SPIRE

On Board Software

Software Specification Document

Document No: SPIRE-IFS-PRJ-001036

Issue: 1.0

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	SAp-SPIRE-Cca-24-00
AD9	SPIRE Data ICD	SPIRE-RAL-DOC-001078

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.

2 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.

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.

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.

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.

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 OBS Architecture

3.1 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 main blocks.

3.1.1 The SEG_DMDA Segment

This segment size is ~200 kW, and hosts the static variables used by the OBS. The biggest chunk of SEG_DMDA, called **tabellone**, is used to hold all the tables that are manageable from the ground via standard TeleCommands; examples are the HouseKeeping packets definition tables, Jiggle tables, Virtual Machine Codes, etc. The size of tabellone is 128 kW (32-bits words).

A table is characterised 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. The set of parameters that characterize each table are 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 to a certain degree of fragmentation in tabellone (holes are left when tables are deleted), that can be removed either via a dedicated TC or automatically by the TABLER task (see §3.5.12).

There is another reserved area in SEG_DMDA segment, which is used by the TABLER task to perform the defragmentation. This is called swappone and its size is 8 kW.

3.1.2 The SEG_CHEAP Segment

This segment is used by VIRTUOSO to hold the **Memory Pools**. Memory Pools are sections of memory where blocks of required size can be allocated by VIRTUOSO upon request by the OBS. When the OBS has finished using the block, it commands VIRTUOSO to release the block back into the Pool. The allocation/deallocation of blocks in the Memory Pools is entirely managed by

VIRTUOSO. Each pool is specified with the maximum number of blocks that can be allocated, and by the size of each block. The different Memory Pools defined in the SPIRE OBS are defined in the VIRTUOSO Project File (delivered together with the OBS code) as follows:

Pool Name	Usage	# of Blocks	Block Size (bytes)
TC_POOL	Telecommand Packets	8	512
EV_POOL	Event Telemetry Packets	28	512
RP_POOL	TC Verification/Execution TM Packets	32	2048
HK_POOL	HouseKeeping TM Packets	32	2048
SD_POOL	Science Data TM Packets	128	2048

Table 3-1 List of VIRTUOSO Memory Pools used in the OBS

3.2 OBS Tasks

The OBS is divided into a series of tasks with certain 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	Keep 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
AUTONOMY	Task that handles Event Packet generation and recovery procedures upon reception of anomaly messages	6
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 consequent reception of parameters	7
CMD_SEQ	Checks the header of the received TC packets, issues appropriate TC verification reports and, upon positive verification, interprets the commands and executes them.	8
HK_ASK_0	First task that generates HK packets	9
HK_ASK_1	Second task that generates HK packets	9
HK_ASK_2	Third task that generates HK packets	9
HK_ASK_3	Fourth task that generates HK packets	9
HK_MONITOR	It monitors the HK parameter and, in case of critical values, invokes the appropriate Autonomy Function	9
TABLER	It performs the SEG_DMDA defragmentation.	10

IDLE	Performs TBD memory checks	11

Table 3-2 OBS Task list

3.3 Inter-Task Communication

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.

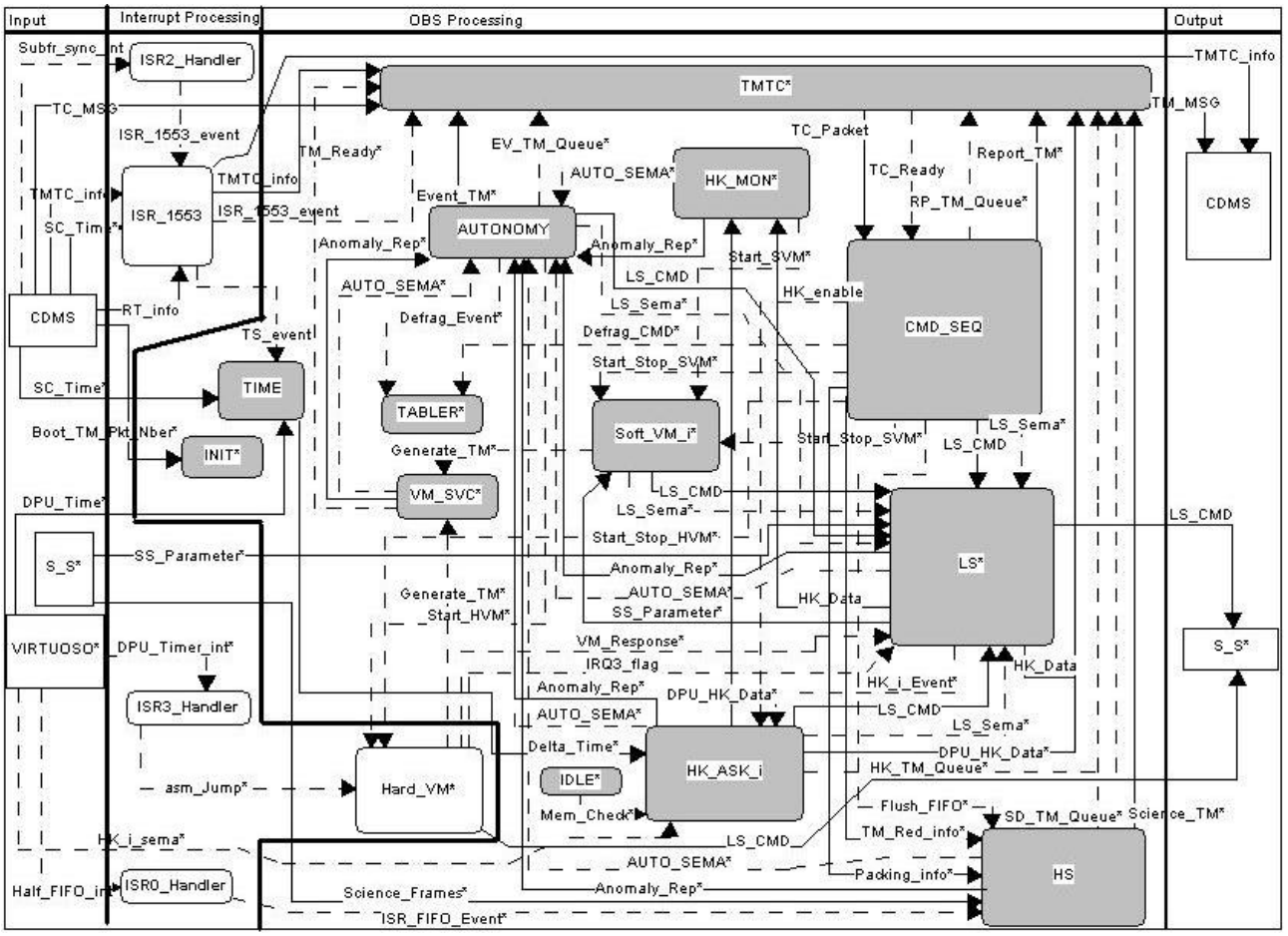

Figure 3-1 OBS Tasks Interconnection Diagram

3.3.1 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:
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

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

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

3.3.2 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
TC_READY	Signals that a new TC has been downloaded from the CDMS and is ready to be verified and executed	TMTC	CMD_SEQ
FRAG_SEMA	Signals that tabellone needs to be defragmented (see §3.1.1)	AUTONOMY	TABLER
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 3-4 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)

3.3.3 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	36	10
HK_TM_QUEUE	Notifies that a new HK TM packet is ready on the HK_POOL	HK_ASK_i	TMTC	16	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	8
LS_LP_QUEUE	Notifies that a low-priority command has to be sent to the SubSystem	CMD_SEQ, HK_ASK_i	LS	1024	8
VM_TM_QUEUE	Notifies that an event/report packet is to be sent	Hard_VM, Soft_VM_i	VM_SVC	64	6
ANOMALY_LP_QUEUE	Notifies a low-priority anomaly		AUTONOMY	64	10
ANOMALY_HP_QUEUE	Notifies a high-priority anomaly		AUTONOMY	64	10

Table 3-5 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)

3.4 C Interrupt Service Routines

3.4.1 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.

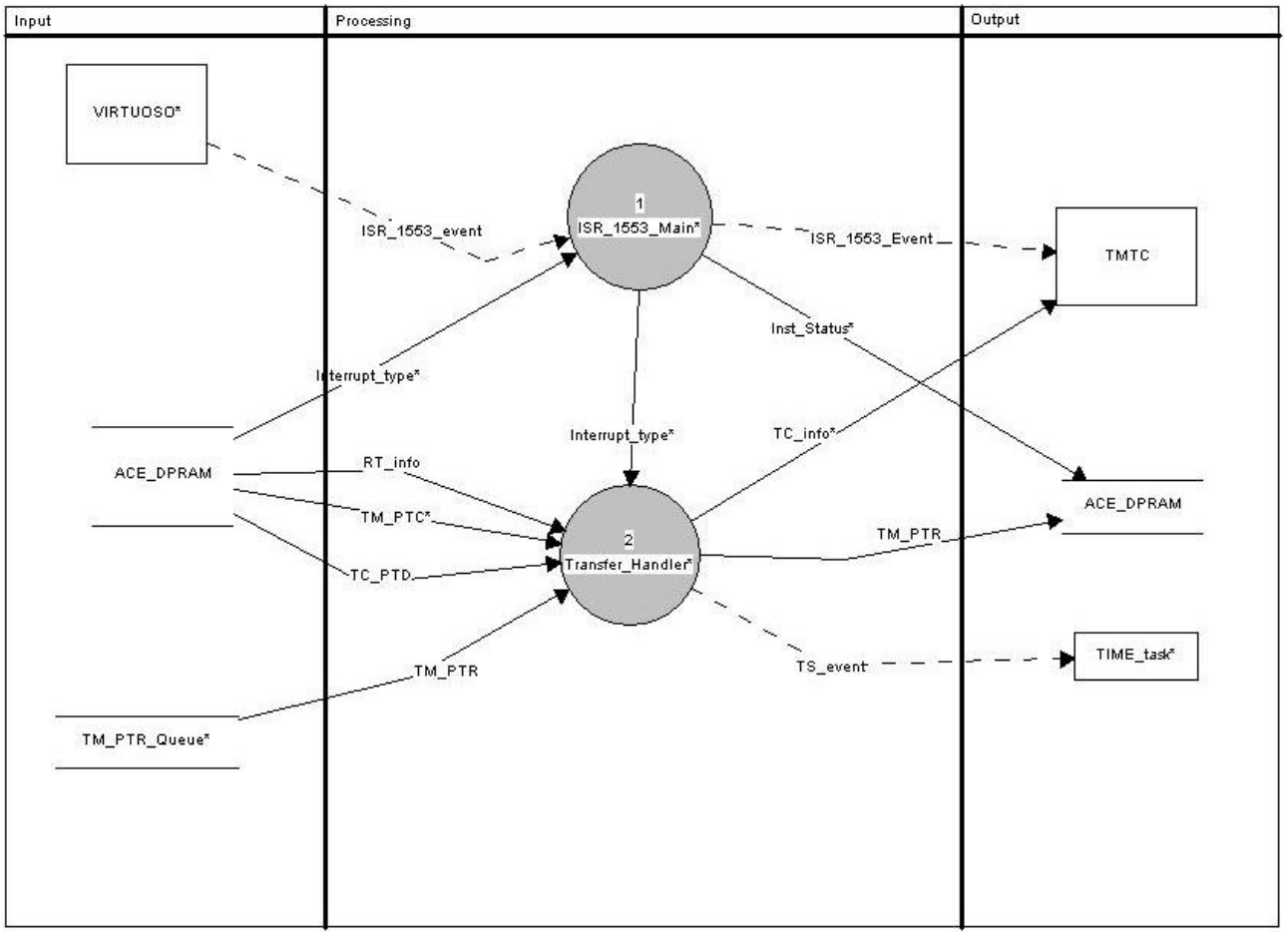

Figure 3-2 ISR_1553 Module Functional Decomposition

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).

3.4.2 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 stored on the DPU/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 bck/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 §3.5.5).

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 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.

3.5 Tasks Description

3.5.1 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.

3.5.2 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 Δt. 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 Δt computed by TIME_task will be added. Δt will be also made available to HK_ASK_i task to include it as a DPU HK parameter.

3.5.3 TMTC task

This task, together with the ISR_1553 interrupt service routine (see §3.4.1), handles the interface with the spacecraft CDMS. It is enabled by the ISR_1553_EVENT raised by ISR_1553.

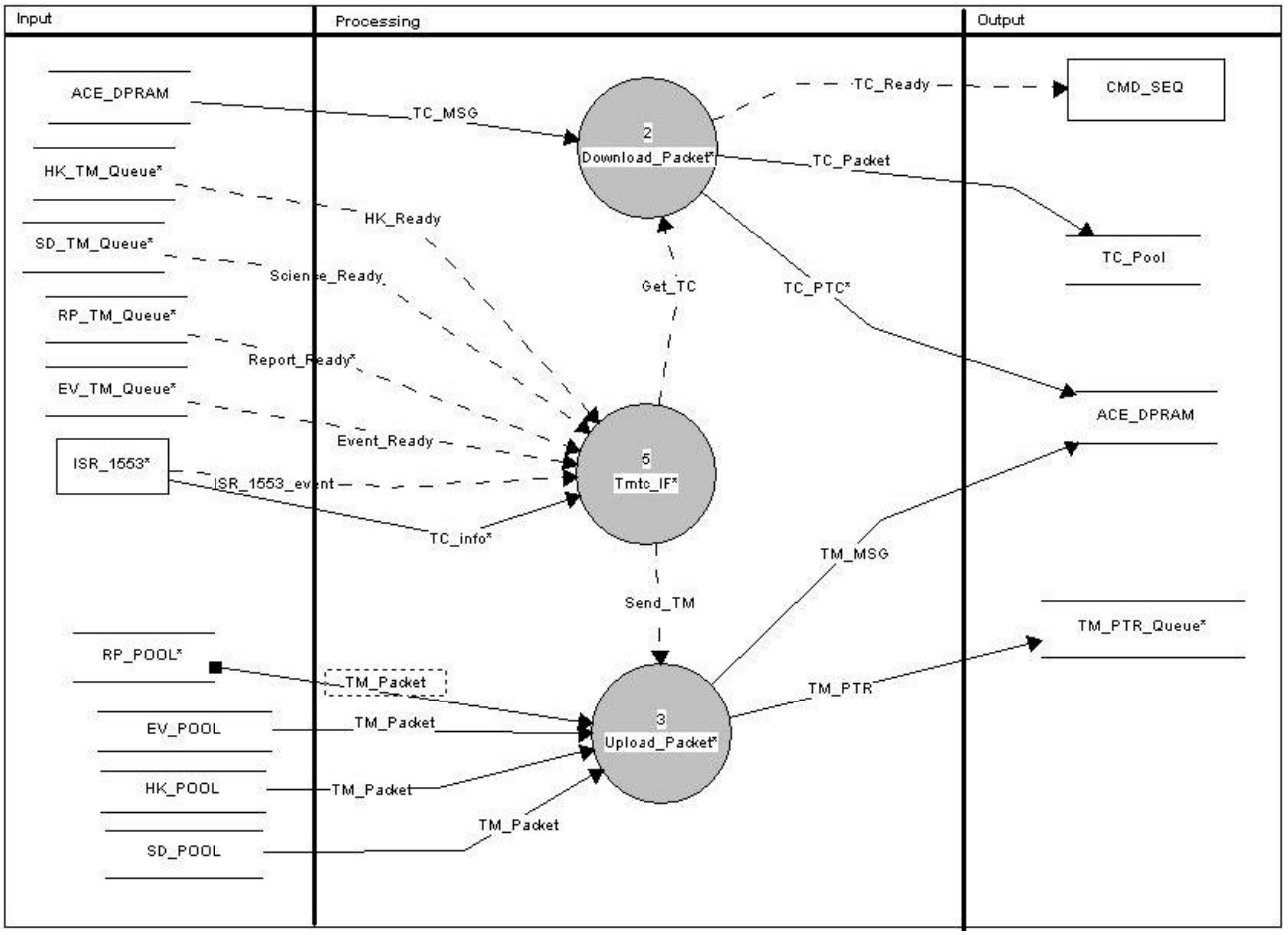

Figure 3-3 TMTC Task Functional Decomposition

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 §3.3.3), 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 §3.4.1).

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

3.5.4 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" semaphores is signaled from task TMTC, notifying the availability of a new TC. When this happens, CMD_SEQ reads 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, 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.

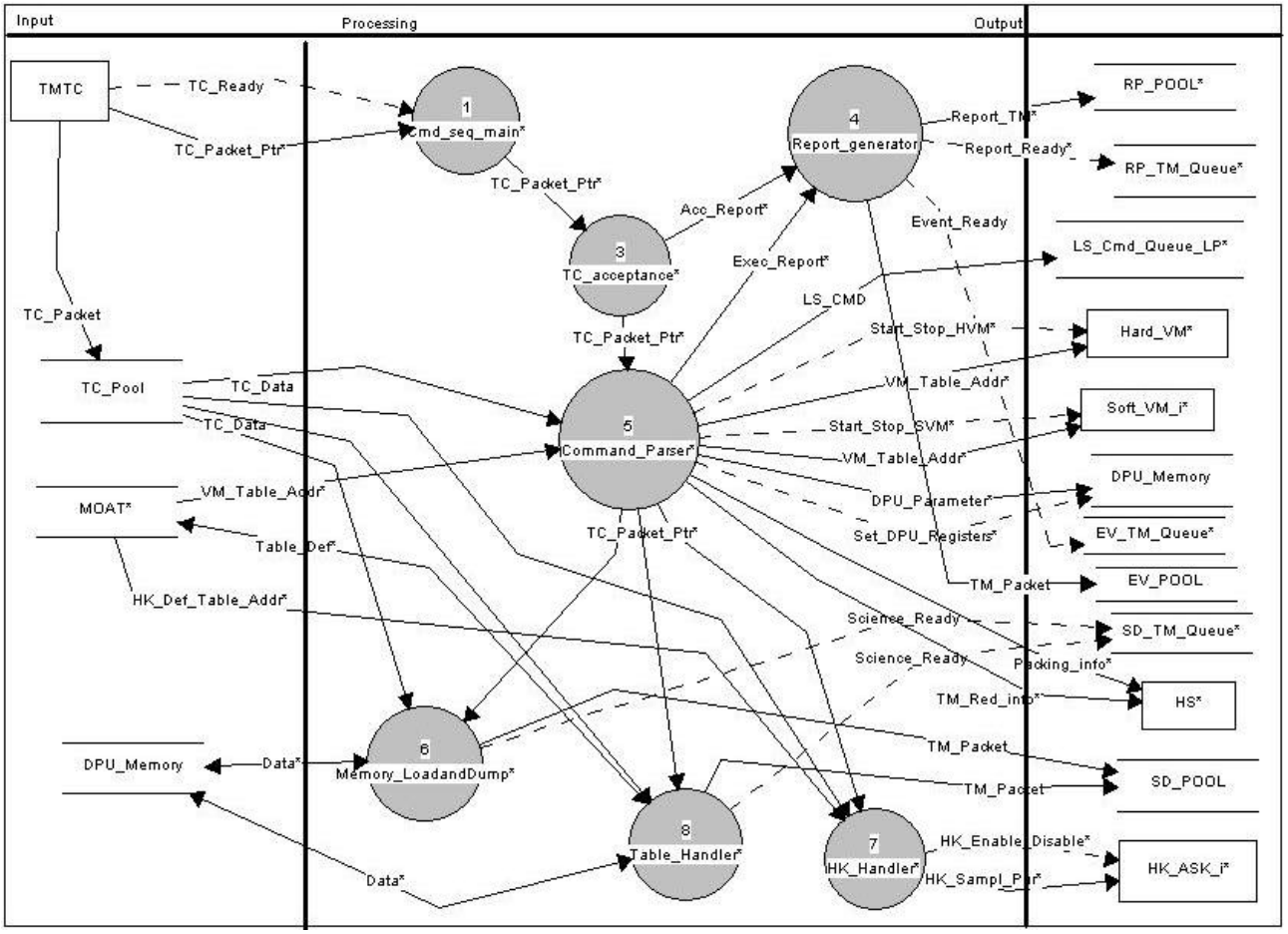

Figure 3-4 CMD_SEQ Task Functional Decomposition

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 RP_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 Rest_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 semaphores (see §3.3.2) 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 §3.1.1). 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.

3.5.5 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 §3.3.2); 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.

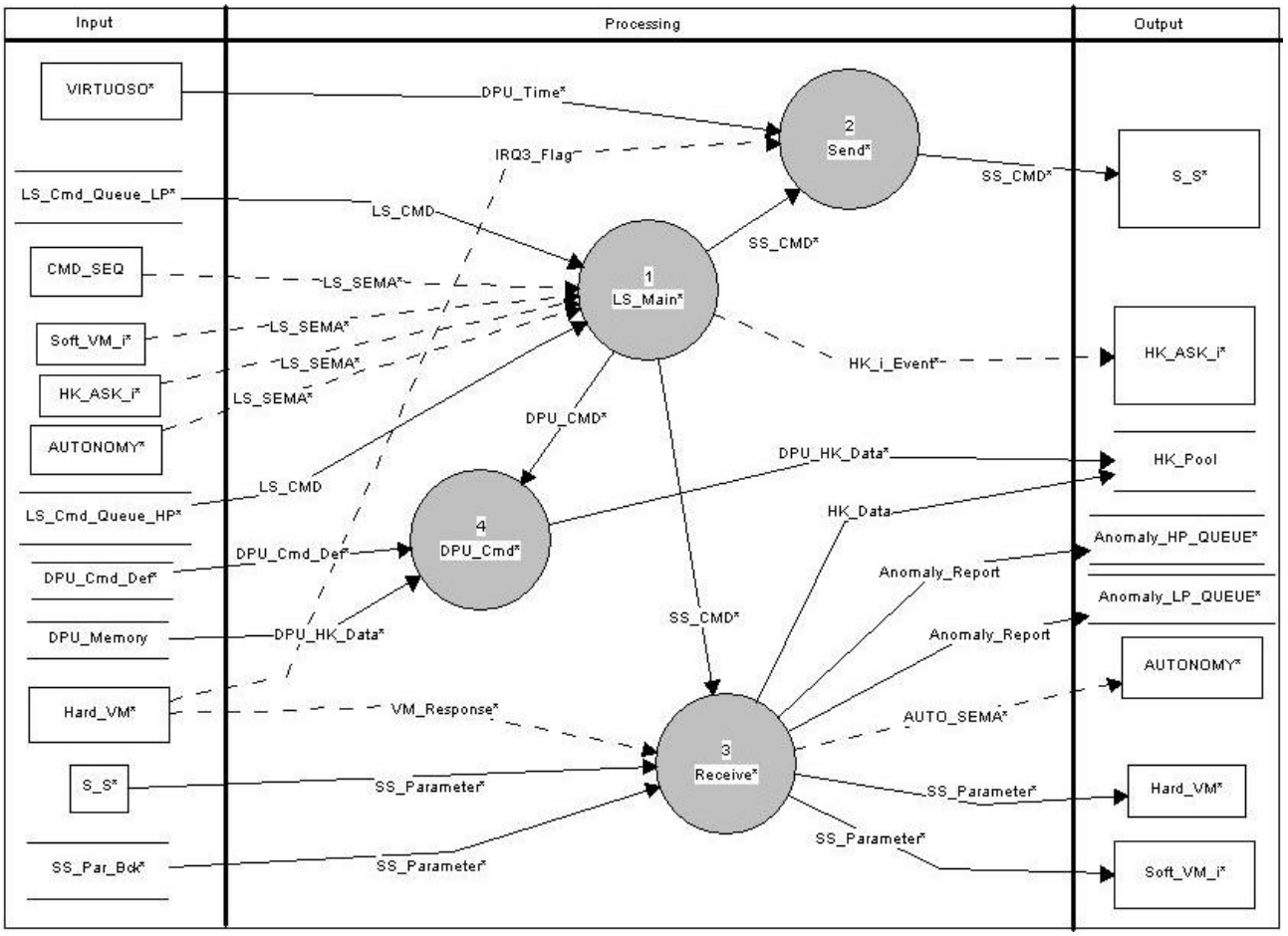

Figure 3-5 LS Task Functional Decomposition

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 (might be used 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 it will be explained in §3.4.2, 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 will read the DPU memory locations where the Sub-System interface "receive" register is mapped, store its contents in a back-up memory location and raise 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 stored it task.

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 a set command (to assign certain values to some Sub-System parameter) and was correctly interpreted and executed, the Sub-Systems will echo the exact copy of the command word. If the command was a get command (to read the current values of some Sub-System parameter – as it is the case for HouseKeeping parameters) the Sub-Systems will return 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; otherwise an Anomaly_Report message shall be sent on the ANOMALY FIFO QUEUEs and the AUTO_SEMA semaphore shall be raised to signal the AUTONOMY task.

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.

3.5.6 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 message is enqueued on the AUTONOMY FIFO QUEUEs and the AUTO_SEMA is raised to notify 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 §3.5.5); 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 sends a message containing the address of the packet in HK_POOL to the TMTC task. 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.

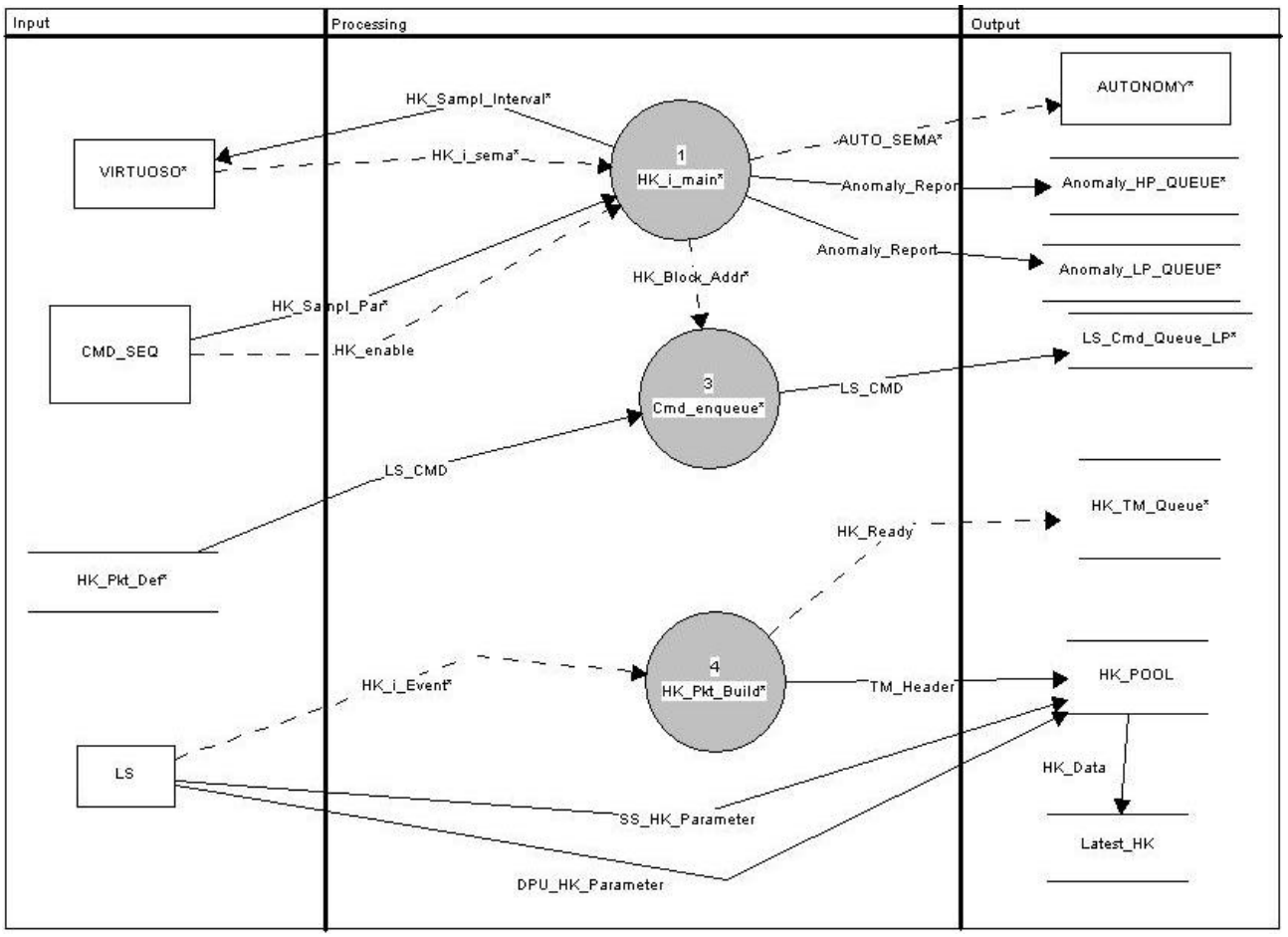

Figure 3-6 HK_ASK_i Task Functional Decomposition

3.5.7 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 (IRQ 0). This interrupt is served by the Interrupt_Handler_ISR0, 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.) an Anomaly_Report message will be enqueued on the AUTONOMY FIFO QUEUEs and the 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.

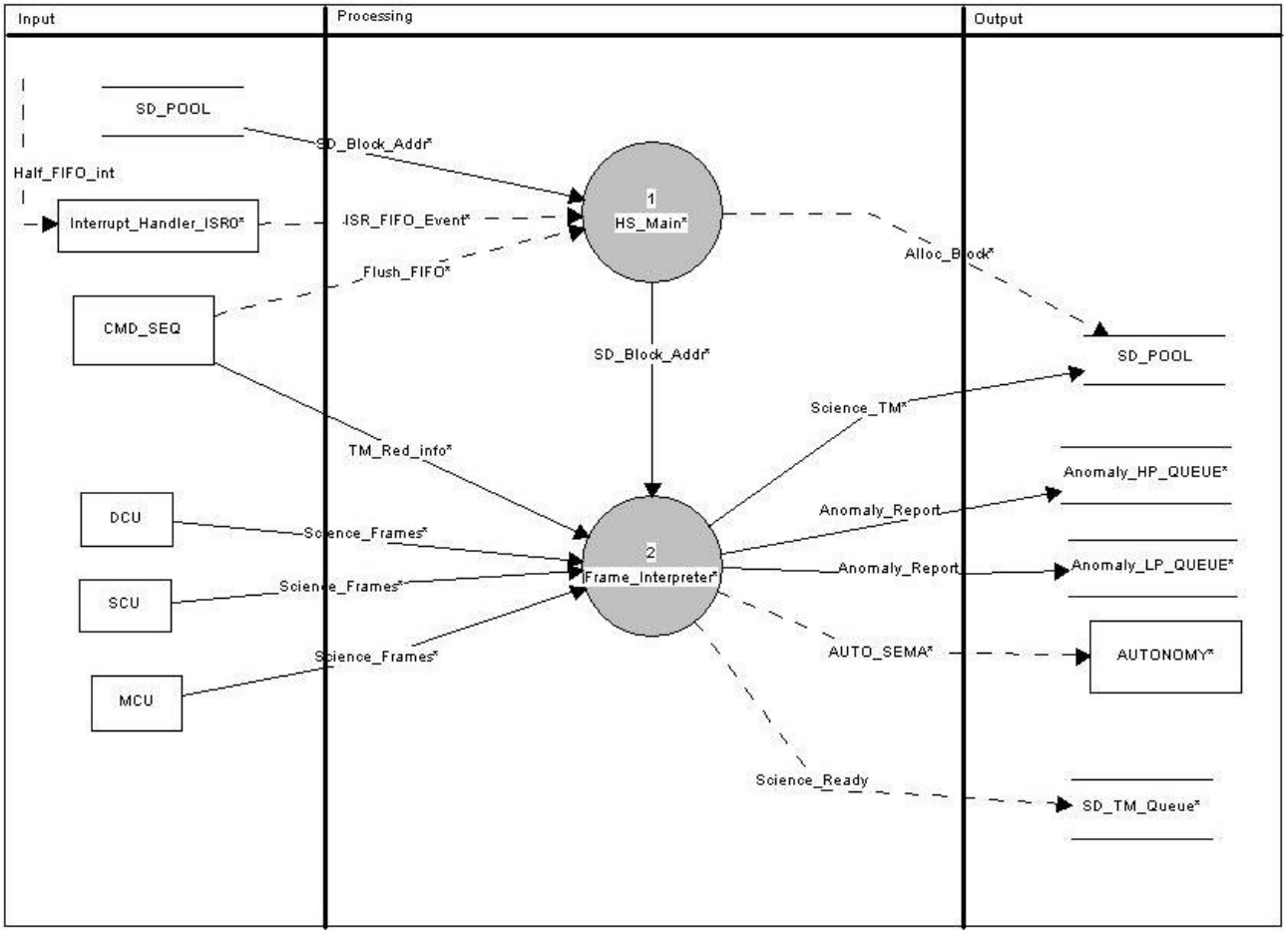

Figure 3-7 HS Task Functional Decomposition

3.5.8 HK_MON task

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_Report message shall be enqueued on the AUTONOMY FIFO QUEUEs and the AUTO_SEMA will be raised to signal the AUTONOMY task.

HK_MON will also monitor the fragmentation status of tabellone and issue the Defrag_Event to the task TABLER to start defragmentation.

3.5.9 AUTONOMY task

This task is triggered by the AUTO_SEMA semaphore, which can be raised from several locations in the OBS. The task will then read from the AUTONOMY_HP_QUEUE and AUTONOMY_LP_QUEUE (in this order) the Anomaly_Report 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 EVENT_TM_QUEUE FIFO (identified as the Event_Ready control flow in Figure 3-1). 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. The only recovery procedure currently present is the defragmentation of tabellone. The VIRTUOSO Defrag_Event is raised by AUTONOMY task upon reception of the proper Autonomy_Report on the AUTONOMY FIFO QUEUEs and the raising of the AUTO_SEMA semaphore by task HK_MON.

3.5.10 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.

3.5.11 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 either generates the proper execution reports or it writes an Autonomy_Report on the AUTONOMY FIFO QUEUEs and raises the AUTO_SEMA semaphore to signal the AUTONOMY task.

3.5.12 TABLER task

This task is responsible to maintain the Table management implementation for the SPIRE DPU, by performing the defragmentation of tabellone in the SEG_DMDA memory segment (§3.1.1).

The Tabler_main function can be triggered either by a dedicated command by CMD_SEQ or automatically by AUTONOMY when tabellone is more than xx% defragmented (as monitored by HK_MON). The MOAT_Stack table contains the list, sorted by address, of occupied blocks in tabellone. Function Tabler_main calls the MOAT_Stack_Parser function that starts parsing the MOAT_Stack records 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.

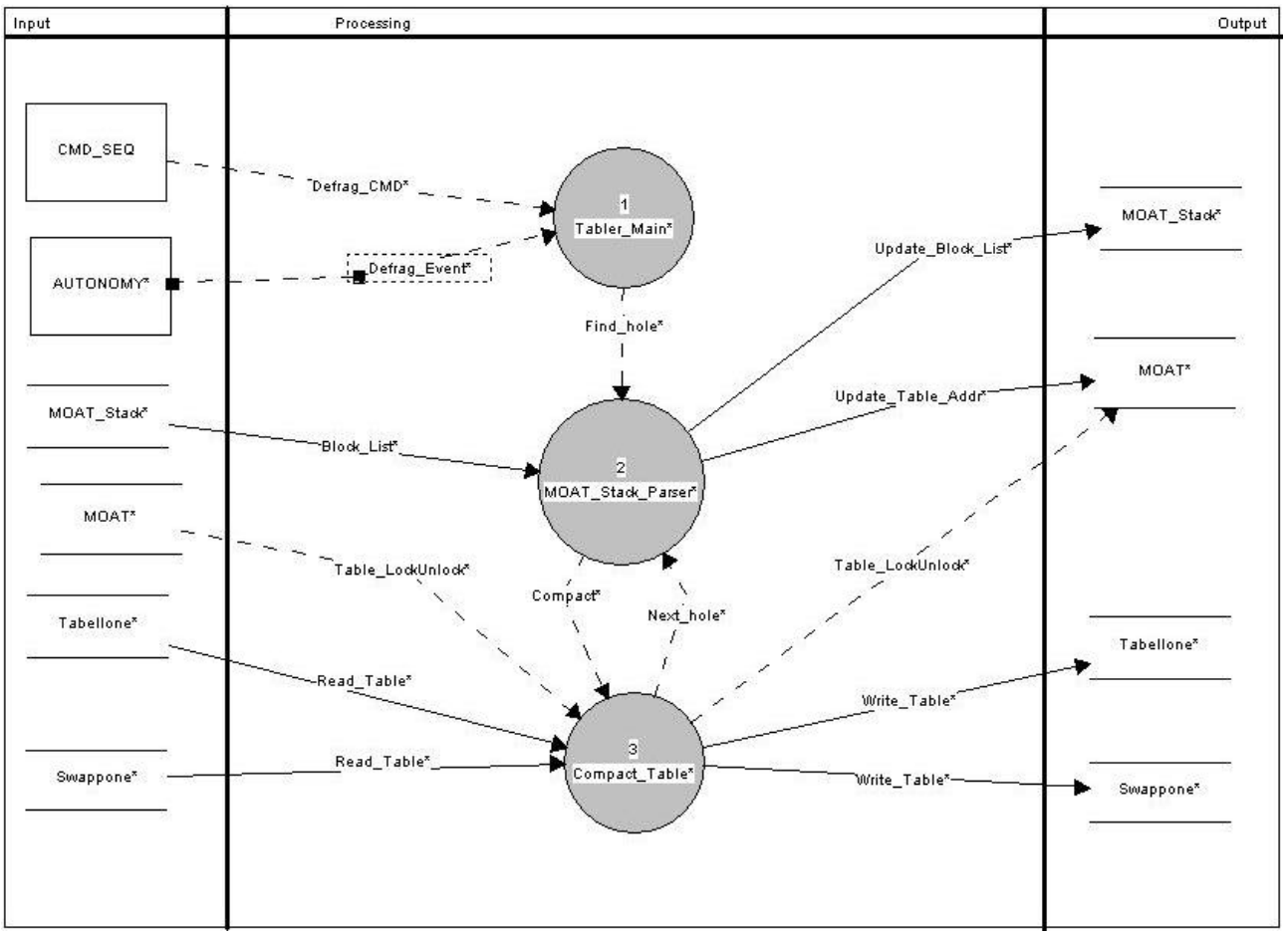

Figure 3-8 TABLER Task Functional Decomposition

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 a 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.

3.5.13 IDLE task

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
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
OBS-UR-TC1	The Command_Parser routine in the CMD_SEQ task (§3.5.4) will decode the [Type, Subtype, Function_ID, Activity_ID] combination using a series of nested “switch” statements.
OBS-UR-TC2	
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 (§3.4.1) and the TMTC task (§3.5.3).

OBS-UR-TC4	TC reception and unpacking is immediate because ISR_1553 (§3.4.1) is triggered by an event (§3.3.1) raised by and Interrupt Service Routine, and the task TMTC (§3.5.3) has the highest priority (see table in §3.2) after the INIT task (§3.5.1), which runs only at start-up, and TIME task (§3.5.2) 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, TC_POOL memory pool block allocation) The TC execution is managed in task CMD_SEQ (§3.5.4). In order of priority CMD_SEQ is preceeded by: • Virtual Machines, which are low duty-cycle tasks (see §3.4.2 and RD6) • HS, which runs only when science data is being received from the DRCU. This occurrence will never 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 (ref ??); 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 (§3.4.2) 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 (§3.5.4) 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.
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 (§3.5.4) will perform all required validity checks (AD4).
OBS-UR-TC9	
OBS-UR-TC10	Validity checks of the TC packet header and application data header are performed in function TC_Acceptance of task CMD_SEQ (§3.5.4). 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	
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 (§3.4.2). This task will implement opcodes to generate proper FIFO messages to trigger the VM_SVC task (§3.5.11) that, finally, will generate the progress report TM packets.
OBS-UR-TC16	See above.
OBS-UR-TC17	
OBS-UR-TC18	See OBS-UR-TC6 above.
OBS-UR-TC19	This requirement is met by the adopted DPU memory management scheme (§3.1.1). table management is handled by the Table_Handler function in task CMD_SEQ (§3.5.4).
OBS-UR-TC20	The transmission of TC verification packets is handled by the Report_Generator function in task CMD_SEQ (§3.5.4) ; this function executes accordingly to the “Ack bits” in the TC packet header.
OBS-UR-TC21	Function Transfer_Handler in ISR_1553 (§3.4.1) 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.
OBS-UR-TC23	The Hard_VM and Soft_VM_i tasks (3.4.2 and 3.5.10) allow the execution of command lists stored on-board and loaded/modified via TCs.

4.3 Telemetry Generation Requirements

Req. ID	Verification
OBS-UR-TM1	Tasks CMD_SEQ (§3.5.4), HK_ASK_i (§3.5.6), HS (§3.5.7), and AUTONOMY (§3.5.9) 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 (§3.4.1) and the TMTC task (§3.5.3).
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 (§3.1.2).
OBS-UR-TM6	Module ISR_1553 (§3.4.1) 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 (§3.5.7) 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 (§3.5.8) TRNS: see OBS-UR-TM7. TEST: this is transparent to the OBS as the test frames are being generated by the DRCU.
OBS-UR-TM9	Once enabled, tasks HK_ASK_i (§3.5.6) run in batch independently from the instrument operating mode.
OBS-UR-TM10	Function DPU_Cmd in task LS (§3.5.5) 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 (§3.5.6) 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 (§3.5.5) 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 (§3.1.1), modifiable via TCs, used by the task HK_ASK_i (§3.5.6).
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 (§3.5.6) 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 (§3.5.4) to ask HK_ASK_i (§3.5.6).
OBS-UR-TM17	This requirement is met with the possibility to generate, using VM code in Hard_VM (§3.4.2) and Soft_VM_i (§3.5.10) tasks, packets containing HK parameters sampled at whatever rate.
OBS-UR-TM18	Task HS (§3.5.7) will put into TM packets the maximum possible number of raw science frames.

4.4 Synchronization Requirements

Req. ID	Verification
OBS-UR-SY1	At each Frame Sync received from the CDMS the module ISR_1553 (§3.4.1) will activate the highest-priority task TIME (§3.5.2), 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 (§3.5.5) 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 (§3.4.2) 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 (§3.5.2) and made available as an HK parameter.

4.5 Testing and Maintainance Requirements

Req. ID	Verification	
OBS-UR-SM1	Entering the instruments Test Mode shall not require disabling of fault management (autonomy) functions.	What’s this ?
OBS-UR-SM2	The IDLE task (§3.5.13) may be used to perform DPU memory checks.	
OBS-UR-SM3	An OBS software verification facility (for PROM, EEPROM, RAM code) shall be provided on board.	What’s this ?
OBS-UR-SM4	The OBS image is stored on EEPROM	
OBS-UR-SM5	See §3.1	
OBS-UR-SM6	The Memory_LoadandDump function of task CMD_SEQ (§3.5.4) 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 (§3.5.4).
OBS-UR-SM9	Tasks HK_ASK_i (§3.5.6), Soft_VM_i (§3.5.10), Hard_VM (§3.4.2) and Tabler (§3.5.12) 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 (§3.1.1).
OBS-UR-SM11	This requirement is not met. A waiver will be requested.

4.6 Autonomy Function Requirements

Req. ID	Verification
OBS-UR-AF1	See task HK_MON (§3.5.8).
OBS-UR-AF2	Procedures are implemented as VM programs stored in tables in tabellone (§3.1.1). Task HK_MON (§3.5.8, but see figure in §3.3) can start Hard_VM with a predefined VM code to be executed.
OBS-UR-AF3	Task HK_MON (§3.5.8) will trigger the AUTONOMY task (§3.5.9) upon detection of an anomaly.
OBS-UR-AF4	See OBS-UR-AF3
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 (§3.4.2).
OBS-UR-AF6	The OBS shall provide all the event packets with a counter that permits the unambiguous identification of missing packets.
OBS-UR-AF7	The AUTONOMY task (§3.5.9) will implement a “transition edge” sensing mechanism for anomaly conditions.
OBS-UR-AF8	Control actions will be implemented as VM codes and, as such, handled by task HK_MON (§3.5.8).
OBS-UR-AF9	Task HK_MON will maintain “activity flags” for VM autonomy procedures in a configurable table in tabellone (§3.1.1)
OBS-UR-AF10	HK monitoring parameters used by task HK_MON are held in tables in tabellone (§3.1.1), as well as autonomy function VM codes; as such thay can be modified via TC.
OBS-UR-AF11	Operation/activities will be implemented as VM codes. Task Hard_VM (§3.4.2) provides opcodes to generate progress reports.
OBS-UR-AF12	Observing mode initialization is performed in VM code and, as such, completely configurable from the ground.
OBS-UR-AF13	This functionality is provided in the Command_Parser function of task CMD_SEQ (§3.5.4).
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).

4.7 Functional Requirements

Req. ID	Verification
OBS-SUR-FU1	These requirements are met by the possibility to execute these procedures either as VM codes run in Hard_VM (§3.4.2) or Soft_VM_i (§3.5.10), or as sequences of direct DRCU commands sent via TCs and managed by the Command_Parser function of task CMD_SEQ (§3.5.4).
OBS-SUR-FU2	
OBS-SUR-FU3	
OBS-SUR-FU4	
OBS-SUR-FU5	
OBS-SUR-FU6	
OBS-SUR-FU7	
OBS-SUR-FU8	
OBS-SUR-FU9	
OBS-SUR-FU10	The design of tasks LS (§3.5.5) and HS (§3.5.7) meets the requirement.
OBS-SUR-FU11	Task HS (§3.5.7) 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
OBS-SUR-GE1	Procedures implemented as VM codes. Beside the main procedure that can be run from Hard_VM (§3.4.2), up to three parallel procedures can be run on the three Soft_VM_i tasks (§3.5.10).
OBS-SUR-GE2	Requirement implemented by the Boot Software (RD8)
OBS-SUR-GE3	The task-oriented OBS architecture meets this requirement.
OBS-SUR-GE4	All instrument settings can be executed as VM code.
OBS-SUR-GE5	
OBS-SUR-GE6	
OBS-SUR-GE7	Anomalies recovery procedure are implemented as VM code and are triggered by task HK_MON (§3.5.8). While task Hard_VM (§3.4.2) is running, the HK_ASK_i task (§3.5.6) is also running.
OBS-SUR-GE8	All observing procedures are implemented as VM code.
OBS-SUR-GE9	The HS task design (§3.5.7) ensures that the OBS is fast enough to support these data rates.
OBS-SUR-GE10	
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 (§3.5.7) by using the TM_Red_info data from CMD_SEQ.
OBS-SUR-GE14	Mode transitions procedures are implemented as VM code; task Hard_VM (§3.4.2) can be run by TC from CMD_SEQ (§3.5.4).