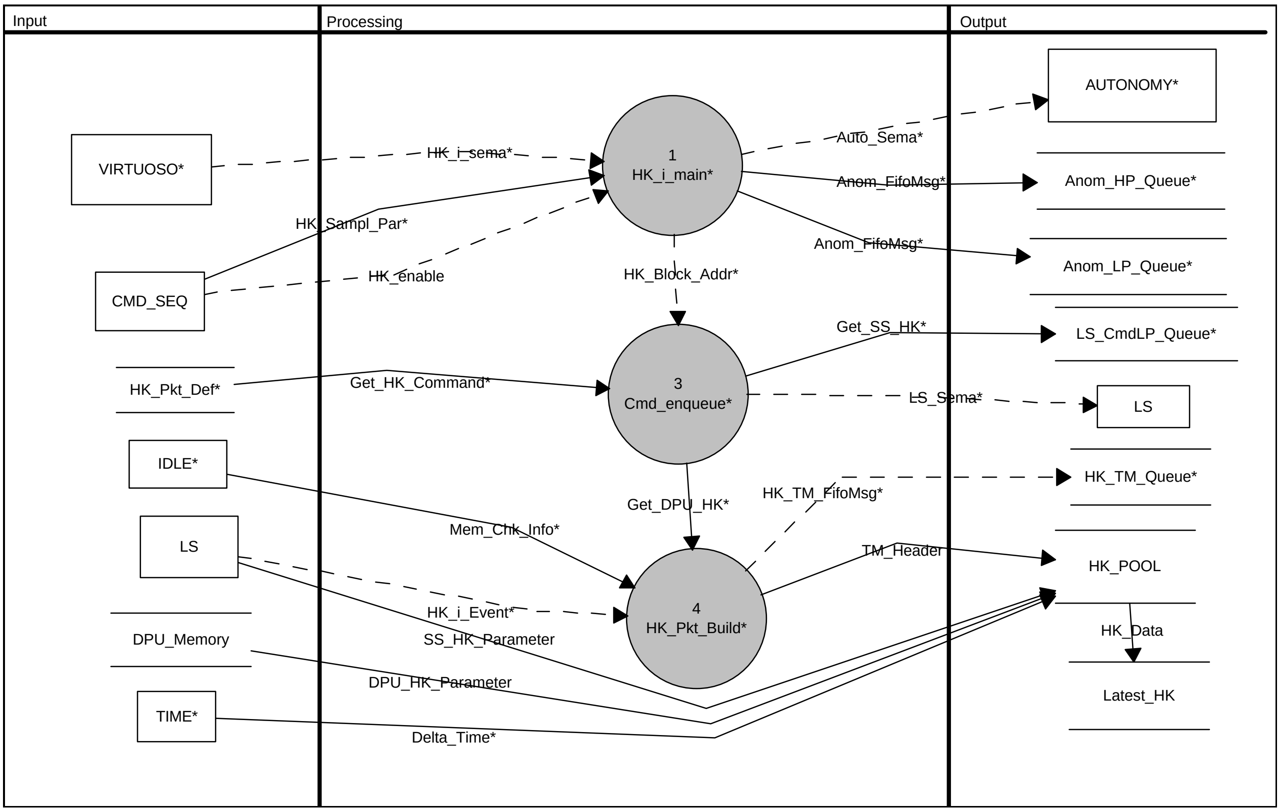


Figure 3-6 HK_ASK_i Task Functional Decomposition

3.2.6.7 HS task

This task collects science data, organized in self-consistent frames, from the Sub-Systems via the high speed I/F. The data on the high speed I/F are temporary stored on three 8Kwords (4Kwords in the AVM) deep HW FIFOs: the "half FIFO full" signal of each FIFO generates a HW interrupt (IRQ0). This interrupt is served by the ISR0_Handler, that in turn raises the ISR_FIFO_Event that activates the HS task operations. Due to the asynchronous operation of the FIFOs, the actual timing of the incoming data is lost and no cause/effect between commands (on low speed I/F) and received data (on high speed I/F) is possible, at least in a simple efficient and reliable way.

There are several types of science packets foreseen for the SPIRE instrument; each of them is made up of raw frames coming from the Sub-Systems (see AD9). The HS_main function allocates a memory block for each possible Frame_ID and transfers the block address info to the function Frame_Interpreter.

This function parses the interrupt registers in order to understand which FIFOs triggered the half_full interrupt and starts reading the science frames from the relevant FIFO. The first word of the frame is the frame_ID and the second is the frame length; the frame ID is converted into a SID so that the Frame_Interpreter is able to channel each frame to the proper TM packet in SD_POOL. The frame length allows to read the exact number of words for that frame; Frame_Interpreter perform an XOR of the frame words and compares it to the checksum word provided by the Sub-Systems at the end of that same frame. In case the frame is not self-consistent (wrong frame_ID, incorrect checksum, etc.) Event TM packets will be generated. An anomaly message will be enqueued on the Auton_LP_Queue FIFO and Auto_Sema will be raised to signal the AUTONOMY task to take appropriate measures. Once the frames have been read and checked they are written into the relevant TM packet in SD_POOL. When the TM packet is ready, Frame_Interpreter sends a FIFO message in the SD_TM_Queue FIFO to TMTC, with the pointer to the newly written TM packet.

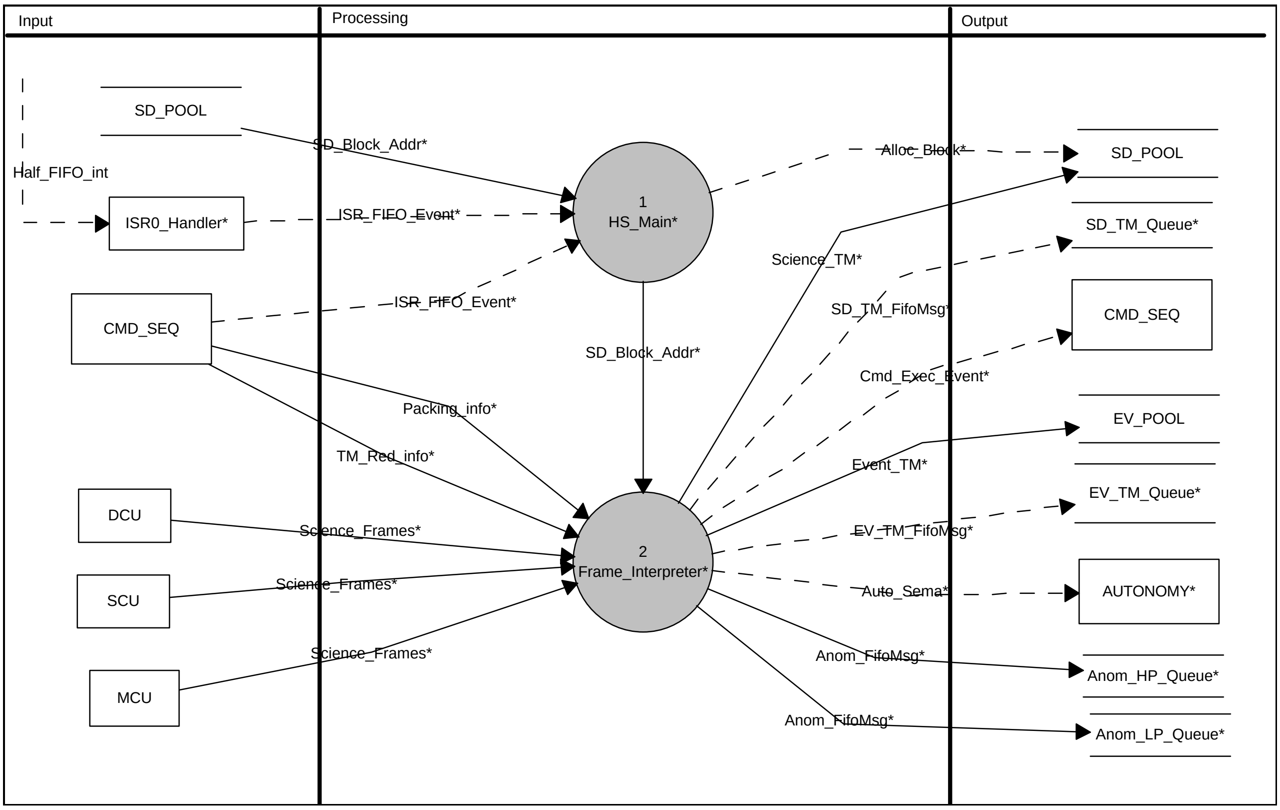


Figure 3-7 HS Task Functional Decomposition

3.2.6.8 HK_MON task (N/A in OBS Version 1)

This task implements a parameter-status conditional monitoring system. A predefined list of HK parameters, modifiable via TCs, is monitored depending on the particular values of other HK parameters. The check is done against soft and hard limits tables stored on-board. The monitoring rate will not exceed the HK collection rate. In case of out-of-limits, an anomaly message shall be enqueued on the Auton_HP_Queue FIFO and the Auto_Sema will be raised to signal the AUTONOMY task.

3.2.6.9 AUTONOMY task (N/A in OBS Version 1)

This task is triggered by the Auto_Sema semaphore, which can be raised from various locations in the OBS. The task will then read from the Auton_HP_Queue and Auton_LP_Queue (in this order) the anomaly message and will take appropriate actions.

The first action will be to generate an Event_TM packet, by writing it into the EV_POOL and notifying it to TMTC task via the EV_TM_Queue. The generation of event TM packets will be done only at the transition between nominal and anomaly conditions; no event packets will be generated as long as the anomaly condition persists. Another event will be generated when the conditions go back to nominal.

The second action will be to start a recovery procedure that will clearly be anomaly-dependent. These procedures will be implemented as compiled pieces of code (in which case the task will be able to, e.g., send commands to the Sub-systems via the LS task, and/or as VM codes to be run on any of the Virtual Machines.

3.2.6.10 Soft_VM_i task

In addition to the Hard_VM Virtual Machine, the OBS provides three mode VMs that, unlike the Hard_VM Virtual Machine, are driven by VIRTUOSO timers. The only other distinction with respect to Hard_VM is the management of command dispatch to the Sub-Systems; the Soft_VM_i tasks send their commands via the LS_HP_Queue, which is the high-priority FIFO queue to LS. These VMs will be used to implement the PID controls.

3.2.6.11 VM_SVC task

The task is on wait on the FIFO queue VM_TM_Queue (written by both Hard_VM and Soft_VM_i tasks); when a message is received on that queue the task reads the info provided and generates the proper execution reports or event requested.

3.2.6.12 IDLE task (N/A in OBS Version 1)

This task is the lowest priority in the whole OBS. It is executed when nothing else is running. It performs TBD checks on the DPU memory (like computing a checksum on portions of DPU memory) and storing results in HK parameters made available to HK_ASK_i.

4 User Requirements Traceability Matrix

This table or requirements is taken directly from AD7. Next to each requirement we state how the present OBS architecture design meets them.

4.1 Switch-on Requirements

Req. ID	Verification	Notes
OBS-UR-ON1	The Switch-on procedure is implemented in the Boot Software, which is not part of the OBS application. Requirements are verified in RD8	
OBS-UR-ON2		
OBS-UR-ON3		
OBS-UR-ON4		
OBS-UR-ON5		

4.2 Telecommands Requirements

Req. ID	Verification	Notes
OBS-UR-TC1	The Command_Parser routine in the CMD_SEQ task (§3.2.6.4) will decode the [Type, Subtype, Function_ID, Activity_ID] combination using a series of nested “switch” statements.	
OBS-UR-TC2	Deleted	
OBS-UR-TC3	The Transfer Layer Protocol specified in AD4, used by the CDMS to send TC packets, is implemented in the OBS by the combination of the ISR_1553 Interrupt Service Routine (§3.2.5.1) and the TMTC task (§3.2.6.3).	
OBS-UR-TC4	TC reception and unpacking is immediate because ISR_1553 (§3.2.5.1) is triggered by an event (§3.2.3.1) raised by an Interrupt Service Routine, and the task TMTC (§3.2.6.3) has the highest priority (see table in §3.2.3) after the INIT task (§3.2.6.1), which runs only at start-up, and TIME task (§3.2.6.2) that runs only once per second. The read/write operations needed to implement complete reception and unpacking of a maximum-size TC packet should not take more than 0.3 msec to execute. Overall VIRTUOSO overhead to pass control from TMTC to CMD_SEQ (assuming no other task is interrupting) is of the order of 0.2 msec (including semaphore, FIFO message, task context switch). The TC execution is managed in task CMD_SEQ (§3.2.6.4). In order of priority CMD_SEQ is preceeded by: • Virtual Machines, which are low duty-cycle tasks (see §3.2.5.2 and RD6) • HS, which runs only when science data is being received from the DRCU. This occurrence is not expected to	

	happen when a TC is received because TC dispatching by the CDMS is timed to the execution duration of the TCs, meaning that no TCs will be sent to the instrument before the previous one has been completed; the only exception is the "Abort" command, which is the only immediate command implemented by the SPIRE OBS, and which only consists in stopping the Hard_VM task (§3.2.5.2) by disabling IRQ3 interrupt. • VM_SVC, which runs occasionally • LS, which is mainly used by the HK_ASK_i which, on turn, have lower priority than CMD_SEQ Assuming a TC (6,1) "Memory Load" maximum-size TC as the sizing case, most of the execution time is taken by CRC computations and read/write operations; we estimate an execution time of 0.5 msec The total required time to receive unpack and execute the TC is then ~ 1msec. The goal of this requirement is to be able to receive, unpack and process up to 25 TCs per second; this corresponds to 1 TC every 40 msec, largely met by our design.	
OBS-UR-TC5	Function Report_Generator in task CMD_SEQ (§3.2.6.4) generates the required TC acceptance and execution reports. The function will execute according to the "Ack bits" setting in the correspondent TC.	
OBS-UR-TC6	Both "immediate" and "normal" commands are passed by TMTC to CMD_SEQ via the TC_POOL memory pool. The only immediate command is the "Abort Measurement" command; this will act to disable the IRQ3 interrupt which triggers the Hard_VM and will not interfere with other previously processed TCs. Hence the foreseen architecture works equally well for "immediate" and "normal" commands.	Partially available in OBS Version 1
OBS-UR-TC7	The only immediate command is the "Abort Measurement" command. Consisting of a single statement (disable IRQ3) its execution time largely meets the requirement.	
OBS-UR-TC8	Function TC_Acceptance in task CMD_SEQ (§3.2.6.4) will perform all required validity checks (AD4).	
OBS-UR-TC9	Deleted	
OBS-UR-TC10	Validity checks of the TC packet header and application data header are performed in function TC_Acceptance of task CMD_SEQ (§3.2.6.4). If the packet is found invalid, the reject report generation is immediately initiated and the task CMD_SEQ exits.	
OBS-UR-TC11	See above.	
OBS-UR-TC12	The estimated time required for a TC packet reception, unpack and processing is 0.5 msec in total (see OBS-UR-TC4 above). The generation, packing and dispatch of TC verification	

	report TM packets take a similar amount of time. the requirement is easily met.	
OBS-UR-TC13	Deleted	
OBS-UR-TC14	After execution of the TC_acceptance function, the task CMD_SEQ passes control to the Command_Parser function .	
OBS-UR-TC15	Function Command_Parser in CMD_SEQ uses the Report_Generator function (in the same task) to generate report TM packets that reflect the success/failure status in the TC execution. Progress reports will be issued only during the execution of observing procedures (execution speed makes this feature useless in all other cases). Observing procedures are handled by VM codes run by Hard_VM task (§3.2.5.2). This task will implement opcodes to generate proper FIFO messages to trigger the VM_SVC task (§3.2.6.11) that, finally, will generate the progress report TM packets.	
OBS-UR-TC16	See above.	
OBS-UR-TC17	Deleted	
OBS-UR-TC18	See OBS-UR-TC6 above.	
OBS-UR-TC19	This requirement is met by the adopted DPU memory management scheme (§3.2.4.1). table management is handled by the Table_Handler function in task CMD_SEQ (§3.2.6.4).	
OBS-UR-TC20	The transmission of TC verification packets is handled by the Report_Generator function in task CMD_SEQ (§3.2.6.4); this function executes accordingly to the “Ack bits” in the TC packet header.	
OBS-UR-TC21	Function Transfer_Handler in ISR_1553 (§3.2.5.1) checks that the TC count in the TC Packet Transfer Descriptor is different from the one of the previously received TC packet. In case it is different by more than one unit (jump in TC packet counter) the function will initiate the generation of an event	
OBS-UR-TC22	The OBS shall be able to execute a peak-up procedure, interacting with the spacecraft.	N/A in OBS Version 1
OBS-UR-TC23	The Hard_VM and Soft_VM_i tasks (3.2.5.2 and 3.2.6.10) allow the execution of command lists stored on-board and loaded/modified via TCs.	

4.3 Telemetry Generation Requirements

Req. ID	Verification	Notes
OBS-UR-TM1	Tasks CMD_SEQ (§3.2.6.4), HK_ASK_i (§3.2.6.6), LS, (§3.2.6.5), HS (§3.2.6.7), and AUTONOMY (§3.2.6.9)	

	generate all TM packets specified in AD9.	
OBS-UR-TM2	The tasks responsible for the generation of all types of TM packets will packetise data accordingly to AD4 and AD9. The Transfer Layer Protocol specified in AD4, used by the OBS to send TM packets, is implemented in the OBS by the combination of the ISR_1553 Interrupt Service Routine (§3.2.5.1) and the TMTC task (§3.2.6.3).	
OBS-UR-TM3	The TM packet assembly will be started with the memory block allocation and the compilation of the TM packet header, which includes the time info, is done before the application data is written.	
OBS-UR-TM4	All TM packets will contain at the beginning of the application data the OBSID and the BBID.	
OBS-UR-TM5	Science data memory pool size meets this requirement (§3.2.4.2).	
OBS-UR-TM6	Module ISR_1553 (§3.2.5.1) implements a simplified TFL protocol that neglects the PTR/PTC mechanisms and uploads a new TM packet based on the RT_info parameter (read from the data word coming with the Subframe Sync) which notifies the RTs which is the one allowed for TM transfer in the current SubFrame.	
OBS-UR-TM7	The Frame_Interpreter function in task HS (§3.2.6.7) can perform subarray selection or data averaging based on configuration parameters stored on-board and uploadable via TC. By default, it will fill the TM science packets with raw science frames.	
OBS-UR-TM8	COCA: The list of HK parameters to be monitored is modifiable via TCs in task HK_MON (§3.2.6.8) TEST: this is transparent to the OBS as the test frames are being generated by the DRCU. TRNS: see OBS-UR-TM7.	N/A in OBS Version 1
OBS-UR-TM9	Once enabled, tasks HK_ASK_i (§3.2.6.6) run in batch independently from the instrument operating mode.	
OBS-UR-TM10	Function DPU_Cmd in task LS (§3.2.6.5) implements a commanding scheme similar to the one used to send commands to the DRCU, to read DPU H/W and S/W parameters.	
OBS-UR-TM11	Function HK_i_main in task HK_ASK_i (§3.2.6.6) stores as a DPU parameter the time when the trigger HK_i_SEMA semaphore signal was received. In the course of the HK packet building, the DPU_Cmd function in task LS (§3.2.6.5) will write that parameter in the proper location of the HK packet in HK_POOL.	
OBS-UR-TM12	The content of HK packets are defined in on-board tables stored in tabellone (§3.2.4.1), modifiable via TCs, used by the task HK_ASK_i (§3.2.6.6).	

OBS-UR-TM13	The OBS shall provide only actual values of the HK parameters and not changes (or delta values) since the last readout.	
OBS-UR-TM14	Tasks HK_ASK_0 and HK_ASK_1 (§3.2.6.6) will be run by default at start-up, providing the required HK packets at the required sampling using predefined tables on-board.	
OBS-UR-TM15	The OBS implements 4 independent HK_ASK_i tasks.	
OBS-UR-TM16	The HK packet sampling period is read from a TC and made available by the HK_Handler function of task CMD_SEQ (§3.2.6.4) to ask HK_ASK_i (§3.2.6.6).	
OBS-UR-TM17	This requirement is met with the possibility to generate, using VM code in Hard_VM (§3.2.5.2) and Soft_VM_i (§3.2.6.10) tasks, packets containing HK parameters sampled at whatever rate.	N/A in OBS Version 1
OBS-UR-TM18	Task HS (§3.2.6.7) will put into TM packets the maximum possible number of raw science frames.	

4.4 Synchronization Requirements

Req. ID	Verification	Notes
OBS-UR-SY1	At each Frame Sync received from the CDMS the module ISR_1553 (§3.2.5.1) will activate the highest-priority task TIME (§3.2.6.2), responsible for the synchronization. The adopted design easily meets the requirement.	
OBS-UR-SY2	Whenever the time has not yet been synchronised (e.g., after switch on or reset), the OBS shall set to 1 the MSB of the time field in the header of TM packets.	
OBS-UR-SY3	The Send function in task LS (§3.2.6.5) will store in DPU memory the time at which the “SyncDRCUCounters” command is being transmitted to the DRCU. Considering that the LS task can be interrupted by the Hard_VM task (§3.2.5.2) at any moment for no more than about 2 msec, the requirement is easily met.	
OBS-UR-SY4	The drift between the S/C clock and the DPU clock is updated every second by the TIME task (§3.2.6.2) and made available as an HK parameter.	

4.5 Testing and Maintainance Requirements

Req. ID	Verification	Notes
OBS-UR-SM1	Entering the instruments Test Mode shall not require disabling of fault management (autonomy) functions. TBD	N/A in OBS Version 1
OBS-UR-SM2	The IDLE task (§3.2.6.12) may be used to perform DPU memory checks.	
OBS-UR-SM3	An OBS software verification facility (for PROM,	N/A in OBS

	EEPROM, RAM code) shall be provided on board. TBD	Version 1
OBS-UR-SM4	The OBS image is stored on EEPROM	
OBS-UR-SM5	See §3.2.2	
OBS-UR-SM6	The Memory_LoadandDump function of task CMD_SEQ (§3.2.6.4) implements service 6 of AD4. Writing into EEPROM is provided in the Command_Parser function of task CMD_SEQ. Reading and checksum are performed by the Boot Software (see RD8).	
OBS-UR-SM7	Requirement met performed by the Boot Software (see RD8).	
OBS-UR-SM8	Service 17 of AD4 is provided in the Command_Parser function of task CMD_SEQ (§3.2.6.4).	
OBS-UR-SM9	Tasks HK_ASK_i (§3.2.6.6), Soft_VM_i (§3.2.6.10) and Hard_VM (§3.2.5.2) can be stopped/started by disabling/enabling timers and/or interrupts.	
OBS-UR-SM10	Procedures are implemented as VM codes stored in tables in tabellone (§3.2.4.1).	
OBS-UR-SM11	This requirement is not met. A waiver will be requested.	N/A in OBS Version 1

4.6 Autonomy Function Requirements

Req. ID	Verification	Notes
OBS-UR-AF1	See task HK_MON (§3.2.6.8).	N/A in OBS Version 1
OBS-UR-AF2	Procedures are implemented as VM programs stored in tables in tabellone (§3.2.4.1). Task HK_MON (§3.2.6.8) can start Hard_VM with a predefined VM code to be executed.	N/A in OBS Version 1
OBS-UR-AF3	Task HK_MON (§3.2.6.8) will trigger the AUTONOMY task (§3.2.6.9) upon detection of an anomaly.	N/A in OBS Version 1
OBS-UR-AF4	See OBS-UR-AF3	N/A in OBS Version 1
OBS-UR-AF5	Since autonomy functions are implemented as VM codes, this requirement is met by the ability to generate events and TM packets from within task Hard_VM (§3.2.5.2).	N/A in OBS Version 1
OBS-UR-AF6	The OBS shall provide all the event packets with a counter that permits the unambiguous identification of missing packets. TBD	N/A in OBS Version 1
OBS-UR-AF7	The AUTONOMY task (§3.2.6.9), as well as anomaly detection codes in the LS (§3.2.6.5) and HS (§3.2.6.7) tasks, will implement a "transition edge" sensing mechanism for anomaly conditions.	N/A in OBS Version 1
OBS-UR-AF8	Control actions will be implemented as VM codes and, as such, handled by task HK_MON (§3.2.6.8).	N/A in OBS Version 1
OBS-UR-AF9	Autonomy functions will be implemented as VM codes and ,as such, a pointer to a table ID containing the appropriate	N/A in OBS Version 1

	program will be associated to any anomaly condition detected: task HK_MON can be told to disable such associations via TC.	
OBS-UR-AF10	HK monitoring parameters used by task HK_MON are held in tables in tabellone (§3.2.4.1), as well as autonomy function VM codes; as such thay can be modified via TC.	N/A in OBS Version 1
OBS-UR-AF11	Operation/activities will be implemented as VM codes. Task Hard_VM (§3.2.5.2) provides opcodes to generate progress reports.	N/A in OBS Version 1
OBS-UR-AF12	Observing mode initialization is performed in VM code and, as such, completely configurable from the ground.	N/A in OBS Version 1
OBS-UR-AF13	This functionality is provided in the Command_Parser function of task CMD_SEQ (§3.2.6.4).	N/A in OBS Version 1
OBS-UR-AF14	Critical subsystem commands will only be sent via TCs with service (8,4) and not as part of a VM code. This requirement will be met using service 8,1 (AD4).	N/A in OBS Version 1

4.7 Functional Requirements

Req. ID	Verification	Notes
OBS-SUR-FU1 OBS-SUR-FU2 OBS-SUR-FU3 OBS-SUR-FU4 OBS-SUR-FU5 OBS-SUR-FU6 OBS-SUR-FU7 OBS-SUR-FU8 OBS-SUR-FU9	These requirements are met by the possibility to execute these procedures either as VM codes run in Hard_VM (§3.2.5.2) or Soft_VM_i (§3.2.6.10), or as sequences of direct DRCU commands sent via TCs and managed by the Command_Parser function of task CMD_SEQ (§3.2.6.4).	
OBS-SUR-FU10	The design of tasks LS (§3.2.6.5) and HS (§3.2.6.7) meets the requirement.	
OBS-SUR-FU11	Task HS (§3.2.6.7) is interrupt driven. Science Frame checksum control is done on-the-fly while reading from the FIFOs and frames are directly written into SD_POOL memory blocks, thus minimizing memory read/write overhead.	

4.8 Operating Modes Requirements

Req. ID	Verification	Notes
OBS-SUR-GE1	Procedures implemented as VM codes. Beside the main procedure that can be run from Hard_VM (§3.2.5.2), up to three parallel procedures can be run on the three Soft_VM_i tasks (§3.2.6.10).	
OBS-SUR-GE2	Requirement implemented by the Boot Software (RD8)	

OBS-SUR-GE3	The task-oriented OBS architecture meets this requirement.	
OBS-SUR-GE4 OBS-SUR-GE5 OBS-SUR-GE6	All instrument settings can be executed as VM code.	
OBS-SUR-GE7	Anomalies recovery procedure are implemented as VM code and are triggered by task HK_MON (§3.2.6.8). While task Hard_VM (§3.2.5.2) is running, the HK_ASK_i task (§3.2.6.6) is also running.	N/A in OBS Version 1
OBS-SUR-GE8	All observing procedures are implemented as VM code.	
OBS-SUR-GE9 OBS-SUR-GE10	The HS task design (§3.2.6.7) ensures that the OBS is fast enough to support these data rates.	
OBS-SUR-GE11	This requirement has to be met by the observing procedure, which is implemented as VM code.	
OBS-SUR-GE12	All instrument settings can be executed as VM code.	
OBS-SUR-GE13	Most of the degraded operations can be handled in VM code. Reduced telemetry rate by sub-array selection can be performed within task HS (§3.2.6.7) by using the TM_Red_info data from CMD_SEQ.	
OBS-SUR-GE14	Mode transitions procedures are implemented as VM code; task Hard_VM (§3.2.5.2) can be run by TC from CMD_SEQ (§3.2.6.4).	N/A in OBS Version 1