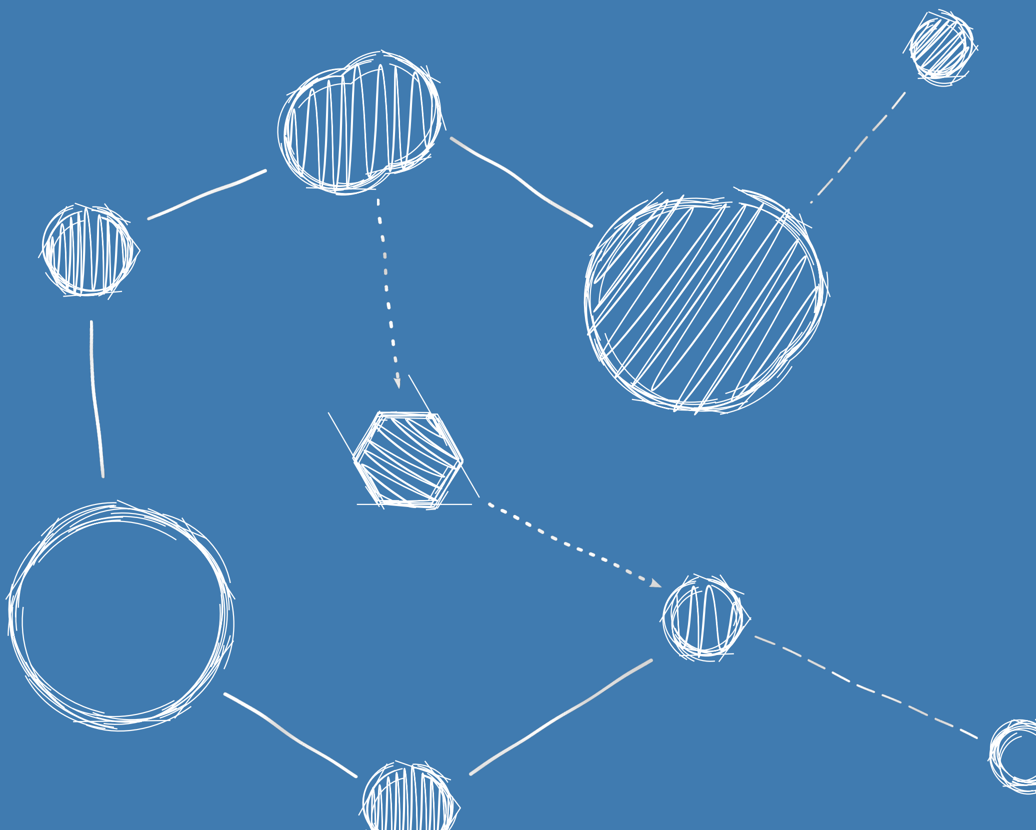
institut für astrophysik
UNIVERSITÄTSSTERNWARTE WIEN

universität wien

LEAN OS – AN OPERATING SYSTEM FOR THE SSDP

Architectural Design Document



Architectural Design Document

Reference: LEANOS-UVIE-ADD-001

Version: Issue 1.0, June 1, 2017

Prepared by: Armin Luntzer[1]

Checked by: Roland Ottensamer[1], Christian Reimers[1]

Approved by: Franz Kerschbaum[1]

[1] Department of Astrophysics, University of Vienna

Contents

List of Designs

Revision History

Revision	**Date**	**Author(s)**	**Description**
0.0	30.09.2015	AL	draft architecture created based on NGAPP
0.1	16.03.2016	AL	initial version with updated specifications from MPPBv2
0.2	02.05.2016	AL	revised after internal design review
1.0	01.06.2017	FK	document approved

1. Introduction

1.1 Purpose of the Document

This document specifies the software architecture for the operating system kernel LeanOS. This document targets developers, testers and advanced users of LeanOS. It is assumed that the reader is familiar with the user requirements and/or the software user manual.

This document follows the document structure for software design documents found in Annex F of ECSS-E-ST-40C [1].

1.2 Scope of the Software

The architectural design aims to be at a high degree of encapsulation, with a comparatively restricted number of system call interfaces to user space. The user Application Programming Interface (API) of LeanOS will orient itself to Portable Operating System Interface (POSIX) standards where applicable. Extra functionality that is typically found in supporting C libraries is part of the user space and must be implemented accordingly.

2. Applicable and Reference Documents

[1] *ECSS-E-ST-40C Space engineering - Software*. ESA Requirements and Standards Division, 2009.

[2] *Massively Parallel Processor Breadboarding Datasheet*. 2016.

[3] *The SPARC Architecture Manual Version 8*. 1991, 1992.

3. Terms, Definitions and Abbreviated Items

3.1 Acronyms

ADC	Analog to Digital Converter
API	Application Programming Interface
BSP	Board Support Package
CPU	Central Processing Unit
DAC	Digital to Analog Converter
DMA	Direct Memory Access
DSP	Digital Signal Processor
ELF	Executable and Linkable Format
FIFO	First In - First Out
GCC	GNU Compiler Collection
ILP	Instruction Level Parallelism
ISR	Interrupt Service Routine
MMU	Memory Management Unit
MPPB	Massively Parallel Processor Breadboarding system
NGAPP	Next Generation Astronomy Processing Platform
NoC	Network On Chip
POSIX	Portable Operating System Interface
PUS	Packet Utilisation Standard
RAM	Random-Access Memory
RISC	Reduced Instruction Set Computing
RMAP	Remote Memory Access Protocol
RR	Round Robin
SMP	Symmetric Multiprocessing
SoC	System On Chip
SSDP	Scalable Sensor Data Processor
VLIW	Very Long Instruction Word

3.2 Glossary

Analog to Digital Converter (ADC)

An Analog to Digital Converter is a system that converts an analog signal into a quantized digital signal. Its counterpart is the Digital to Analog Converter (DAC).

Application Programming Interface (API)

The Application Programming Interface defines how a developer can write a program that requests services from an operating system or application. APIs are implemented by function calls composed of verbs and nouns, i.e. a function to execute on an object.

Board Support Package (BSP)

A Board Support Package is the implementation of a specific interface defined by the abstract layer of an operating system that enables the latter to run on the particular hardware platform.

Central Processing Unit (CPU)

The Central Processing Unit is the electronic circuitry that interprets instructions of a computer program and performs control logic, arithmetic, and input/output operations specified by the instructions. It maintains high-level control of peripheral components, such as memory and other devices.

Digital Signal Processor (DSP)

A Digital Signal Processor is a specialised processor with its architecture targeting the operational needs of digital signal processing.

Digital to Analog Converter (DAC)

A Digital to Analog Converter is a system that converts a quantized digital signal into an analog signal. Its counterpart is the ADC.

Direct Memory Access (DMA)

Direct Memory Access is a feature of a computer system that allows hardware subsystems to access main system Random-Access Memory (Random-Access Memory (RAM)) directly, thereby bypassing the Central Processing Unit (CPU).

Executable and Linkable Format (ELF)

The Executable and Linkable Format is a common standard file format for executables, object code, shared libraries, and core dumps.

First In - First Out (FIFO)

In FIFO processing, the ”head” element of a queue is processed first. Once complete, the element is removed and the next element in line becomes the new queue head.

GNU Compiler Collection (GCC)

The GNU Compiler Collection is a compiler system produced by the GNU project. It is part of the GNU toolchain collection of programming tools.

Instruction Level Parallelism (ILP)

Instruction-level parallelism (ILP) is a measure of how many instructions in a computer program can be executed simultaneously by the CPU.

Interrupt Service Routine (ISR)

An Interrupt Service Routine is a function that handles the actions needed to service an interrupt.

LEON2

The LEON2 is a synthesisable VHDL model of a 32-bit processor compliant with the SPARC V8 architecture. It is highly configurable and particularly suitable for System On Chip (SoC) designs. Its source code is available under the GNU LGPL license

LEON3

The LEON3 is an updated version of the LEON2, changes include Symmetric Multiprocessing (Symmetric Multiprocessing (SMP)) support and a deeper instruction pipeline

LEON3-FT

The LEON3-FT is a fault-tolerant version of the LEON3. Changes to the base version include autonomous error handling, cache locking and different cache replacement strategies.

Massively Parallel Processor Breadboarding system (MPPB)

The Massively Parallel Processor Breadboarding system is a proof-of-concept design for a space-hardened, fault-tolerant multi-DSP system with various subsystems to build a powerful digital signal processing system with a high data throughput. Its distinguishing features are the Network On Chip (Network On Chip (NoC)) and the Xentium DSPs controlled by a LEON2 processor. It was developed under ESA contract 21986 by Recore Systems B.V.

Memory Management Unit (MMU)

A Memory Management Unit performs address space translation between physical and virtual memory pages and protects unprivileged access to certain memory regions.

Network On Chip (NoC)

A Network On Chip is a communication system on an integrated circuit that applies (packet based) networking to on-chip communication. It offers improvements over more conventional bus interconnects and is more scalable and power efficient in complex System On Chip (SoC) desgins.

Next Generation Astronomy Processing Platform (NGAPP)

Next Generation Astronomy Processing Platform was an evaluation of the MPPB performed in a joint effort of RUAG Space Austria and the Department of Astrophysics of the University of Vienna. The project was funded under ESA contract 40000107815/13/NL/EL/f.

Packet Utilisation Standard (PUS)

The Packet Utilisation Standard addresses the end-to-end transport of telemetry and telecommand data between user applications on the ground and applications onboard a satellite. See also ECSS-E-70-41A.

Portable Operating System Interface (POSIX)

The Portable Operating System Interface is a family of standards specified by the IEEE Computer Society for maintaining compatibility between operating systems.

Random-Access Memory (RAM)

Random-Access Memory is a type of memory where each memory cell may be accessed directly via their memory addresses.

Reduced Instruction Set Computing (RISC)

RISC is a CPU design strategy that intends to improve performance by combining a simplified instruction set with a microprocessor architecture that is capable of executing an instruction in a smaller number of clock cycles.

Remote Memory Access Protocol (RMAP)

The Remote Memory Access Protocol is a form of SpaceWire communication that transparently communicates writes to memory mapped regions between different hardware devices.

Round Robin (RR)

Round Robin is a scheduling algorithm where time slices are assigned in equal poritions and in circular order. In the context of threads, priorities are usually only used to control re-scheduling order when a mutex is accessed by a thread.

Scalable Sensor Data Processor (SSDP)

The Scalable Sensor Data Processor (SSDP) is a next generation on-board data processing mixed-signal ASIC, envisaged to be used in future scientific payloads requiring high-performance on-board processing capabilities. It is built opon a heterogeneous multicore architecture, combining two Xentium DSP cores with a general-purpose LEON3-FT control processor in a Network On Chip (NoC).

SpaceWire

SpaceWire is a spacecraft communication network based in part on the IEEE 1355 standard of communications.

SPARC

SPARC (”scalable processor architecture”) is a Reduced Instruction Set Computing (RISC) instruction set architecture developed by Sun Microsystems in the 1980s. The distinct feature of SPARC processors is the high number of Central Processing Unit (CPU) registers that are accessed similarly to stack variables via “sliding windows”.

Symmetric Multiprocessing (SMP)

Symmetric Multiprocessing denotes computer architectures, where two or more identical processors are connected to the same periphery and are controlled by the same operating system instance.

System On Chip (SoC)

A System On Chip is an integrated circuit that combines all components of a computer or other electronic system into a single chip.

Very Long Instruction Word (VLIW)

Very Long Instruction Word is a processor architecture design concept that exploits Instruction Level Parallelism (ILP). This approach allows higher performance at a smaller silicone footprint compared to serialised instruction processors, as no instruction re-ordering logic to exploit superscalar capabilities of the processor must be integrated on the chip, but requires either code to be tuned manually or a very sophisticated compiler to exploit the full potential of the processor.

Xentium

The Xentium is a high performance Very Long Instruction Word (VLIW) DSP core. It operates 10 parallel execution slots supporting 32/40 bit scalar and two 16-bit element vector operations.

4. Software Design Overview

4.1 Software Static Architecture

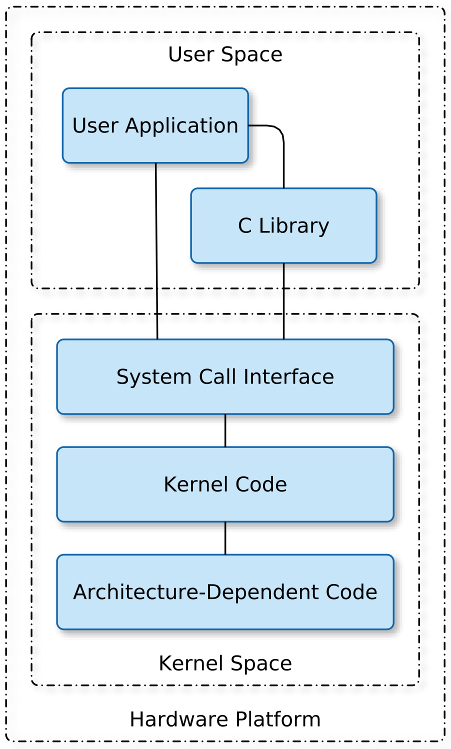


Figure 4.1: The fundamental architecture of LeanOS

The fundamental architecture of LeanOS is shown in Figure 4.1. At the top is the user space, where user applications are set up and executed. The C library is part of the user space. It provides the system call interface that connects the user to kernel space functionality and memory address space. Below the user space is the kernel space. Here, the tasks of the operating system kernel are executed.

Kernel space can be further divided into three gross layers. At the top is the system call interface, which implements any I/O functionality to the user space. Below the system call interface is the architecture-independent kernel code. While LeanOS is not designed with a high degree of portability in mind, it is nonetheless sensible not to mix hardware dependent code into layers that run on abstract functional concepts. On the lowest level sits the architecture-dependent code, which forms what is typically called a BSP. This code serves as platform-specific glue to the underlying hardware.

4.2 Software Dynamic Architecture

LeanOS itself has no real time requirements. It does offer real time support for threads in both kernel and user space, these are however specific to the implementation of user space code or drivers/modules.

4.3 Software Behaviour

The behaviour of LeanOS is configuration-dependent. Usually, the kernel will configure itself and the hardware, followed by user space initialisation.

Without user space, all actions by the drivers or other subcomponents will be their default action in base configuration. To give an example, interface drivers will usually ignore their inputs and drop all received.

4.4 Interfaces Context

The internal and external software interfaces of LeanOS are described as part of its source code in Doxygen markup, from which a document can be generated.

4.5 Long Lifetime Software

LeanOS is designed with the SSDP as a target platform. To allow re-use and adaptation to new hardware, the software components of LeanOS are designed to be modular. Unless neeeded for particular drivers, all hardware access is abstracted into a separate layer as much as possible, so improved portability is ensured.

4.6 Memory and CPU Budget

LeanOS is designed to work within the constraints of the SSDP hardware specification. Care is taken to minimise overheads, but run time needs of the user ultimately define memory and CPU usage of the operating system.

4.7 Design Standards, Conventions and Procedures

The high level design of software functionality is done with the help of *yEd*, which is a general-purpose diagramming program supporting the creation of UML, flowcharts and entitiy relationship diagrams. LeanOS is written primarily in C, hence certain restrictions apply in the use of UML, which is mainly used of object oriented programming languages such as C++.

The software architecture is described on a component level and the interaction or communication between components. The detailed internal functional design of a component is left to the implementation and is described as part of the version dependent issue of the source code. This is done to ensure that a certain amount of flexibility is present in the implementation and changes that do not affect the fundamental design architecture of LeanOS can be applied quickly and efficiently, since it is foremost a tool to the user with the purpose of creating a run-time, not a product implementing a well constrained task.

The Linux kernel coding style is applied to all C code in LeanOS. Its use is enforced by the *checkpatch.pl* utility found in the Linux kernel source tree.

No external software packages are reused in the implementation of LeanOS.

5. Software Design

5.1 General

The purpose of LeanOS is to create an operating system that is easy to use and on point with regard to the features of the target platform. Still, it is designed to be flexible enough that the operating system kernel may be used with other (LEON3) platforms. One of its distinguishing features is its focus on processing with the Xentium DSPs.

When dealing with the particular features of the SSDP or the MPPB [2], the term *kernel* may be used in different contexts. With regard to processing and the Xentium DSPs, a *processing kernel* is a small functional program that runs on the Xentium, usually performing a single task. With regard to the general purpose processor and operating system features, the term *kernel* refers to the operating system core program.

Functional requirements are always referenced to their design components, others only as needed or for clarification. This is reflected in the traceability matrix.

5.2 Overall Architecture

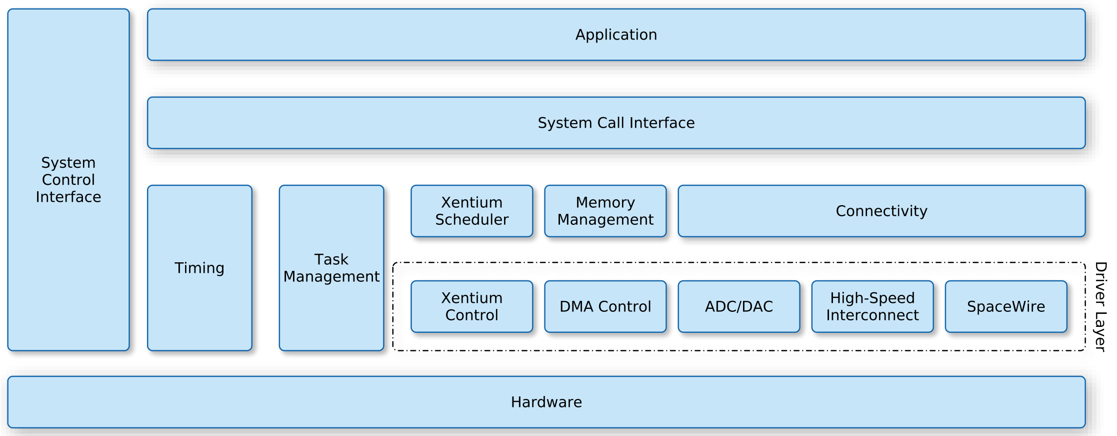


Figure 5.1: The architectural model of LeanOS. Here, the ”hardware” layer represents both the hardware and the hardware abstraction layer of the software.

In LeanOS, the SSDP hardware is accessed in multiple layers of abstraction (see Figure 5.1). Typical CPU tasks such as thread/task management and timer operation are used as part of the operating system kernel and are also accessible by user applications via a system call interface.

Other functional hardware components of the SSDP such as the NoC DMA have their own driver modules. These are in turn used by the Xentium scheduler and other higher level modules in the operating system. Configuration of and access to the latter from user space is done via a system call interface. The system control interface serves as an intermediate between all layers of the operating system, where system or module states and hardware modes or usage statistics may be exported by individual components for external (user) access.

5.3 Software Components Design - General

D-GEN-0001	**Identifier**	**Function**
	Boot	This is the hardware entry point of LeanOS. The boot procedure sets up and configures the LEON3-FT processor for operation, initialises a minimum set of needed hardware devices and memories as well as the initial system stack.
Purpose	R-GEN-0006, R-FUN-0007, R-FUN-0804	

D-GEN-0002	**Identifier**	**Function**
	User Space	After the kernel has finished initialising, it calls a function *kernel_init()* that executes an *init()* function configured by the user. This is the run-time adaptation point from which user space is started. From there, the user can start processes and reconfigure or load kernel options and modules via the system call or sysctl interfaces.
Purpose	R-FUN-0804	

D-GEN-0003	**Identifier**	**Function**
	Interrupts, Traps	In order to operate a SPARC v8 CPU properly, a trap table must be configured, in particular to handle register window under and overflow traps if used with regular GCC code generation for the target platform. LeanOS configures a callback system for interrupt traps that can later be used to install custom handlers, for example as part of driver modules. Interrupt statistics are exported via the system control interface.

Purpose	R-GEN-0006, R-GEN-0009, R-GEN-0018, R-GEN-0010, R-FUN-0023
Comment	The trap table and its function are described in [3] in detail.

D-GEN-0004	**Identifier**	**Function**
	Timers	Timing functionality is a core element of an operating system by scheduling periodic and non-periodic kernel wakeup events that subsequently control the system's exectuion. The operating system kernel maintains these timers to measure time or schedule kernel wakeups. In addition to the CPU bound timers, the real-time clock of the SSDP is supported.

Purpose	R-FUN-0011, R-FUN-0016, R-FUN-0017

D-GEN-0005	**Identifier**	**Function**
	Kernel Mode	The operating system protects access to privileged registers by disabling supervisor mode when switching to user space. On the SPARCv8, traps enable supervisor mode, so the privileged mode is automatically entered when the kernel is executing after a trap/interrupt or a system call from user space, which is also implemented via the trap table. Kernel and unmapped memory access from user space is protected by the MMU.

Purpose	R-FUN-0803, R-FUN-0008

D-GEN-0006	**Identifier**	**Function**
	MMU	The kernel uses the MMU to map pages of the system memory into a virtual address space for most of its own processes and for user space. If needed, physical memory is directly accessible by driver components or mapped accordingly.
Purpose	R-FUN-0803, R-FUN-0008	
Comment	The proper handling of address space translation is particularly relevant for use with the NoC DMA driver and Xentium scheduler.	

D-GEN-0007	**Identifier**	**Function**
	Thread Support	This component supports the definition and creation of tasks. Task priorities and scheduling deadlines are supported depending on the selected mode. Synchronisation and execution control between threads is provided via semaphores and mutexes.
Purpose	R-FUN-0012, R-FUN-0013, R-FUN-0019	

D-GEN-0008	Identifier	Function
	Thread Schedul-ing	Threads are scheduled by the kernel according to their run state, their priorities and optionally their deadline. The state of mutexes and semaphores is used to temporarily re-order priorities if needed, so lower priority threads do not block higher priority threads and vice versa when locks are employed. Unless strict FIFO mode is configured, threads of all scheduling priorities are regularly assigned an execution time slice. The length of this time slice decreases with priority. If Round Robin (RR) scheduling is to be used, time slices are set to be equal for all threads regardless of priority. The latter is then only used to control re-ordering of the thread schedule to more efficiently solve mutex lock situations. The user is in any case responsible to assign different priorities to threads where deadlock situations might occur that cannot be resolved by the scheduler (e.g. with identical priorities). Access to mutexes and semaphores result in random scheduling events if the lock is actively held or waited for by any other threads. In tickless mode, regular scheduling occurs when a thread is preempted at the end of its timeslice. If the kernel is configured to tick periodically, scheduling events will occur at a configurable integer fraction of that rate. A special type of real-time thread is supported, which, with the exception of an ISR may also preempt the operating system kernel if its release condition occurs and may run to its deadline without being preempted by other threads. This functionality is reserved for highly timing-critical tasks and is reserved to functionality and modules executed in kernel space. If a multi-core processor is present in the system, the kernel can schedule tasks to run on more than one CPU if so configured.
Purpose	R-FUN-0014, G-FUN-0015, R-FUN-0016, R-FUN-0017, R-FUN-0019, R-FUN-0020, R-GEN-0028	

D-GEN-0009	Identifier	Function
	Message Queues	Message queues are a facility for tasks/processes to communicate arbitrary data to each other. A named queue is created by one thread and opened by at least one other thread. The threads can then send and receive messages of arbitrary length. If a thread wants to actively wait for a message, it can request to be notified and is subsequently woken by the scheduler once a message arrives.

Purpose	R-FUN-0021
Comment	The implementation follows the POSIX message queue API definition.

D-GEN-0010	Identifier	Function
	Loadable Modules	Kernel modules can be loaded via the system call interface. The module is supplied as an ELF binary that holds special sections that are executed by the kernel as their *init()* or *exit()* functions whenever they are loaded or unloaded. The *init* function of a module is typically used to configure its functionality within the kernel, e.g. register a thread, gain access to a hardware device or register an interrupt callback. If the module is no longer needed or explicitly unloaded by the user, the *exit* function is used to de-register and clean up functionality used by the module. Start-up arguments can be supplied to the module during loading. At run time, the module may publish its internal settings via the kernel's system control interface.

Purpose	R-FUN-0022

D-GEN-0011	Identifier	Function
	System Control Interface	The system control interface is a generic parseable interface that is accessible from user and kernel space. It is modeled on a logical tree that holds nodes representing entries of kernel components and can be viewed as a file/directory structure. The input/output of each entry and how it may be parsed is defined by its creator and may be used for run-time configuration or statistics keeping. Data may be read and written to the nodes and are exchanged via character-based text. Special nodes may be created that allow exchange of raw binary data, for example internal memory dumps or configuration data. The entries in a node are functions that are part of a module or system component. The component registers these functions along with a name and a positional branch name within the logical tree. Components may also define new branches where their interface is positioned. If the node is read, the system control interface executes the associated output function of the component and returns a character buffer to the reader. If the node is written, the writer-supplied buffer is passed to the component's internal associated input function and parsed by the latter. Success or failure of the operation depends on the correct use of the individual make-up of the character buffers.
Purpose	R-FUN-0023, R-FUN-0024	

D-GEN-0012	Identifier	Function
	SpaceWire Driver	The SpaceWire devices of the SSDP are accessible from user space via a file descriptor that can be read or written atomically. If a SpaceWire device is only used in the Xentium processing network, it may be configured by the driver for direct DMA to the input buffer stage. Similarly, it may be used with DMA for the output buffer stage. If the RMAP feature is to be used, the configured physical memory block is mapped into a virtual memory space.

Purpose | G-FUN-0025, R-FUN-0026, R-FUN-0027

D-GEN-0013	Identifier	Function
	ADC/DAC Driver	The ADC/DAC devices of the SSDP are accessible from user space via a file descriptor that can be read or written atomically. If a ADC device is only used in the Xentium processing network, it may be configured by the driver for direct DMA to the input buffer stage. Similarly, the DAC may be used with DMA for the output of the processing chain.

Purpose | G-FUN-0025

D-GEN-0014	Identifier	Function
	Xentium Driver	The Xentium driver and scheduler loads binary images of functional DSP program kernels via a system call interface. Each image is assigned a signature that identifies the type of function of the DSP kernel and a mask that control the number of instances and Xentium DSPs it may run on at any time. To each DSP kernel image, an input buffer is assigned, which stores references to metadata packets. Buffer fill level thresholds are defined that control the dynamic processing priority during run-time. The first threshold defines the *minimum* fill level above which the processing kernel may be scheduled. This is relevant in situations where more than input packets are needed to perform a task. The second threshold is the *low* threshold. As long as the buffer fill level is below this threshold, it is only scheduled with the lowest priority. If the level is above the *low* threshold, the buffer contents are scheduled with *medium* priority. The third threshold is the *high* threshold. If the buffer level exceeds this threshold, it is scheduled for processing with the highest priority.

Purpose	R-FUN-0026
Comment	The minimum input threshold is useful in situations where the processing kernel must combine individual data segments, for example when stacking individual frames that passed through preprocessing stages earlier.

D-GEN-0015	Identifier	Function
	Xentium Queues	The scheduling queues are ordered by buffer fill level. The queue is updated as packets move between metadata buffers. The update mechanism is part of the metadata buffers and triggers a message to the driver whenever a level threshold is either over- or under-run and the most recent critical (*high*) buffer is moved to the head of the queue. If all buffers are above the *high* threshold, they are processed in an Round Robin (RR) pattern. The driver maintains a separate *bonded* queue for each DSP that holds program kernels that are allowed to run only on a particular DSP and one shared queue that holds all other kernels. The scheduling priority of equal-level buffer states of bonded queues is higher than the shared queue, i.e. buffers above the *high* threshold in a bonded queue are processed first, followed by the *high* buffers in the shared queue, followed by the *medium* level buffers in the bonded queue etc.
Purpose	R-FUN-0027	

D-GEN-0016	Identifier	Function
	Xentium Sched-uler	If a program kernel's buffer is above the *high* threshold, duplicates of the program kernel may be scheduled to run in parallel on other DSPs according to the kernel's control mask. If more than two Xentiums are available in the system, the distribution scheduling algorithm ensures that all available DSPs are assigned a program kernel above the *high* threshold with respect to their queues before scheduling duplicate instances. If DSPs are available, the head of the queue is duplicated according to its control mask, followed by the next item and so on, as long as free Xentiums are available and pending *high* threshold buffers exist in the queue. The driver also maintains a special type of metadata buffer for input and output nodes that connect the Xentium processing chain to external data links or mass storage. Of these, the input buffers are processed with the highest absolute priority relative to their fill status. This is done to ensure that if a data flow stall occurs, it does so as far back in the processing chain as possible and data input is maintained until all buffers are filled to their limit.

Purpose	R-FUN-0026, R-FUN-0027
Comment	Typically, the input buffer is the input for a special DSP kernel that processes and interprets the incoming data stream (of e.g. PUS packets) and prepares and/or distributes metadata packets to the inputs of the processing kernels according to the desired application.

D-GEN-0017	Identifier	Function
	Xentium Data Buffers	The metadata buffers track metadata packets as they move through the processing network. With the exception of the input buffers to the network, they are the input of one and the output of an arbitrary number of Xentium program kernels. Each item in a buffer corresponds to one metadata packet descriptor. A metadata packet holds a pointer to data buffers of arbitrary size and a field describing a route through the processing network of available kernels by their signature identifier, i.e. it defines its own processing chain. Each entry holds an associated pointer to an optional argument field, which may hold parameters for the processing kernel. Program kernels may collect one or more of these packets at their input depending on their processing functionality and either operate on the contents of the packet or generate new packets from one or more input packets. As the packet moves through the kernels, items on the processing routing stack are moved to a history field and they are output into a meta-buffer according to their pending routing node.
Purpose	R-FUN-0027	

5.4 Software Components Design - Aspects of each Component

The detailed design of the individual components is documented as part of their source code in Doxygen markup. This ensures continued flexibility in development and allow for effortless maintainance of a fully update state between the implementation and its detailed documentation.

5.5 Internal Interface Design

The internal interfaces of the operating system are described as part of the source code in Doxygen markup and may be generated from there.

5.6 Requirements to Design Components Traceability

Functional requirements are always reference to their design components, others only as needed or for clarification. This is reflected in the traceability matrix.

	D-GEN-0001	D-GEN-0002	D-GEN-0003	D-GEN-0004	D-GEN-0005	D-GEN-0006	D-GEN-0007	D-GEN-0008	D-GEN-0009	D-GEN-0010	D-GEN-0011	D-GEN-0012	D-GEN-0013	D-GEN-0014	D-GEN-0015	D-GEN-0016	D-GEN-0017
R-GEN-0001																	
R-GEN-0002																	
R-GEN-0003																	
R-GEN-0004																	
R-GEN-0006	■		■														
R-GEN-0601																	
R-GEN-0602																	
R-FUN-0007	■																
R-FUN-0803					■	■											
R-FUN-0008					■	■											
R-FUN-0011				■													
R-FUN-0012							■										
R-FUN-0013							■										
R-FUN-0014								■									
G-FUN-0015								■									
R-FUN-0016				■				■									
R-FUN-0017				■				■									
R-FUN-0019							■	■									
R-FUN-0020								■									
R-FUN-0021									■								
R-FUN-0022										■							
R-FUN-0804	■	■															
R-FUN-0023			■								■						
R-FUN-0024											■						
G-FUN-0025												■	■				
R-FUN-0026												■		■		■	
R-FUN-0027												■			■	■	■
R-GEN-0009			■														

	D-GEN-0001	D-GEN-0002	D-GEN-0003	D-GEN-0004	D-GEN-0005	D-GEN-0006	D-GEN-0007	D-GEN-0008	D-GEN-0009	D-GEN-0010	D-GEN-0011	D-GEN-0012	D-GEN-0013	D-GEN-0014	D-GEN-0015	D-GEN-0016	D-GEN-0017
R-GEN-0018			■														
R-GEN-0010			■														
R-GEN-0028								■									
R-GEN-2001																	
R-GEN-0101																	
R-GEN-0801																	
R-GEN-0802																	
R-GEN-0805																	
G-GEN-0201																	
R-GEN-0301																	
R-GEN-0401																	
R-GEN-0402																	
R-GEN-0403																	
R-GEN-0005																	
R-GEN-1001																	
R-GEN-1002																	