UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

MEMORY PROTECTION AND QUALIFICATION OF REAL-TIME OPERATING SYSTEMS FOR SPACE APPLICATIONS

José Joaquim Pinto de Sousa

MESTRADO EM ENGENHARIA INFORMÁTICA
Arquitectura, Sistemas e Redes de Computadores

2009

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

MEMORY PROTECTION AND QUALIFICATION OF REAL-TIME OPERATING SYSTEMS FOR SPACE APPLICATIONS

José Joaquim Pinto de Sousa

PROJECTO

Trabalho orientado pelo Prof. Doutor José Manuel de Sousa de Matos Rufino

e co-orientado pelo Engº Helder Filipe Monteiro da Silva

MESTRADO EM ENGENHARIA INFORMÁTICA

Arquitectura, Sistemas e Redes de Computadores

2009

Declaration

José Joaquim Pinto de Sousa, student nº26666 of Faculdade de Ciências Universidadede Lisboa, declares to give the copyrights of its Report of Projecto em Engenharia Informática, named “Memory Protection and Qualification of real-time perating systems for Space Applications”, accomplished during the year 2008/2009 to Faculdade de Ciências Universidade de Lisboa for effects of archive and consultation on its libraries and publication on electronic format on the Internet.

José Joaquim Pinto de Sousa aluno nº26666 da Faculdade de Ciências Universidade de Lisboa, declara ceder os seus direitos de cópia sobre o seu Relatório de Projecto em Engenharia Informática, intitulado “Memory Protection and Qualification of real-time perating systems for Space Applications”, realizado no ano lectivo de 2008/2009 à Faculdade de Ciências Universidade de Lisboa para o efeito de arquivo e consulta nas suas bibliotecas e publicação do mesmo em formato electrónico na Internet.

Lisboa, 29 de Setembro de 2009

Resumo

O RTEMS (**R**eal **T**ime **E**xecutive for **M**ultiprocessor **S**ystems) é um sistema operativo de tempo real (RTOS) que está a ser activamente desenvolvido e utilizado em aplicações de tempo real. Este facto motivou o desenvolvimento de um centro de investigação e desenvolvimento para o RTEMS, o Edisoft RTEMS Centre, com o intuito de dar suporte à comunidade espacial europeia.

As primeiras actividades do RTEMS Centre consistiram na criação de ferramentas de suporte e auxílio à configuração e compilação do sistema operativo RTEMS. E ainda na criação de uma ferramenta que verifica o comportamento de uma aplicação em tempo de execução. Numa fase mais avançada do RTEMS Centre foi iniciado o projecto RTEMS Improvement, o qual visa disponibilizar uma versão ajustada do sistema operativo RTEMS (4.8.0) com o objectivo de ajudar e facilitar o processo da qualificação de aplicações baseadas no RTEMS para as missões espaciais. A versão produzida auxilia o processo de qualificação e inclui uma bateria de testes que rastreia os requisitos de software, estes testes cobrem o código fonte segundo a norma SW-DAL (SoftWare Development Assurence Level) nível B, o qual obriga a que todas as linhas do código fonte de uma aplicação sejam executadas e que todos os blocos de decisão tenham sido também executados.

A qualificação de software que necessita de seguir as normas definidas no Galileo Software standards SW-DAL B (**GSWS** [RD1]) é um processo longo e complexo. O processo completo de qualificação das aplicações espaciais só pode ser concluído conjugando o sistema com os componentes de software da aplicação e com o hardware onde a mesma se executa. Uma vez que o hardware não se encontra disponível, o principal objectivo consiste em facilitar o processo de qualificação de aplicações que utilizem RTEMS fornecendo: a RTEMS tailored version com todas as modificações necessárias para corrigir os bugs detectados; a RTEMS Test Suite, que testa o RTEMS Tailored; e a documentação associada (documento de requisitos, documento de desenho detalhado, documento de configuração, manual do utilizador, etc).

Numa fase mais avançada do projecto do RTEMS Improvement, e quando todas as etapas anteriores estiverem concluídas será desenvolvido um módulo de gestão de memória para o RTEMS para a classe de processadores LEON3.

Palavras-chave: RTEMS Improvement, RTEMS, Qualificação de aplicações para o espaço, Gestão de Memória, Sistemas Embebidos e de Tempo-Real, Missão Crítica, Qualificação .

Abstract

The **R**eal **T**ime **E**xecutive for **M**ultiprocessor **S**ystems (RTEMS) is a **R**eal **T**ime **O**perating **S**ystem (RTOS) that is being actively developed and used in hard real time applications development. This fact has motivated the development of a RTEMS Centre, the Edisoft RTEMS Centre, which investigates on RTEMS and be able to give help and support to the European space community.

The first RTEMS Centre activities were the development of support tools to help in the configuration and build RTEMS. Another tool has been developed; this tool verifies the RTEMS applications behaviour at execution time. In a later phase, the RTEMS Centre has started the RTEMS Improvement project that aims to create a RTEMS (4.8.0) Tailored version that will help in the facilitation of qualification process for the RTEMS applications to the space missions. The produced material that helps in the facilitation of qualification process has included the design of a new test suite to cover the requirements for software and the source code using the standard SW-DAL (SoftWare Development Assurance Level) level B with 100% statement coverage, all the lines of code has to be executed, and 100% decision coverage, all the decision blocks has been executed.

The qualification of software that needs to following the Galileo Software Standards SW-DAL B (**GSWS** [RD1]) is a long and complex process. The complete qualification process implies the qualification of both software and hardware platform where the software application runs. Since the RTEMS Improvement project does not have the hardware support needed, the main goal is to facilitate the qualification process of the applications that runs with RTEMS, through: a RTEMS Tailored version, with the necessary modifications required to correct the detected bugs; a RTEMS Test Suite, to test the RTEMS Tailored; and the associated documentation (requirements document, design document, configuration document, user manual, etc).

In a later phase of the RTEMS Improvement project when all the previous tasks for the RTEMS Tailored version add been concluded it will be developed a RTEMS Memory Management module for the LEON3 class processors.

Keywords: RTEMS Improvement, RTEMS, Qualification of Applications to Space, Memory management, Real-Time embedded systems, Critical Mission, Qualification.

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

Term	Description
ADD	Architectural Design Document
API	Application Programming Interface
BSP	Board Support Package
COTS	Commercial Off The Shelf
DAL	Development Assurance Level
DDD	Detailed Design Document
DDF	Design Definition File
DJF	Design Justification File
DRD	Document Requirements Description
ESA	European Space Agency
GCC	GNU Compiler Collection
GCOV	GCC Coverage Tool
GDB	GNU Debugger
GNU	GNU's Not Unix
GPL	General Public License
HW	HardWare
IPR	Intellectual Property Rights
ISVV	Independent Software Verification and Validation
LGPL	Lesser GPL
MMU	Memory Management Unit
OAR	On-Line Applications Research
OS	Operative System
RAMS	Reliability, Availability, Maintainability, Safety
RD	Reference Document
RTEMS	Real Time Executive for Multiprocessor Systems
RTOS	Real Time OS
SCF	Software Configuration File
SDD	Software Design Document
SDP	Software Development Plan
SIP	Software Integration Test Plan
SOM	Software Operation Manual
SOW	Statement Of Work
SPA	Software Product Assurance

SPAP	Software Product Assurance Plan
SPAR	Software Product Assurance Report
SPARC	Scalable Processor ARChitecture
SRD	Software Requirements Document
SUP	Software Unit test-Plan
SVR	Software Verification Report
SVVP	Software Verification and Validation Plan
SW	SoftWare
UML	Unified Modelling Language
VTS	Software Validation Testing Specification

Chapter 1

Introduction

This document describes the work developed during one year in the RTEMS (Real-Time Executive for Multiprocessor Systems)Improvement project on the Edisoft RTEMS CENTRE, in collaboration with ESA[1], to produce a RTEMS Tailored version optimized for the SPARC V8 compliant processors, namely the ERC32, LEON2 and LEON3 processors, in order to facilitate the qualification of RTEMS based space applications.

1.1 Motivation

RTEMS, the Real Time Executive for Multiprocessor Systems, is a free open source Real Time Operating System (RTOS) designed for deeply embedded systems that aim to be competitive with closed source and commercial products. It was developed to support applications with strict timeliness requirements, making possible for the user to develop hard real time systems. RTEMS also offers several features, such as multitasking capabilities, inter-task communication and synchronization, support for several network protocols, different platform support and it is being actively developed.

The RTEMS CENTRE was a project under the ESA-Portugal Task Force aiming to develop a support and maintenance centre for the RTEMS operating system. This project was also the first shoot to develop a European RTEMS CENTRE for the European Space Community. In an initial phase the Edisoft RTEMS CENTRE team developed technical expertise and some support tools for the RTEMS. In a second phase the RTEMS improvement project was started.

The RTEMS Improvement is a project that aims to produce a tailored version of the RTEMS Operating System, based on version 4.8.0, that facilitates the qualification

[1] ESA-European Space Agency

of software space applications produced over the RTEMS Tailored version. To achieve this goal a stripped version of the RTEMS kernel has to be produced. This stripped version includes only a minimal set of functionalities and a minimal set of managers. The process taken to choose this minimal set of functionalities and managers is explained in detail in chapter 2 and chapter 3 of this document.

Another goal for the RTEMS Improvement project is to produce a RTEMS memory management module for SPARC V8, LEON 3, processors. This goal is not the primary goal of the RTEMS Improvement project, and it will be developed only upon completion of the previous tasks.

1.2 Objectives

The main goal of the work described in this document is to produce a RTEMS Tailored version with a minimal set of functionalities and managers that satisfy the software requirements for space applications. This set of functionalities and managers will be chosen taking in account the managers and functionalities most used in space software applications. The aim is to facilitate the RTEMS qualification for future space missions. The full qualification implies to qualify both the software application and the hardware where that application will work. Each one of these missions has its own software and hardware and particular configuration. By using the outputs produced in the RTEMS Improvement project, it shall be possible for the European Space Community to qualify applications using the tailored version of RTEMS.

Another important goal of the RTEMS Improvement project is to provide RTEMS with Memory Management and protection support for LEON processors. The RTEMS version 4.8.0, which is the version adopted for the development of the RTEMS Improvement project do not offer a Memory Management Module.

The usage of a Memory Management module is important in memory protection since without usage of such module, memory access violations, i.e. the access to memory that do not belongs to the running task, are undetectable. These unwanted accesses need to be avoided because one task could change data in the memory of another task. The gravity of this unwanted access depends on the how important is the changed data and how important is the task that has his data changed. The lack of memory protection leads the applications to be more susceptible to errors.

1.3 Institutional integration

Established in 1988, **EDISOFT, S. A**. is a specialised Portuguese company that offers technologically advanced software solutions and highly qualified IT consulting services to the design, development and integration of critical real-time command, control, communications, computer and intelligence systems, being thus a reference in the national defence industry nucleus.

EDISOFT has a solid technical and technological expertise in air traffic control systems, networking, information security and the integration of strategic collective security systems, dedicated to the professional emergency and civil protection sector.

EDISOFT also holds a profound knowledge in the development of integrated business solutions, in the banking, civil service, telecommunications and logistic areas, and in the definition of decisional solutions based on geographical information systems, as well as a broad experience in international research and development projects in the Space field of expertise.

Due to the friendly environment existent on Edisoft my integration was easy and a positive experience. I have joined to a team that have a large and deeply knowledge on the embedded systems area and also have a large experience to work with ESA on the development of space applications.

EDISOFT also promotes that their collaborators acquire new or more knowledge in the area that they are working. With this in mind the company very often gives formation to their collaborators.

In order to give RTEMS Improvement team members knowledge about work with FPGA[2], EDISOFT has given formation to them on VHDL[3] for Xilinx FPGA boards. I have participated in that formation to acquire knowledge to design components in VHDL. In the formation it has been used the Xilinx[4] ISE® *WebPACK*™[5] development tool and the Xilinx Spartan 3A FPGA development boards. This formation has taken an entire week and has started from the basics, the design of one VHDL simple component and has evolved to the design of a more complex and structured component that integrates all the previously designed components. This formation was given in a strong practical environment and the team could test the developed systems on both Xilinx ISE simulator and on Xilinx Spartan 3 development FPGA boards.

[2] FPGA – **F**ield **P**rogrammable **G**ate **A**rrays

[3] VHDL – **V**HSIC **H**ardware **D**escription **L**anguage, VHSIC stands for **V**ery-**H**igh-**S**peed **I**ntegrated **C**ircuit

[4] Xilinx – http://www.xilinx.com/

[5] ISE® *WebPACK*™ is a free and fully featured front-to-back FPGA design tool

1.4 Publications

As a member of the RTEMS Improvement team the following articles have been published:

- DASIA 2009 - RTEMS CENTRE – SUPPORT AND MAINTENANCE CENTRE TO RTEMS OPERATING SYSTEM, Silva, H.; Constantino, A.; Freitas, D.; Coutinho, M.; Faustino, S.; Mota, M.; Colaço, P.; Sousa, J.; Dias, L.; Damjanovic, B.; Zulianello, M. And Rufino, J.
- INFORUM 2009 - RTEMS Improvement – Space Qualification of RTEMS Executive, Silva, H.; Sousa, J.; Freitas, D.; Faustino, S.; Constantino, A. and Coutinho, M.

1.5 Document organization

This document is organized as follows:

- Chapter 2 – will give a RTEMS operating system overview and a brief explanation of the RTEMS managers.
- Chapter 3 – provides a justification for the inclusion and the exclusion of some managers in the RTEMS Tailored version and also provides a detailed description of the tailoring plan for RTEMS version 4.8.0, and a justification for the dead code removal.
- Chapter 4 – presents a brief explanation of the Galileo SW-DAL[6] B software requirements. It also explains the steps taken in order to achieve RTEMS facilitation of qualification and the policy adopted for the RTEMS Improvement test suite in all of its steps: design, implementation and execution.
- Chapter 5 – will give an overview over the Memory Management Module.
- Chapter 6 – draws some conclusions about the work performed and presents a few directions for future work.

[6] SoftWare Development Assurance Levels

Chapter 2

RTEMS Overview

The Real Time Executive for Multiprocessor Systems (RTEMS) is a free open source Real Time Operating System (RTOS) designed for deeply embedded systems that aim to be competitive with closed source and commercial products. It was developed to support applications with strict timeliness requirements, making it possible for the user to develop hard real time systems on top of it. RTEMS is maintained by the On-Line Research Corporation (OAR) and it offers several features such as network protocols, file systems support, debug support via gdb[7] and other debugging tools and new features are currently being actively developed by RTEMS community. RTEMS also supports several CPUs from different architectures which include the SPARC, i386, PowerPC, ARM, Motorola, MIPS and Hitachi processor families. The first version of RTEMS was released in 1988.

The basic RTEMS kernel features supported include multitasking, different scheduling algorithms such as Event Driven Priority Based Preemptive and Rate Monotonic, inter-task synchronization and inter-task communication, interrupt management and dynamic memory management. The kernel is highly configurable as it allows selecting which modules to use before it is compiled, avoiding unnecessary initialization delays and memory usage on the final target image. As for networking capabilities it uses a customized high performance FreeBSD TCP/IP stack and supports different protocols like UDP, TCP, ICMP, RARP and DHCP; additionally there are also servers implemented for FTP and HTTP protocols. Supported file systems include the IMFS (In Memory File System), FAT12, FAT16, FAT32 and clients for TFTP and NFS. Despite lacking of an Integrated Development Environment (IDE) or polished monitoring/trace tools, there is support for remote debugging using GDB via the Ethernet or serial ports.

[7] Gdb is the gnu debugger tool, it is a free open source debugger (http://directory.fsf.org/project/gdb)

Applications can be developed in C, C++ and Ada95 (although the support is limited for Ada95) using different APIs such as Ada, POSIX, µITRON and RTEMS' own API set (based on the RTEID/ORKID standard); all of them use RTEMS internal functions, except for the Ada API which uses RTEMS' API as an abstraction layer. The RTEID (Real Time Executive Interface Definition) was developed by Motorola with technical input from the company that developed the pSOS RTOS. The VITA (VMEbus Industry Trade Association) adopted RTEID as a draft for their interface, ORKID (Open Real Time Kernel Interface Definition). Posterior efforts in RTOS interface standardization resulted in the POSIX (Portable Operating System Interface) 1003.1b standard, which included real time extensions. The µITRON (Micro Industrial The Real time Operating system Nucleus) is an interface that aims to standardize RTOS specifications for embedded systems.

RTEMS can be characterized by three layers: hardware support, kernel and APIs. The user then develops his application by using the available APIs. This layered architecture is depicted in the diagram of Figure 1.

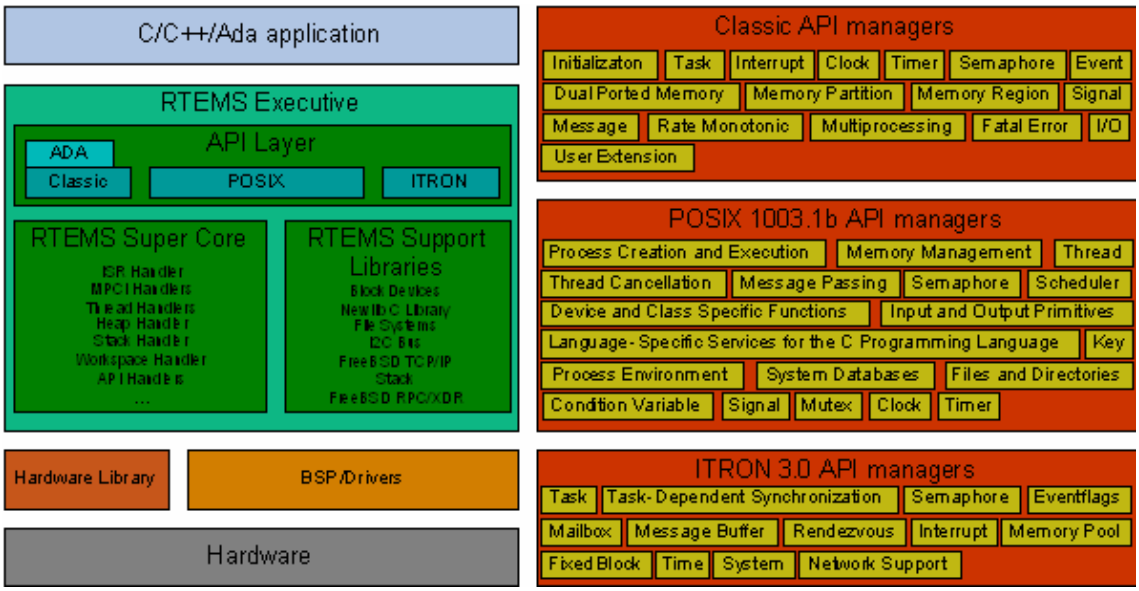


Figure 1: RTEMS Architecture

The hardware support layer encompasses the processor and board dependent files as well as a common hardware library. RTEMS Board Supported Packages (BSP) and Device Drivers is the layer that ports RTEMS to the hardware boards.

The kernel layer is the heart of RTEMS operating system and encompasses the ***super core***, the ***super API*** and several portable support libraries. The ***super core*** is

organized into handlers and provides a common infrastructure and a high degree of interoperability between APIs. The ***super API*** contains the code for services that are beyond any standardization, such as API initialization and extensions support.

The API layer makes the bridge between the kernel and the application. The Classic, POSIX and ITRON APIs are implemented in terms of ***super core*** services. Each API is organized into managers (the right side of the Figure 1 illustrates that). The Ada API is a direct mapping of the Classic interface.

2.1 RTEMS Managers

This section provides a brief description of all the RTEMS Managers and its relevance for the development of RTEMS applications.

2.1.1 Initialization Manager

The Initialization Manager is called by the BSP in the system Initialization phase and is responsible for the initialization and shutting down of the remaining RTEMS Managers and RTEMS Core.

2.1.2 Task Manager

The Task Manager is responsible for the management of RTEMS tasks (in Linux nomenclature RTEMS tasks are referred as threads). It contains primitives to create/destroy tasks, as well as functions to suspend/resume a task or make a task sleep for a specified amount of time, etc.

2.1.3 Interrupt Manager

The Interrupt Manager is responsible for the management of interrupts. It allows the establishment of user-defined Interrupt Service Routines (ISR) and the disable/re-enable of interrupts.

2.1.4 Clock Manager

The Clock Manager is responsible for the management of the time in RTEMS. It can be called by the application (in order to set or get the current time, for example) or by the BSP (in order to update the current clock tick).

2.1.5 Timer Manager

The Timer Manager is responsible for the management of timers in RTEMS. Timers are RTEMS objects that call user specified routines at defined instants in time. There are two forms of timers: raw and server. With the raw timers, the user specified routine is executed in the context of an ISR. As such, it has a limited functionality. The server timers are executed in the context of the highest priority task named “Timer Server”. The user routines have more flexibility in the operations performed (e.g., they can use floating point operations).

2.1.6 Semaphore Manager

The Semaphore Manager is responsible for inter-task synchronization through the use of semaphores. It implements both binary and counting semaphores. Only the binary semaphores implement priority inheritance or priority ceiling protocols (in order to avoid priority inversions scenarios).

2.1.7 Message Queue Manager

The Message Queue Manager is responsible for inter-task synchronization and communication through the passage of messages from one task to another(s).

2.1.8 Event Manager

The Event Manager is responsible for inter-task synchronization through the triggering of events from one task to another. A task may choose to block until a specific event is sent.

2.1.9 Barrier Manager

The Barrier Manager is responsible for the synchronization of a set of multiple tasks at once, i.e. one set of tasks could be waiting in the barrier for some condition; when that condition happens the barrier releases the tasks in order to them proceed with their execution.

2.1.10 Signal Manager

The Signal Manager is responsible for the management of signals in RTEMS. Signals are user-specified routines that are executed when the RTEMS scheduler decides to perform task switching. These user-specified routines can perform more

operations than the ISR's counterpart but have a greater latency (need more time until they are executed). Because the execution of signals is somewhat unpredictable (is only performed during the context switches of the task to which the signal is directed to), it is difficult to integrate in a schedulability analysis. Furthermore, the functionality provided by this Manager can also be obtained through the use of events which are processed in the context of the task.

2.1.11 Partition Manager

The Partition Manager is responsible for dynamically creating fixed-size memory units. The usage of fixed sized memory buffers prevents the memory external fragmentation phenomenon, which happens when the memory is full of small pieces of free memory that because of its size could not be allocated. However it could increase the memory internal fragmentation, which happens when the size of the allocated block is bigger than the requested leading that the application do not use all the memory that it holds. The usage of fixed sized memory buffers could improve performance and decrease the allocation time.

2.1.12 Region Manager

The Region Manager is responsible for dynamically creating variable-size memory units.

2.1.13 Dual-Ported Memory Manager

The Dual-Ported Manager is responsible for converting address representations between internal and external dual-ported memory areas. This conversion is sometimes needed in multiprocessor systems or systems with intelligent peripheral controllers, to convert the internal representation of a memory of one processor/component to another.

2.1.14 I/O Manager

The I/O Manager is responsible for directing applications calls (such as a read or write operation) to the selected driver.

2.1.15 Fatal Error Manager

The Fatal Error Manager is responsible for processing fatal errors in RTEMS. The user can announce the occurrence of a fatal error which can be caught (and recovered) by a user specified error handling routine.

2.1.16 Rate Monotonic Manager

The Rate Monotonic Manager is responsible for make tasks periodic, that is, to produce periodic tasks to be scheduled according to the rate monotonic scheduling analysis.

2.1.17 User Extensions Manager

The User Extensions Manager is responsible for invoking user-specified functions at specific scheduling points, such as task creation, task dispatching, fatal error, etc. The functions specified on these extensions are invoked in the reverse order that they are installed , LIFO[8] order, and they are invoked before the default RTEMS function.

2.1.18 Multiprocessing Manager

The Multiprocessing Manager is responsible for providing inter-processor synchronization and communication.

2.1.19 Stack Bounds Checker

The Stack Bounds Checker is responsible for determining the stack usage of each task. The determination of the stack usage is performed at each context switch and the time to perform this operation is currently proportional to the stack size.

2.1.20 CPU Usage Statistics

The CPU Usage Statistics Manager is responsible for determining the total execution time and CPU percentage of each task. Note that these values are related with the average case. This is not useful in a schedulability analysis for hard real-time systems because the worst case scenario is not accounted.

2.2 Summary

This chapter has presented a RTEMS Architecture. It includes brief explanation for the RTEMS operating system major components such as Classic API, POSIX API and µITRON API. It also presents an overview of the RTEMS executive kernel which includes the API Layer, RTEMS Super Core and the RTEMS support libraries.

[8] LIFO (Last In First Out), the first extension called is the last extension installed.

Due the fact that the RTEMS integrates several features like network protocols, file systems support, multitasking support, interrupt management, inter-task communication and inter-task synchronization, the basic RTEMS kernel is highly configurable.

This chapter also provides a brief explanation for some of the RTEMS Managers such as Initialization Manager; Task Manager; Interrupt Manager; Clock Manager; Timer Manager; Semaphore Manager; Fatal Error Manager; and Rate Monotonic Manager.

Chapter 3

RTEMS Suite Tailoring

This chapter will present and explain the selection process that has been taken to decide the included RTEMS Managers and the excluded managers for the RTEMS tailored version and provides a brief justification for the inclusion or the exclusion of some managers. It explains the necessity to create a new test suite, the Tailorable test suite, which covers all the requirements imposed to the RTEMS Tailorable version.

This chapter, also, explains the necessary workflow to achieve a RTEMS tailorable pre-qualified version by applying the RTEMS patch that will be produced by Edisoft, the Edisoft patch, and it includes a RTEMS Improvement Overview.

3.1 RTEMS Managers Utilization Survey

This section presents the RTEMS Managers Candidates to be validated. To perform a correct tailoring of the RTEMS Operating System, EDISOFT has contacted some of the European space industry members like SAAB and OHB and heard their necessities about the RTEMS managers used on their developed applications as also has heard ESA necessities and requirements for the space applications they need. Based on the collected information Edisoft has performed a representative survey of the possible RTEMS managers used by the European Space community. Based on the Edisoft produced survey, Table 1 provides some high level information of the managers currently being used in the space applications.

RTEMS Managers	SAAB	OHB	ESA
Initialization Manager	Yes	Yes	Yes
Task Manager	Yes	Yes	Yes
Interrupt Manager	Yes	Yes	Yes
Clock Manager	Yes	Yes	Yes
Timer Manager	Yes	Yes	Yes
Semaphore Manager	Yes	Yes	Yes
Message Manager	Yes	Maybe	Yes
Event Manager	Yes	Maybe	Yes
Signal Manager	No	Maybe	Yes
Partition Manager	Yes	Maybe	No
Region Manager	No	Maybe	No
Dual-Ported Memory Manager	No	No	No
I/O Manager	No	Yes	Yes
Fatal Error Manager	Yes	Yes	Yes
Rate Monotonic Manager	Yes	Yes	Yes
Barrier Manager	No	Maybe	No
User Extensions Manager	No	No	Yes
Multiprocessing Manager	No	No	Yes
Stack Bounds Checker	No	No	No
CPU Usage Statistics	No	No	No

Table 1: Edisoft Representative Survey Results

The information contained on Table 1 provides RTEMS Improvement guides for the facilitation of qualification to be done. From a careful observation of Table 1 we could verify that the most important RTEMS managers for the space applications are:

- Initialization Manager;
- Task Manager;
- Interrupt Manager;
- Clock Manager;
- Timer Manager;
- Semaphore Manager;
- Fatal Error Manager;
- Rate Monotonic Manager.

3.2 RTEMS Managers Evaluation

Based on the survey answers, completed by a detailed analysis performed by Edisoft RTEMS CENTRE team, the proposed RTEMS candidate managers are:

- Initialization Manager;
- Task manager;
- Interrupt Manager;
- Clock Manager;
- Timer Manager;
- Semaphore Manager;
- Message Queue Manager;
- Event Manager;
- Fatal Error Manager;
- Rate Monotonic Manager;
- I/O Manager;
- User Extensions Manager.

Following is the list that justifies the included managers:

- Initialization Manager - This manager is necessary for the initialization of the other managers, the system do not work properly if this manager is excluded.
- Task Manager - This manager is necessary for multitasking capabilities and is one essential requirement of the RTEMS tailored version.
- Rate Monotonic Manager - This manager is useful to create periodic tasks.
- Interrupt Manager - This manager is necessary to control the access to the hardware interface.
- Clock Manager - This manager is necessary for time management.
- Timer Manager - This manager is necessary to perform periodic operations such as task switching.
- Semaphore Manager - This manager is necessary for inter-task synchronization and control access to critical sections such as shared variables and hardware resources between others.
- Message Queue Manager - This manager is necessary for inter-task synchronization and communication.
- Event Manager – The team has decided to include this manager because it is useful for inter-task synchronization and communication.
- I/O Manager - This manager is necessary to interfacing the application with the device drivers.

- Fatal Error Manager - This manager is useful for error handling and recovering.
- User Extensions Manager - This manager is useful for, at least, the Fatal Error Manager.

Following is the list that justifies the excluded managers:

- Barrier Manager - Since this functionality can also be achieved with the semaphore manager, even though with greater complexity. This manager is out of scope of the RTEMS Improvement project.
- Signal Manager – For the evaluation performed by Edisoft RTEMS Improvement project team this manager is not often used in space applications and is out of scope of the RTEMS Improvement project.
- Partition Manager – Since the project has to be developed using the code standards of MISRA-C and the rule 20.4, "*Dynamic heap memory allocation shall not be used*" forbids the usage of dynamic memory. The team has decided to not include this manager, staying it out of the RTEMS Improvement scope.
- Region Manager - The functionality provided by this manager has a determinism closely related with the memory units in the system, hence it not safe to use in terms of the amount of execution time required.
- Dual-Ported Memory Manager – Since this functionality is not required in most systems, this manager is out of the project scope.
- Multiprocessing Manager - This manager is not in the scope of the project.
- Stack Bounds Checker - This functionality will not be used on the onboard software because it introduces a great overhead during each context switch to determine the stack usage. This Manager is out of the project scope.
- CPU Usage Statistics - Because the onboard software will not use this feature this manager is out of the scope of the RTEMS Improvement project

3.3 RTEMS Tailoring Plan

The RTEMS Tailoring Plan is illustrated in Figure 2. As depicted, on a first stage, from the original RTEMS 4.8.0 source code, the RTEMS Improvement project team will remove the dead code, as could be seen in Figure 2, in order to produce a minimal subset of RTEMS according to the RTEMS Candidate Managers. At the same time, a test suite will start to be created in order to test this minimal subset. On the second stage, the minimal RTEMS and the test suite will be used as inputs to find and fix bugs present. At this stage the test suite shall be completed. The third stage will take the complete test suite and fixed RTEMS and customize them both in order to produce the

tailorable test suite and tailorable RTEMS, used by the third parties to create and certify their applications.

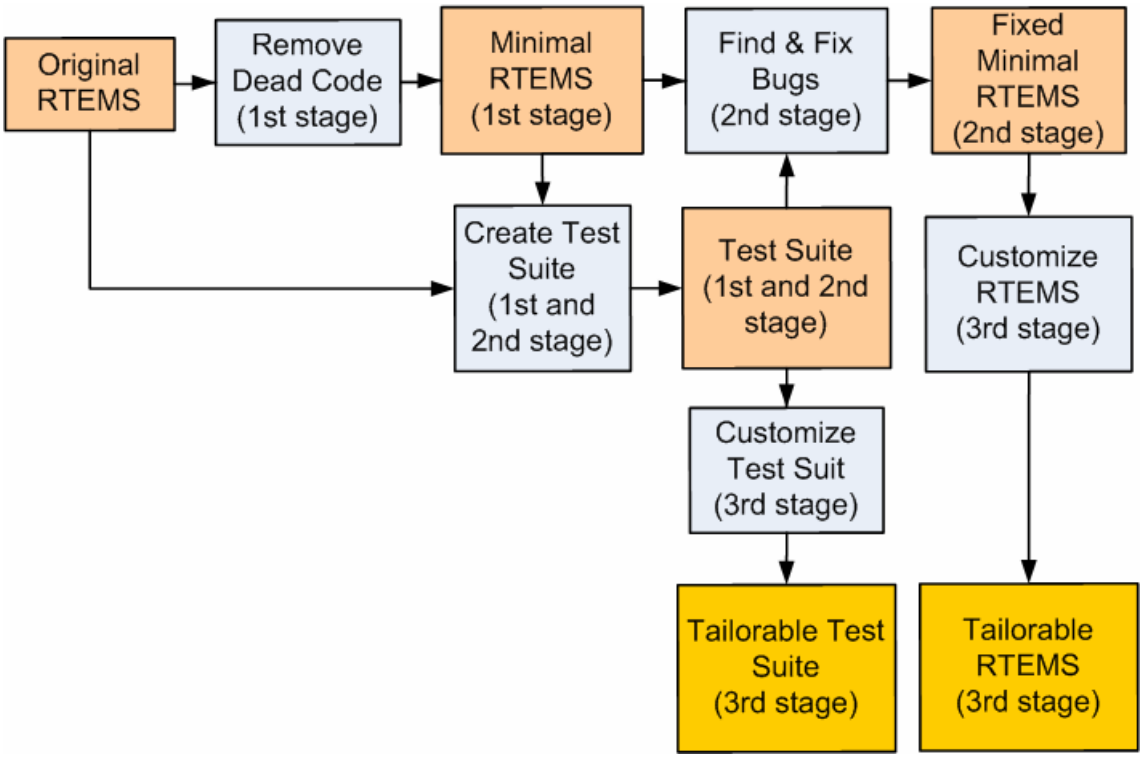


Figure 2: RTEMS Tailoring Plan

3.3.1 Dead Code Removal

Due the fact that the tailored RTEMS version has some restrictions at source code level, it should not contain dead code or code that is not executed. All the unused RTEMS Managers as well as all the unused RTEMS BPS's should be removed from the source tree since it is considered dead code. As could be seen in Figure 2 the removal of dead code, identified by Remove Dead Code in the figure, is in the first stage of the Tailoring Plan and it leads to a RTEMS minimal version, Minimal RTEMS in the figure, which is the start point to build the RTEMS Tailored version.

The unused Managers and BSP's removal implies also the necessity to perform changes in the remaining RTEMS source code in order to remove permanently functionalities and/or code related with the removed Managers and BSP's. The research made by the RTEMS Improvement team shows that some of this unwanted functionalities require some interaction between RTEMS Managers. This requires that

the source code needs to be changed carefully to avoid breaking functionalities on the Managers selected for the tailored version of RTEMS produced by the RTEMS Improvement project.

In a first phase the dead code removal has been done by selecting all the source code files related with the exclude Managers and BSP's and remove them from the Makefile's structure. This causes that when compiling RTEMS the removed functionalities are not built. Since RTEMS is a project completely managed by the gnu auto tools, when changes like add or remove source code files or configuration files is also necessary to change some of the auto tools configuration files, like Makefile.am and configure.ac files, in the directories where the changes have been done.

In a second phase the detection of the dead code has been done by running the test suite produced by RTEMS Improvement team. This test suite is fully featured and covers all the requirements for the RTEMS Improvement project. The test suite runs on all processors and boards selected for the RTEMS Improvement project, ERC32, LEON2 and LEON3, with all the selected managers. By using the GNU Coverage tool GCOV it was possible to verify and analyse what was dead code and what was defensive code.

The code considered to be dead code will be removed through one patch, the Edisoft Patch produced by the RTEMS Improvement project team. This patch, besides remove dead code, when applied transforms an original RTEMS distribution (version 4.8.0) downloaded from the OAR website in a RTEMS tailored version.

3.3.2 Create Test Suite

After removed all the unwanted Managers and BSP's from RTEMS source code, the resultant code needs to be properly tested to meet the requirements for the RTEMS Improvement project. Since the Minimal RTEMS that results from the dead code removal phase have some new features and have features removed, in relation with the original RTEMS version, this implies the design and the creation of a new tailored test suite that covers not the original REMS source code but to cover the source code resultant from the dead code removal phase. This tailored test suite has been design to test and cover all the situations that the Tailored RTEMS could be exposed.

3.3.3 Find and Fix bugs

After the creation of the tailorable test suite, it has been executed to find all the possible bugs that remain in the Minimal RTEMS. All the found bugs have been fixed

by applying the OAR[9] patches, if they already exist, or producing new patches to fix the RTEMS source code. In some cases, when the patch to fix a bug already exist and since the Minimal RTEMS is different from the original RTEMS, it was necessary to change the Minimal RTEMS source code to reflect the changes introduced by the applicable parts of the patch. Applying the full patch could introduce errors in the Minimal RTEMS. In section 4.7 (Test Suite Execution) this step will be explained with more detail.

The execution of the test suite also has been helpful to detect dead code. The output of this phase was the Fixed Minimal RTEMS.

3.3.4 Customize RTEMS

The Fixed Minimal RTEMS was an important input to this phase, since it was the start point of the RTEMS Tailorable version. Customize RTEMS has implied, the necessity to change the RTEMS source code to be MISRA-C compliant and also to clean the source code from all the dead code detected in the Find and Fix bugs phase.

The customization process of RTEMS has implied changes and adds some features to RTEMS in order to meet the project software requirements. One important feature added was a safe state point to where RTEMS switch if a fatal error occurs allowing RTEMS take an action to treat or recover from the fatal error that occurred.

Another feature that has been added was system parameter verification at boot time, this feature checks if all the system parameters are being initialized with the correct values avoiding the wrong behaviour of the system. If during the initialization phase the system detects that a configuration value is wrong, it generates an event that is reported to the user application identifying the parameter that has the wrong value.

One of the added features was a set of macros that alerts the user, using compilation warnings, if the application uses more objects than the maximum number of objects specified for the RTEMS Tailorable, i.e. the system should have a maximum of 64 threads and for some reason the user declare 65 threads in their application, when the user is compiling its application, the user receives a warning message indicating that the number of threads used in the application exceeds the maximum number of threads that the RTEMS Tailored version should handle.

[9] Online Applications Research, http://www.rtems.com

3.3.5 Customize Test Suite

After customize RTEMS to achieve the Tailorable RTEMS version it is also necessary to customize the created test suite to cover the changes made in the Minimal RTEMS. The first version of the test suite was designed to meet the requirements for the RTEMS Improvement project and to test the Minimal RTEMS. Since the Minimal RTEMS has been modified in the RTEMS customization process, the test suite has also to test and cover the modified features. At this phase of the project the customization process of the test suite had implied the necessity to design stress to verify the Tailorable RTEMS version reliability. The customization process also had implied the creation of tests that verify the maximum memory usage of the applications, the performance of it, the size of the produced executables, between other system features.

3.4 RTEMS Tailoring Workflow

In Figure 3 it is illustrated the RTEMS Workflow. In the final of the project there are two ways to achieve the Tailorable RTEMS, the first is downloading the source code from the RTEMS Centre site and the other is applying the Edisoft patch to original RTEMS Version. As depicted, the Edisoft patch is applied to the original RTEMS 4.8.0 source code to remove dead code and fixes bugs found. This will produce a Tailorable RTEMS with all functionalities of the selected managers available and free from all the known bugs, at date of the patch delivery.

Since the applications only need a subset of the RTEMS API, this can be configured to fit the application’s needs so that a minimal footprint can be achieved. Accordingly, only a subset of the Test Suite will also be produced, the necessary tests to ensure the RTEMS correct operation. The tests are also important because is they ensure the pre-qualification of the application, if the Tailored RTEMS passes all of the generated tests.

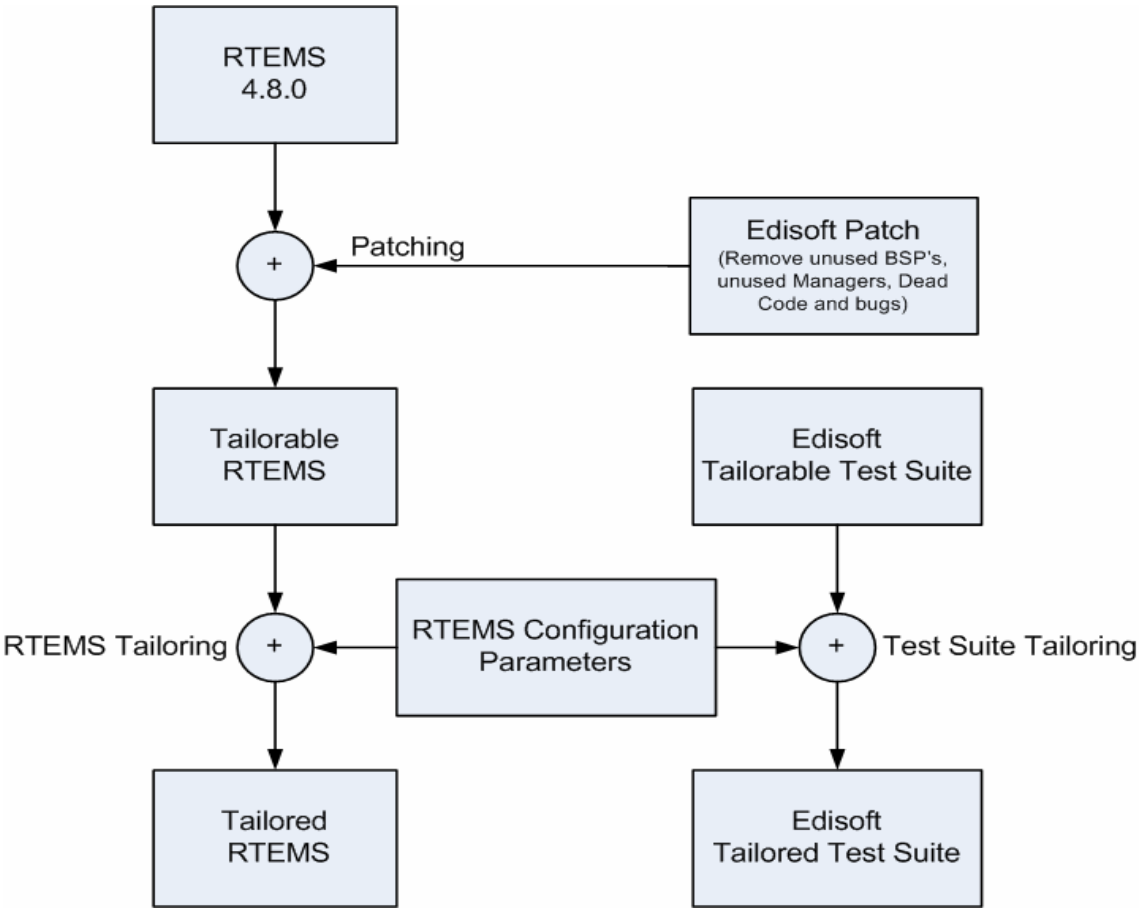


Figure 3: RTEMS Workflow

3.5 RTEMS Improvement Overview

In the Figure 4 is depicted the RTEMS Improvement Overview where is possible to clearly see the target processors and boards for this project. It also shows the Test Suite that was developed to facilitate the qualification of RTEMS. The main purpose of the tests is to validate RTEMS against the Software Requirement Document (**SRD**).

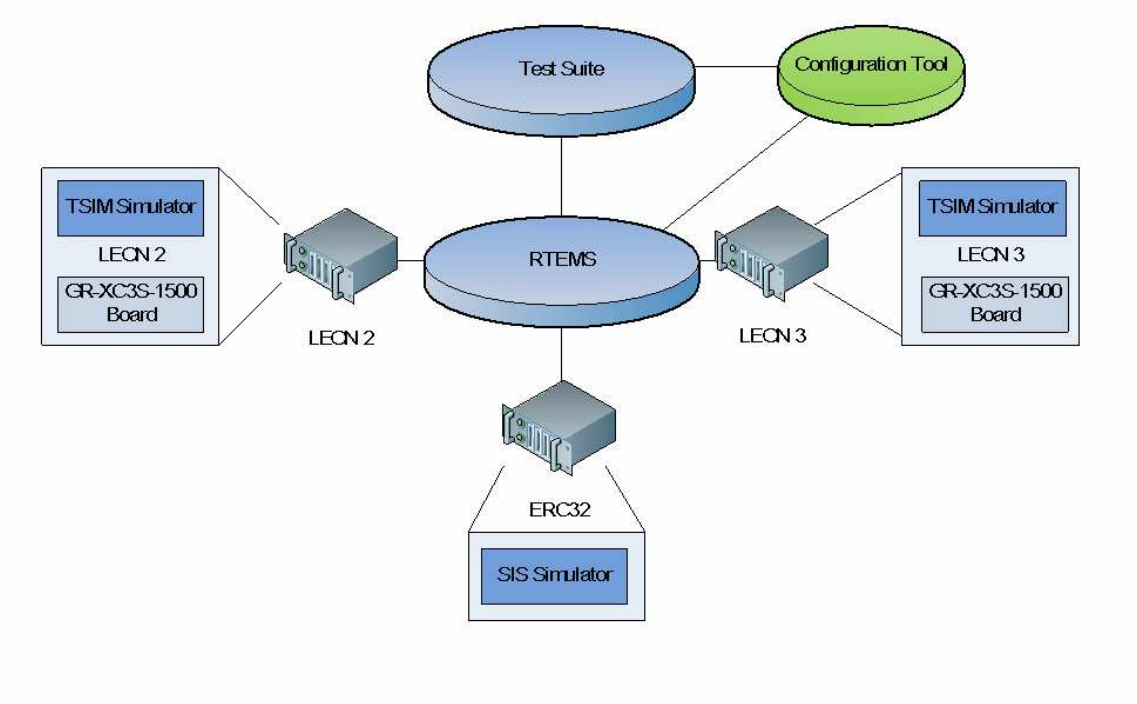


Figure 4: RTEMS Improvement Overview

When the tailorable RTEMS and the tailored Test Suite are complete an additional extra module will be developed and integrated in the RTEMS Operating System, the Memory Management module. This module will be tested by an independent team that will perform the verification and validation of it. Figure 5 shows the RTEMS Improvement Overview now with the new Memory Management module incorporated.

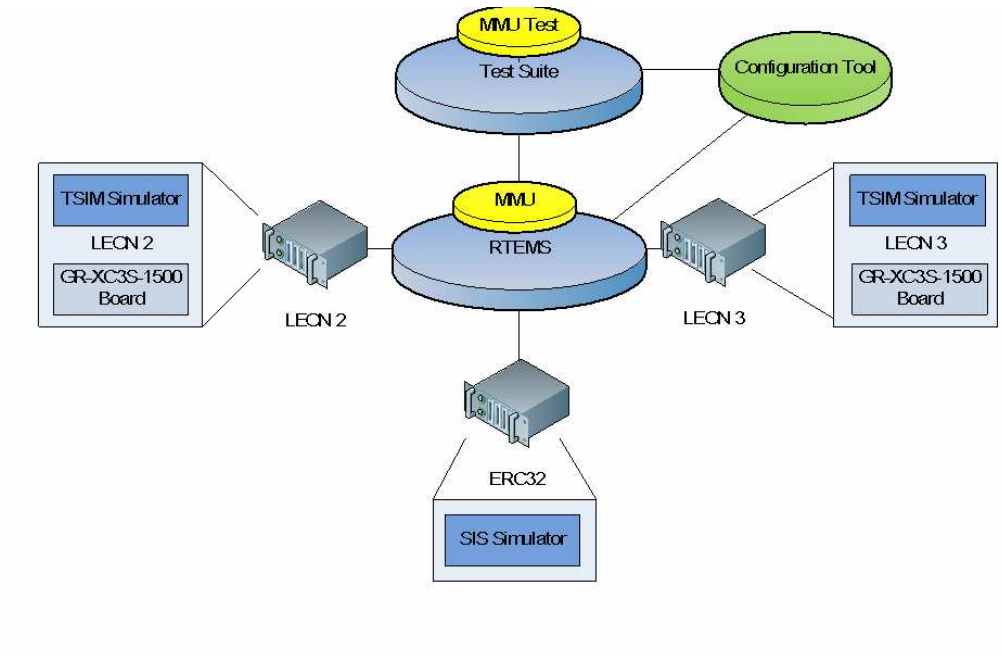


Figure 5: RTEMS Improvement Overview with Memory Management module

The Figure 6 presents the RTEMS Improvement roadmap. At the base of the project we can find the original version of RTEMS operating system, the RTEMS patches that will be produced in the project, the tests that have been created for RTEMS facilitation of qualification.

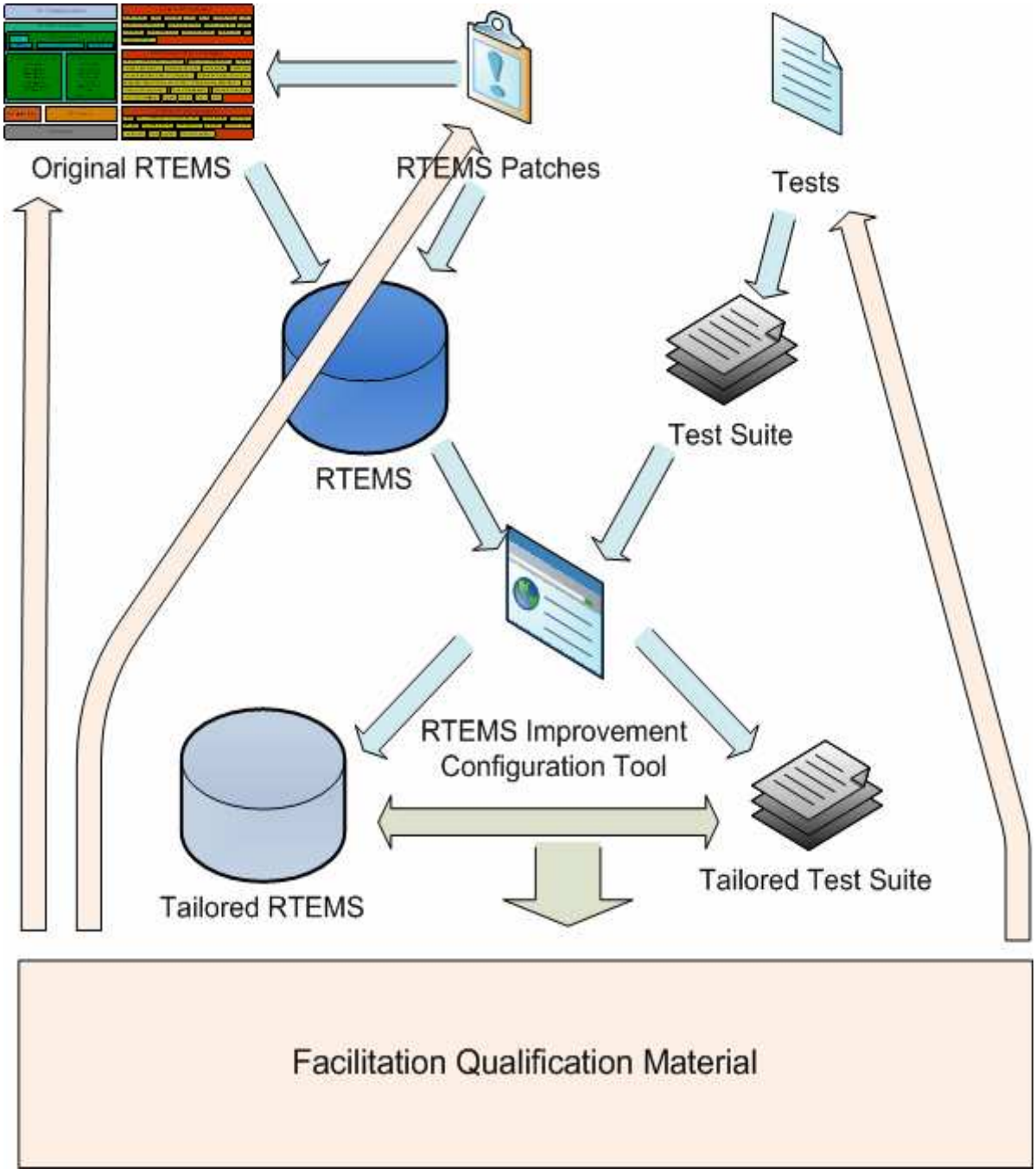


Figure 6: RTEMS Improvement Roadmap

With the original RTEMS software and the RTEMS patches it will be possible to generate a version of the RTEMS Tailored software. Comparing to the original RTEMS, this version has bugs fixed, dead code removed, all managers and APIs present and the complete kernel. In parallel, a Test Suite to test the requirements, produced based in the RTEMS source code and RTEMS User Manual, will be generated and will have the complete set of tests.

The next step is to configure the RTEMS and Test Suite for the space mission. Each space mission has its own requirements, the RTEMS shall be configured to support and accomplish the requirements of the space mission. To support the above described configuration, RTEMS Improvement will develop a configuration tool able to correctly configure the RTEMS and Test Suite. The output, the test suite plus the project documentation, of this activity is the Tailorable RTEMS and the Tailorable Test Suite.

The Tailorable RTEMS and the Tailorable Test Suite are merged to produce and execute the tests for the facilitation of the qualification. The results are then used as inputs for the RTEMS OS updates, the patches to be produced, the MMU updates, the Tests and the MMU tests updates. This will close the loop of the RTEMS Improvement.

The project was divided into two distinguish phases, phase 1 implemented the RTEMS improvement and the MMU and phase 2 maintained and updated the RTEMS test suite. The Figure 7 displays a schematic with the project activities.

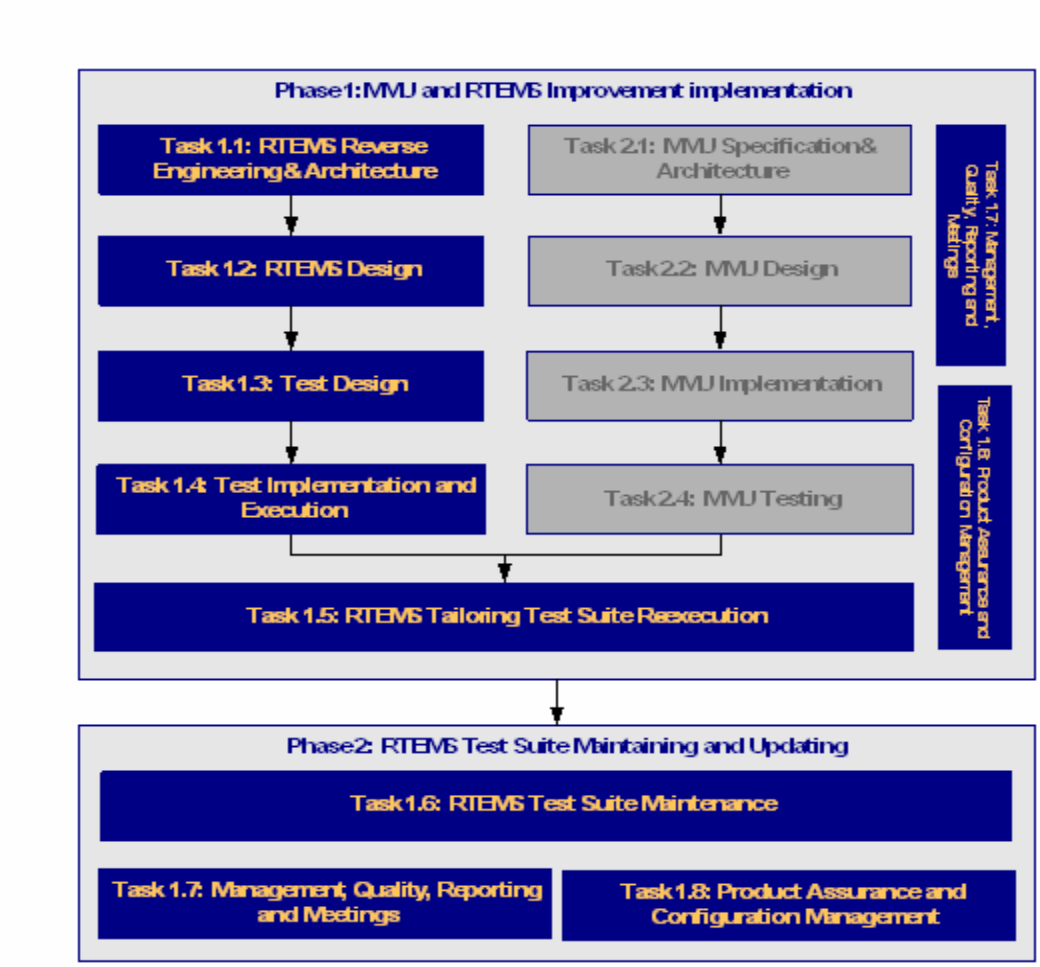


Figure 7: RTEMS Improvement Main Activities

Phase 1, MMU and RTEMS Improvement implementation were developed and all the activities related with the development on the Tailorable RTEMS, the Tailorable Test Suite and the MMU block. The following sub-activities were defined:

- Task 1.1: RTEMS Reverse Engineering & Architecture – RTEMS documentation was analyzed and improved, source code was reversed engineered and the RTEMS architecture was built;
- Task 1.2: RTEMS Design – RTEMS design was performed using the UML language, interface description was performed and RTEMS dynamic aspects were developed and described;
- Task 1.3: Test Design – This activity designed the Test Suite described above;
- Task 1.4: Test Implementation and Execution – Tests were implemented and execution of the tests in the ERC32 and LEON;
- Task 1.5: RTEMS Tailoring Test Suite Re-execution – RTEMS and Test Suite were tailored and executed jointly;

- Task 1.6: RTEMS Test Suite Maintenance – The Test Suite will be maintained for the period of time;
- Task 1.7: Management, Quality, Reporting and Meetings – This task managed the project and was in charge of the reporting to the agency;
- Task 1.8: Product Assurance and Configuration Management – Product assurance and configuration management activities were performed in this task;
- Task 2.1: MMU Specification & Architecture – This task specified will develop the architecture of the MMU block;
- Task 2.2: MMU Design – MMU design will be produced in this activity;
- Task 2.3: MMU Implementation – Code will be developed in this activity;
- Task 2.4: MMU Testing – MMU will be tested in this activity.

3.6 Summary

This chapter has presented and explained the selection process taken to decide the RTEMS Managers that have been included and the managers that have been excluded from the RTEMS tailored version. It explains the necessity of a new test suite for Tailorable RTEMS, the Tailorable test suite.

The chapter explains the necessary workflow by applying the Edisoft patch to achieve a tailorable pre-qualified RTEMS version and gives a RTEMS Improvement Overview.

Chapter 4

RTEMS facilitation of qualification

The qualification of one application for space mission is only possible using both software application and the hardware platform (the mission hardware) where that application will run. Taken this in account the complete qualification must be performed using the produced RTEMS Tailored version adapted to the space mission and running in mission hardware support[10].

One important output of this project is that a Test Suite will be developed in order to facilitate the qualification of RTEMS. This Test Suite will provide 100% statement and decision coverage of the RTEMS source code and will facilitate the qualification based in the Galileo Software Standards SW-DAL B level and a minimum part of it will be used to reach desired coverage.

4.1 Galileo Software standards

The "Galileo Software Standards ", GSWS, are the standards adopted by ESA and it defines procedures to be followed for software engineering, software product assurance and software configuration management.

These standards are defined in a document produced by the Galileo Industries. The GSWS document define five levels, from the most exigent SW-DAL A to most flexible SW-DAL E, and for each type of software and for each SW-DAL, it also define the required life cycle model, the programming languages that can be used in the project development, the structural coverage, the format of headers that identify the author and the company to be used in the code files, the dependability and safety management. It presents formulas for the metrics and defines the format for the project applied documentation.

[10] The hardware platforms used in RTEMS Improvement are outside of scope of the facilitation of qualification.

For the RTEMS Improvement project, the GSWS defines both documentation format and the documents that need to be completed at every deliverable. The GSWS also defines the format used to exchange information between the RTEMS Improvement team and ESA. The usage of GSWS SW-DAL B for the RTEMS Improvement project also implies that the source code changed and produced during the development phase to integrate the RTEMS Tailored version, excluding the tests that belongs to the test suite, needs to have 100% of statement coverage, this means that each RTEMS line of code needs to run at least once, all the declared variables should be used, the application should not contain unused variables. The source code should not contain dead code, source code that never runs. The GSWS SW-DAL B also obliges that the RTEMS source code should have 100% of decision coverage, this means that decision blocks, **if-else, while, do-while, for, switch** and others should run at least once.

4.2 SW-DAL B facilitation of qualification

For the RTEMS Tailorable version, the final product of the RTEMS Improvement project, it was adopted the Galileo SW-DAL level B. This SW-DAL level implies that the RTEMS CENTRE team only can use Assembly, ADA or C languages in development and in structural coverage[11] the tailored RTEMS version, must have 100% statement coverage and 100% of decision coverage. In order to meet the 100% of statement coverage the tailored RTEMS version should not contain dead code. This means all the code that is not covered by the tailorable test suite must be removed.

To analyse the code coverage and the decision coverage the RTEMS Centre team has adopted the open source tool GNU GCOV.

4.2.1 GNU GCOV brief explanation

Gnu GCOV is an open source tool that is used to analyse and give the values for the statement coverage and decision coverage for the source files. To give the coverage information values, the GNU GCOV uses three types of files to give the coverage, the original source code file (in the RTEMS case could be a .c or a .inl file), the .gcno file, which is a graph file where GCOV stores the information about the source file, this .gcno file is created at compilation time when GCC generates the object file. For example if the RTEMS file clockset.c was compiled using the GCOV flags the GCC will create at compilation time two files: the clockset.o, that is the object file, and the

[11] Is the result from the Statement Coverage for the source code, Statement Coverage for the object code, Decision Coverage for the source code and Modified Condition & Decision Coverage for the source code

clockset.gcno that is the graph file for the clockset.c. The graph file contains the information about number of lines that is possible to cover in that source file. It also contains numbers to identify the functions that exists on clockset.c, and other important coverage information.

At running time the application creates a file, the .gcda file that is a data file that contains the counts for the lines and functions executed and information about decision coverage like branches executed.

Later, after running the application, when GCOV is invoked to process the coverage information for a specific application, it reads the information contained in both .gcno and .gcda files. With that information GCOV generates all the code coverage and decision coverage information for each source code file that belongs to the application storing it in a .c.gcov[12] file. In the above example, for the RTEMS clockset.c file, GCOV stores all the coverage information in clockset.c.gcov in a human readable form.

These .gcov files are quite similar to the original source code files. GCOV adds a header where it identifies the source code file, the graph file and the data file, number of times that the program has been run. Besides this header gcov adds two columns in front of each line of source code. The first of these columns indicates the number of times that that line has been executed; the second of these columns identifies the line number.

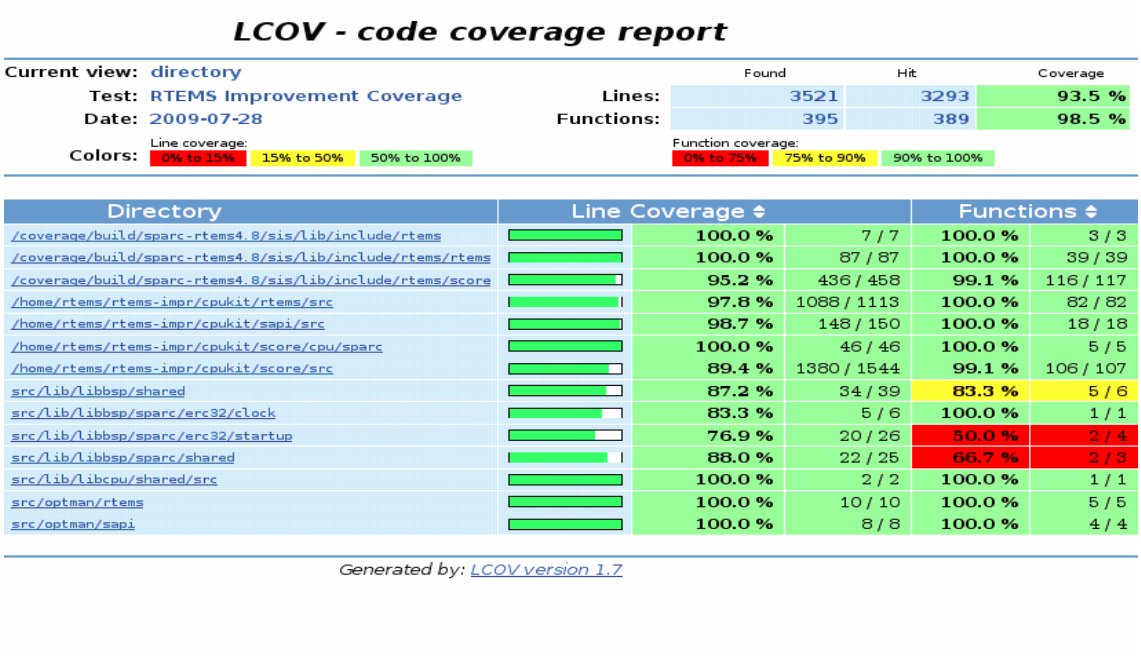
Figure 8: shows an example of a part of the GCOV file for the above example the RTEMS clockset.c source file. In it is possible to see three columns: the first with the count values, the second that identifies the line number and the third that is the source code line. In Figure 8 it is possible to see that line number 53 has been executed 24 times and the line number 57 only had executed one time.

[12] Assuming the source file has another extension GCOV will produce one file named like name_of_the_file.extension_of_the_file.gcov

```
    -:   50:rtems_status_code rtems_clock_set(
    -:   51:  rtems_time_of_day *time_buffer
    -:   52:)
   24:   53:{
    -:   54:  struct timespec  newtime;
    -:   55:
   24:   56:  if ( !time_buffer )
    1:   57:    return RTEMS_INVALID_ADDRESS;
    -:   58:
   23:   59:  if ( _TOD_Validate( time_buffer ) ) {
   18:   60:    newtime.tv_sec = _TOD_To_seconds( time_buffer );
   18:   61:    newtime.tv_nsec = time_buffer->ticks *
    -:   62:                 (_TOD_Microseconds_per_tick * TOD_NANOSECONDS_PER_MICROSECOND);
    -:   63:
   18:   64:    _Thread_Disable_dispatch();
   18:   65:      _TOD_Set( &newtime );
   18:   66:    _Thread_Enable_dispatch();
   18:   67:    return RTEMS_SUCCESSFUL;
    -:   68:  }
    5:   69:  return RTEMS_INVALID_CLOCK;
    -:   70:}
```

Figure 8: Example of a GCOV coverage file

In order to have general coverage information in a more user friendly format the RTEMS Centre team uses the LCOV tool. LCOV tool is a front end for the GNU GCOV that generates fancy html pages to analyse all the coverage information using a web browser, Figure 9 shows the start page for the Tailored RTEMS.



LCOV - code coverage report

			Found	Hit	Coverage
Current view:	directory	Lines:	3521	3293	93.5 %
Test:	RTEMS Improvement Coverage	Functions:	395	389	98.5 %
Date:	2009-07-28				

Colors: Line coverage: 0% to 15% | 15% to 50% | 50% to 100%; Function coverage: 0% to 75% | 75% to 90% | 90% to 100%

Directory	Line Coverage			Functions	
/coverage/build/sparc-rtems4.8/sis/lib/include/rtems		100.0 %	7 / 7	100.0 %	3 / 3
/coverage/build/sparc-rtems4.8/sis/lib/include/rtems/rtems		100.0 %	87 / 87	100.0 %	39 / 39
/coverage/build/sparc-rtems4.8/sis/lib/include/rtems/score		95.2 %	436 / 458	99.1 %	116 / 117
/home/rtems/rtems-impr/cpukit/rtems/src		97.8 %	1088 / 1113	100.0 %	82 / 82
/home/rtems/rtems-impr/cpukit/sapi/src		98.7 %	148 / 150	100.0 %	18 / 18
/home/rtems/rtems-impr/cpukit/score/cpu/sparc		100.0 %	46 / 46	100.0 %	5 / 5
/home/rtems/rtems-impr/cpukit/score/src		89.4 %	1380 / 1544	99.1 %	106 / 107
src/lib/libbsp/shared		87.2 %	34 / 39	83.3 %	5 / 6
src/lib/libbsp/sparc/erc32/clock		83.3 %	5 / 6	100.0 %	1 / 1
src/lib/libbsp/sparc/erc32/startup		76.9 %	20 / 26	50.0 %	2 / 4
src/lib/libbsp/sparc/shared		88.0 %	22 / 25	66.7 %	2 / 3
src/lib/libcpu/shared/src		100.0 %	2 / 2	100.0 %	1 / 1
src/optman/rtems		100.0 %	10 / 10	100.0 %	5 / 5
src/optman/sapi		100.0 %	8 / 8	100.0 %	4 / 4

Generated by: LCOV version 1.7

Figure 9: LCOV Index page for the RTEMS Improvement project

LCOV also navigates in the source code to visualize and analyze the coverage information resultant from the test suite execution without needing to edit each .c.gcov, .h.gcov and .inl.gcov file created be the GCOV tool, which is the default method.

4.2.2 GNU GCOV on RTEMS

To evaluate the code coverage and decision coverage the RTEMS CENTRE team has adopted the GNU GCOV tool. This tool is part of GCC[13] and since that is primarily designed and built to be used for applications that runs in the host computer. This means GCOV will not work on the target[14] boards without be modified. The main GCC library that GCOV uses is libgcov, which uses functionalities from the host libc. The functionalities from the host libc that are used by libgcov will not be present in the application that goes to the target board. By this reason the libgcov has been modified in order to remove all the functionalities that use host libc references from it.

The libgcov changes have been made in order to have a small part of the GCOV library working inside the target, a stand alone libgcov that only have the basic functionalities to GCOV work properly has been created. The rest of the libgcov library was built as a program that works autonomously on the host. This part, which runs on the host, contains all the stuff that uses host libc references.

The Figure 10 shows how the libgcov was separated in order to give the target code and decision coverage information.

In the left side of the Figure 10 is represented the libgcov part that is compiled with the application that goes to the target. The target could be a development board or a simulator. This part of the libgcov is responsible to initiate the GCOV_INFO structures. These structures are where GCOV stores the information values and counts, for the statement coverage and decision coverage. Later when the application is terminating, the target libgcov is called again to start sending the data stored in the GCOV_INFO structures, via serial port, to the host part of libgcov. This host part is the application built with the remaining code of libgcov and executes in the host computer. It is represented in the right side of Figure 10, which is attached to the host serial port collecting data to create the gcda's data files. These gcda's files will be evaluated and used by GCOV to generate the coverage information values for each one of the RTEMS source code files that has been used to build the test-application executing in the target.

[13] GCC is the GNU C Compiler and GCOV is a library that is part of GCC.

[14] The target is the board where the application will run, for example the LEON3 development board is one of the possible targets to the applications developed using the RTEMS tailored.

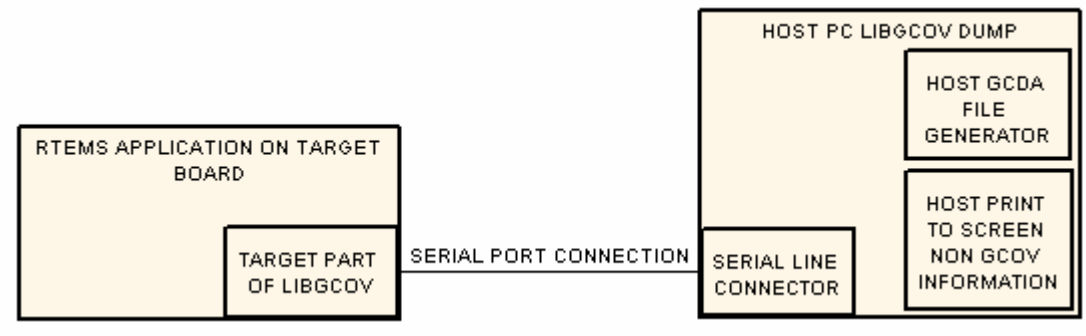


Figure 10: libgcov, work on target boards

After the changes needed in GCOV modules, the makefile structure has also to be changed in order to compile the sources with the GCOV flags and to use the libgcov produced by the RTEMS Improvement team instead of the default GCC libgcov, which do not implement all the necessary functionalities to GCOV work on target. The cross compiler only includes dummy implementations[15] of the GCOV required functionalities.

4.3 RTEMS reverse engineering

The reverse engineering is a technique that follows an approach in the reverse order that normally is used to develop software projects. Normally, in most of the software projects, the development starts by the design of the system followed by the build phase, the production of the source code, until reach the final product when the system is complete and ready to be delivered to the client. In the reverse engineering process we start for the last phase, the system is already built and the main goal is to go in the reverse path, in source code direction. This reverse path implies a deep and careful analysis from both application and source code, when it is available. This careful analysis is necessary to understand the main idea of the RTEMS Improvement project.

The RTEMS reverse engineering was the process that lead us to a most deeply knowledge of the RTEMS functionalities and implementation. This process was a necessary step to achieve one important goal of the RTEMS Improvement Project (Task 1.1 to Task 1.3), obtain a strong and deep knowledge of the RTEMS source code, performing a deep analysis on them, and also to discover how the RTEMS managers are related one to each other and how they are implemented, in order to produce a tailored

[15] Dummy implementations in this case implies that the cross compiler only includes one implementation of the functionality with an empty body this implies that the default libgcov in the cross compiler for the target do not collect any statistics or counts about statement and decision coverage.

version of the RTEMS operating system and a tailored Test Suite. During this phase of the project, a deep assessment of the source code has been made. As result of that deep analysis violations to MISRA-C coding standards and bugs were found. The MISRA-C coding standards define strict rules and recommendations to use when developing and coding applications for real time and critical systems. These rules are defined to promote that the code execution flow and the execution time is determinate, avoiding the usage of all the functionalities that execution time is not determinate. The rules are also defined to make readable and understandable code.

Some of the violations to MISRA-C coding standards, founding in the code analysis phase, have major severity and are related with the usage of the C statements *goto* and *continue* that could the execution of the code unpredictable. The *goto* statement violates the MISRA-C rule 14.4, “The *goto* statement shall not be used” and the *continue* statement violates the MISRA-C rule 14.5, “The *continue* statement shall not be used”. Other violations to MISRA-C coding standards are related with MISRA-C rule 14.7, “A function shall have a single point of exit at the end of the function”. A clearly violation of this rule is stated in Figure 8, that shows part of the code of clockset.c file. The Figure 11 shows in (A) the original code of the function rtems_clock_set, the same code that appears in Figure 8, and in (B) the necessary changes that have been made to the function source code to turn it compliant with MISRA-C rule 14.7.

A

```
...
rtems_status_code rtems_clock_set(
  rtems_time_of_day *time_buffer
)
{
  struct timespec  newtime;

  if ( !time_buffer )
    return RTEMS_INVALID_ADDRESS;

  if ( _TOD_Validate( time_buffer ) ) {
    newtime.tv_sec = _TOD_To_seconds( time_buffer );
    newtime.tv_nsec = time_buffer->ticks *
                 (_TOD_Microseconds_per_tick * TOD_NANOSECONDS_PER_MICROSECOND);

    _Thread_Disable_dispatch();
      _TOD_Set( &newtime );
    _Thread_Enable_dispatch();
    return RTEMS_SUCCESSFUL;
  )
  return RTEMS_INVALID_CLOCK;
}
...
```

B

```
...
rtems_status_code rtems_clock_set(
   rtems_time_of_day *time_buffer
)
{
   struct timespec  newtime;
   rtems_status_code ret_status = 0;

   if ( !time_buffer ) {
        ret_status = RTEMS_INVALID_ADDRESS;
   ) else {
        if ( _TOD_Validate( time_buffer ) ) {
             newtime.tv_sec = _TOD_To_seconds( time_buffer );
             newtime.tv_nsec = time_buffer->ticks *
             (_TOD_Microseconds_per_tick * TOD_NANOSECONDS_PER_MICROSECOND);
             _Thread_Disable_dispatch();
             _TOD_Set( &newtime );
             _Thread_Enable_dispatch();
             ret_status = RTEMS_SUCCESSFUL;
        ) else {
             ret_status = RTEMS_INVALID_CLOCK;
        )
   )
   return ret_status;
}
...
```

Figure 11: rtems_clock_set, changes to be MISRA-C rule 14.7 compliant

Violations to the MISRA-C rule 14.8, “The statement forming the body of a switch, while, do ... while or for statement shall be a compound statement” has also been found.

Some of the bugs that have been found are related with semaphores as described in the section Test Suite Execution.

4.4 RTEMS Documentation

Added to the development of the RTEMS Tailored version and validation test suite, a full set of documentation deliverables for RTEMS operating system has been developed. The documentation associated to RTEMS Improvement project is an

important issue since it is 90% of the deliverable material for this project, The Table 2 presents a list with some of the deliverable documentation produced by the RTEMS Improvement project on which the author has participated.

Deliverable Item	Reference
RTEMS Improvement Software Requirement Document	**SRD**
RTEMS Improvement User Manual and Design Notes	**UMDN**
RTEMS Improvement Design Document	**SDD**
RTEMS Improvement Software Integration Test Plan	**SIP**
RTEMS Improvement Software Unit Test Plan	**SUP**
RTEMS Improvement Validation Testing Specification – Technical Specification	**VTS**
RTEMS Improvement Generic Test Report	**GTR**
RTEMS Test Suite	**SFW**
RTEMS Tailored	**SFW**
Software Development Plan	**SDP**
RTEMS Improvement SOC[16] with GSWS	**SOC**

Table 2: Some of the RTEMS Improvement Deliverables

The first step of the RTEMS Improvement project was generating the documentation that will serve to the basis of the RTEMS Improvement project. The RTEMS Improvement project also uses Galileo Software Standards (**GSWS/SOC**) because this project is a Galileo SW-DAL B classification. The produced documentation for RTEMS Improvement project is a full set of documents that includes:

- RTEMS User Manual and Design Notes (UMDN) for the tailored version of RTEMS OS;
- System Requirements Document (SRD) were the requirements for the project are stated;
- Validation Test Specification (VTS) where it is described all steps and all the stuff related with the Validation test suite;
- Software Integration Test Plan (SIP) like the VTS but for the integration test suit. This last document is under the responsibility of the author of this dissertation with the participation of other elements of the RTEMS Centre team.
- Software Unit Test plan (SUP), this document is like VTS and SIP but is for the unitary test suite.

[16] State Of Compliance

In the documentation part of the project the author of this dissertation also have been actively involved in the production of scripts[17] to analyse and generate documentation, like traceability tables between others, and scripts that also generate documentation in various formats like:

- CSV[18] tables that lately was used as input for necessary project documentation;
- Word documents that have specific parts changed by the scripts and
- Documents in rich text format (RTF).

Most of the produced scripts have been made in:

- Perl;
- JavaScript;
- Visual basic scripts between other scripting languages.

In a later phase of the project the author has also been responsible for the doxygen tool that also generates some graphical documentation associated to the project. The doxygen tool generates graphs that shows the interaction between RTEMS components, it also generate documentation that includes the headers of the functions. All the documentation generated by doxygen is in both HTML and XML format which gives an easy way to browse over the RTEMS functionalities. The doxygen was also used to help in the generation of the Software Design Document (SDD). Figure 12 and Figure 13 are two examples of the documentation automatically generated by doxygen in this case HTML[19]. The doxygen tool also generates other formats of documentation such as XML[20], RTF and **LaTEX**.

[17] a *script* is a sequence of instructions that is interpreted by another program

[18] Comma-Separated-Values (CSV) is a file format that is used to store data structured in a table of lists form, where a row corresponds to a row in the table, the fields of the different columns in a row are separated by a comma.

[19] Hyper Text Markup Language

[20] eXtended Markup Language

Figure 12: doxygen first page for RTEMS SDD

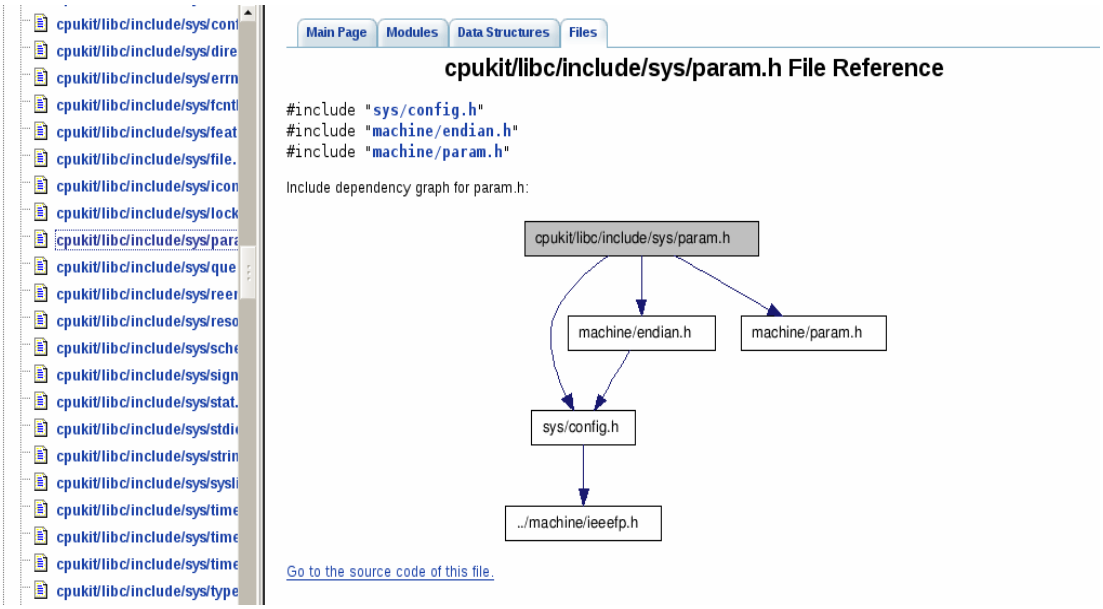


Figure 13: the interaction between header files

The documentation associated to the project is actively being developed, actualized and produced since every delivery of the project the documentation has to be actualized to match the current stage of the project. Also the documentation has to be revised and changed to meet the ESA requirements for that specific delivery.

4.5 Test Suite Design

The test suite have been thinking and designed in order to cover a set of requirements that are stated on the Software Requirements Document, also known as SRD, for the RTEMS Improvement Project. This set of requirements was the result of the evaluation of the selected RTEMS managers, and that in that sense the foreseen necessities for the European Space Community.

The policy adopted for the test suite design was to create subsets of tests that cover the requirements for the specific RTEMS managers used in the RTEMS Improvement project. We also have followed the approach of make black box tests for the validation tests, assuming that the RTEMS source code is unknown at this time, using only the functionalities described in the RTEMS Improvement user manual. It was used grey box and white box for the integration and unit tests. Since this phase comes after the completion of the validation test suite we assume that the RTEMS source code is already known and the test design will be specifically to cover some functionality that is not covered by the validation tests.

By following the above policy, we have noticed that there are some managers that could not be separated from other managers. For example the task manager is needed to create tasks to run the tests for this reason the task manager is present in all of the test subsets. Initialization manager is like the task manager, this manager is also present in all of the other tests since it is needed to initialize all of the other managers and also some specific parts of the hardware.

Another feature of this Project, and is related with the fact that is a reverse engineering Project, is that the test design was started in reverse order, instead of start by unit tests, passing by the integration tests and finishing on the validation tests, the test design have started by the validation tests, designing integration and unit tests only to cover specific situations or code that have not been covered by the execution of the validation test suite.

The test design has also separated the tests by layers. RTEMS have the following well known layers: ***CORE***, ***SUPER CORE*** (SCORE), ***API*** and ***SUPER API*** (SAPI).

Next we discuss with some detail some specific test scenarios. In the design of the tests for the semaphore manager, all of the combinations for semaphores have been taken in account in order to design tests that will cover all the possible situations that probably the designers of space applications will need and to cover several situations as also all combinations that are prone to originate errors. To verify if RTEMS can handle it, the same is done in the test design for the Interrupt manager and all the other managers.

RTEMS offers three types of semaphores: the Counting Semaphores; Simple Binary Semaphores, which act as a Boolean, the semaphore is or is not free, and Binary Semaphores that are really mutex. At the contrary of Simple Binary Semaphores, Binary Semaphores can handle nested access, Priority Ceiling and Priority Inheritance Protocols. Following are presented some of the combinations used in the tests of the Semaphore Manager:

- Binary semaphores (mutex) with FIFO queuing policy
- Binary semaphores with priority queuing policy, using priority inheritance and priority ceiling protocols and without any of these protocols.
- Simple Binary semaphores with FIFO and priority policy
- Counting semaphores with both FIFO and priority policy

The design of the stress tests was made taking into account that system is working at it full capacity. This means that the system is working with maximum configurable threads, that are 64 for the RTEMS tailored version, 256 semaphores, 64 rate monotonic periods, 64 timers, 16 user extensions and 256 message queues. These maximum object numbers for the configuration are stated on the SRD document as a requirement for the RTEMS Improvement Project. In this test the threads are always changing messages between them, make use of semaphores that controls the access to some shared variables and the threads are always sending events to other threads.

The test suite will run as standalone and it will provide 100% statement coverage and decision coverage for RTEMS source code. This is necessary to facilitate the qualification based in the Galileo Software Standards SW-DAL B level.

4.6 Test Suite Implementation

The test suite implementation has closely followed the adopted strategy in the design phase. The majority of the tests have been implemented in a manager fashion, as expressed in the previous section. The test suite implementation was a process that leads us to increase the knowledge of what RTEMS is and how the managers interact one to each others, not in code analysis this time but in a practical way.

As described in the Test Suite Design we had started the test suite implementation creating the validation tests. This approach reveals to be the best choice because in earlier tests execution we had the opportunity to get values rounding 90% in the statement coverage and decision coverage, reducing the number of Integration and Unit tests that needs to be designed and implemented. Starting from those values for

statement coverage and decision coverage, we made an assessment to the produced tests, the validation tests, and check if they could be changed in order to add some new features to better cover the requirements. As a result of this assessment the values for statement coverage and decision coverage have increased to 93%. The remaining 7% of source code was been covered by the integration and unit tests. The source code that has not been covered by the validations tests, the integration test and unit tests, has been accessed and if it was considerate dead code it was been removed. An example of a RTEMS source code file that needs more than one unit test, to verify all the possible return conditions, for the functions defined in it, was the file semtranslatereturncode.c. The unit tests for this file have the purpose to get the RTEMS_INTERNAL_ERROR return value. Return on lines 104 and 144.

In order to implement some of the designed tests we needed to make use of some techniques such as using wrappers and stubs. The use of such techniques was necessary to implement some of integration and unit tests; we have to test components and verify their behaviour. Some times we had to test some specific RTEMS component in isolation. In this case we had used wrappers or stubs that permit us replace the functionalities that are called by that specific RTEMS component testing it against all the possible returns of the functionalities that it calls. In most cases the usage of wrappers reveals to be a better choice since they permits us to make the replacement of some functionality without the necessity to isolate the RTEMS component from the rest of the code.

Like stubs, wrappers also need to have the same name and parameters of the function that it goes to replace but adding the prefix __wrap_ to the function name and at compile time passing one extra parameter to compiler, -Wl,--wrap,function_name and at link time the linker replace the real function for the wrapper function that could or not call the real function.

In the Tasks 1.4 and 1.5 in the test suite execution phase it was need to use wrappers to some RTEMS functions in order to get performance results in the execution of the test suite. The use of wrappers has also been necessary to get the results of code coverage and decision coverage on target. In this case the wrappers were necessary to invoke the GCOV constructors; in order to they collect statistics and to send the collected data to the host computer. This is usually done by the host libc that is not present on target and should, also, not be there because it increases the size of the application that should be as small as possible. Besides that, the RTEMS tailored version does not use the libc[21]. This means that when the application starts it does not

[21] The RTEMS Tailored version should be possible to compile with a bootstrap compiler and without the need of any library external to RTEMS. A bootstrap compiler its one compiler that was built only with

invoke the constructors created by GCOV at compilation time, or writes the collected data to the gcov data files. This is done by functionalities offered by the host libc

In the RTEMS Improvement project this was been done somehow by the hard way. It was solved using one wrapper to the first function called in the system initialization, in this case is the wrapper for the RTEMS bootcard function that is the first function called during the RTEMS initialization. And using a wrapper to last function called when the system is finishing is execution, in this case the function rtems_initialize_executive_late.

4.7 Test Suite Execution

The produced test suite for the RTEMS Improvement Project has been executed in all of the target processor defined in the Project, plus the UNIX BSP. During the phase of implementation the test suite have been already executed not directly on the target boards but in the processor simulators like tsim-leon2, tsim-leon3 and sis64 namely for LEON2, LEON3 and ERC32 processors.

The UNIX BSP has been only used in an earlier phase of the Project to collect some statistics for code coverage and decision coverage. This BSP also give the chance to have a quickly way to verify if the RTEMS answer to the executed test is what we expected.

During test suite execution phase, especially on the execution of the semaphore manager tests, some bugs have been detected, in RTEMS version 4.8.0. These bugs are related with Priority Ceiling Protocol (PCP) and Priority Inheritance Protocol (PIP). The found bugs are violations of the protocol definitions. Next are presented some of the bugs found.

The Simple Binary semaphores should not use Priority Ceiling or Priority Inheritance since the behaviour for these types of semaphores is not defined when using these protocols, despite this; RTEMS has the same behaviour for both semaphores Simple Binary and Binary semaphores. This bug was reported to the OAR Bugzilla and fixed.

In another case the bug is when a task, for example, with priority 10 tries to acquire a semaphore with a priority ceiling value 20. In RTEMS priority value 10 is higher than priority value 20. This acquire generate an error and acquire should fail. This happens if

---

the compiler sources without using any external library. Most of the cross compilers are bootstrap compilers.

the task is the first task trying to acquire the semaphore, but if that we have another task on the system with priority 30, that already has the semaphore when the first task, the higher priority task, try to acquire the semaphore. The higher priority task will wait for the semaphore and when the less priority task releases it the higher priority task acquires the semaphore successfully when it should again receive one error.

4.7.1 Semaphore with Priority Inheritance/Ceiling protocol bug

Another bug detected, this time for both Priority Inheritance Protocols.

Priority Inheritance Protocols

The priority inheritance protocols, Priority Ceiling Protocol (PCP) and Priority Simple Inheritance Protocol (PIP), were introduced to solve the priority inversion phenomenon. This situation occurs when a high priority task is blocked on a semaphore owned by a low priority task. If a middle priority task becomes ready and does not allow the low priority task to free the semaphore, the high priority task will not execute. Hence, in this situation, a high priority task is starved by a middle priority task.

A semaphore with priority ceiling protocol is associated with a priority ceiling value. This priority ceiling must be specified during the creation of the semaphore and corresponds to the priority of the highest priority task that may obtain the semaphore. When a task obtains a semaphore with priority ceiling will have its priority increased up to the semaphore priority ceiling value. When the task releases the semaphore it returns to is previous priority value.

A semaphore with priority simple inheritance (in the RTEMS nomenclature, simple inheritance is equivalent to inheritance) is not associated with any value, like the priority ceiling. Instead, when a task tries to obtain a semaphore with priority simple inheritance (currently held by another task), the priority of the task that holds the semaphore will be increased up to the priority of the highest priority task blocked on the semaphore.

Bug Detection

The bug was detected when a system uses two or more semaphores with a priority protocol (ceiling or inheritance) and uses four or more tasks, all with different priorities. The scenario for this test is quite similar for both protocols: high priority tasks are blocked by semaphores (with a priority protocol) which are all held by a low priority task. When the low priority task starts releasing the semaphores, its priority is now correctly lowered, as defined by the protocol. For the priority ceiling protocol, it should be lowered to the highest priority ceiling value of all semaphores held by the task. For the priority inheritance protocol, it should be lowered to the priority of the highest

priority task still blocked on a semaphore hold by the lower priority task. Instead, for both protocols, RTEMS only lowers the priority of the low priority task when all semaphores are release. This behaviour corrupts the scheduling analysis taken into account when using semaphores with these priority protocols.

4.8 RTEMS Tailored and Test Suite Execution

The RTEMS tailored is the final product of the RTEMS Improvement project. Again this is a RTEMS version 4.8.0 tailored to run specifically over the SPARC ERC32, LEON2 and LEON3 processors and it has a small set of RTEMS managers, the included managers are that are used in space applications. The RTEMS Improvement test suite has been designed in order, like all the other RTEMS sources, to give the maximum code and decision coverage.

The test suite execution will be made majority in target boards. In punctual cases where EDISOFT do not has the specific board the test suite is executed on processor simulators. The existing target boards have a software processor over a Spartan 3 Xilinx FPGA. The use of software processors turns the target board highly configurable. For example to change the current processor in the board the only thing that we need to do is to get the new processor, or the wanted processor, VHDL code and reprogram the existing FPGA board.

4.9 Summary

In this chapter we have described the steps to the qualification process of the Tailorable RTEMS. It starts giving a brief explanation of what is the Galileo Software Standards followed by an explanation of what is necessary to do to achieve the Galileo SW-DAL B requirements. In this chapter an overview of the reverse engineering was given and is explained some of the MISRA-C code standards violations as some of the bugs found. It is introduced the necessary project documentation to meet the Galileo SW-DAL B requirements and explained the steps taken in the test suite design, implementation and execution. Finishing the chapter it haves a brief presentation of the final product, the Tailored RTEMS.

Chapter 5

Memory Management

The memory management is an important module in any operating system since this module makes the interface between the operating system and the MMU that has the responsible to verify and check if one certain user or application could access to some portion of memory.

5.1 Memory Management module

For the RTEMS Improvement project, that the most important goal is the presentation of RTEMS Tailorable version that facilitate the qualification of applications for space missions, the primary goal to the Memory Management module is to provide memory protection to RTEMS operating system.

5.1.1 Memory used by the application

Currently RTEMS determines the necessary memory for the applications at compilation time using a set of C macros [22] to calculate the needed memory space for the application. RTEMS reserve the calculated necessary memory in a continuous space that contains all the variables used by the tasks that are part of the application. When the application is running, is difficult to know where the memory for a specific task starts and ends.

[22] C macros are a set of instructions that are interpreted by the C pre-processor and replaced by the value that they represent in compilation time, the values calculated by the pre-processor are unchangeable during execution time.

5.1.2 RTEMS Memory Protection

Currently RTEMS, version 4.8.0, do not offer mechanisms to perform memory protection. Proofing this are tests that have been developed and verify that one task could change data of another task just using a pointer[23] to a memory position. I.e., task A holds a piece of memory for exclusive use; task B uses a pointer to a variable owned by it self. The execution of the above mentioned tests have shown that by incrementing or decrementing the value of the pointer, changing the memory to where it points, Task B could read, write (changing, clearing or writing wrong data) on any memory address, including the memory owned by task A.

There at least three important memory places, used by any application, which needs memory protection, those three regions are the (stack[24], code and data which also includes the BSS[25] and the heap):

- data section – is the place where the initialized data and variables of the application are stored, this section could also have the heap and BSS section;
- text or code section – is where the code of the application is stored, it contains the instructions that will be loaded by the processor; the instruction pointer register iterates over this section;
- the stack – this is very sensible region because it is the place where the application reserves memory for function variables, stores the values that will be passed from the caller to the called function, stores the address to return after the function finishes is execution. The return value of the function usually is stored in a one general propose register.

The inexistence of a memory protection mechanism implies that any task could read or write in any place on the memory, such as data, code or stack area, of another task, leading that the execution of one task could terminate in an erroneous state.

[23] A pointer is a variable that reference the content of another variable, basically a pointer contains the memory address of the variable that is pointed from it.

[24] The stack is a LIFO (last in first out) structure, usually located in the higher part of memory. In the SPARC Architecture the stack “grows" from the lower memory address to higher memory addresses on every register, immediate value or stack frame being added to it. A stack frame consists at least in a return address

[25] BSS – stands for Block Start by Symbol. In embedded software, the BSS segment is mapped into "Uninitialized RAM".

The problems caused by this unwanted access depends on the importance of the task that have is data changed. I.e., if task A transmits important data, says meteorological data, stored on a memory buffer and task B writes over that buffer, when task A will transmit the stored data, it sends erroneous data because task B meanwhile have changed the data stored in the memory buffer.

5.1.3 Memory Management and Protection Models

There are basically two main models of memory protection, the segmentation model and paging model.

The Segmentation model, briefly, consists in divide the memory in segments, basically four segments, the code segment, the data segment, the stack segment and extra segment, this model is especially used in Intel x86 Architectures and it reveals to be a good choice to implement some memory protection features.

The Paging Model consists in divide the memory in small pieces of the same size, called pages. The Paging model reveals no to be the best model for embedded systems where applications are entirely loaded in physical memory. Although this consideration the memory management model implemented in the SPARC V8 Architecture, namely in the LEON2 and LEON3 processors is the Paging model.

5.2 Hardware Memory Management Unit

5.2.1 Hardware MMU support

The SPARC V8 Reference MMU provides two primary functions:

1) Address translation from the virtual addresses of each running process to physical addresses in main memory. This mapping is done in units of 4KiB and any virtual page can be mapped into any available physical page.

2) Memory protection, so a process cannot read or write the address space of another process. This is necessary to allow multiple processes to safely reside in physical memory at the same time.

5.2.2 Address Translation

The MMU provides address translation from a virtual address (corresponding to a 32 bit CPU) to a physical address (corresponding to a 36 bit Main Memory bus). This mapping allows processes with large memory requirements, e.g., 8 MiB, to be located in different memory areas instead of one contiguous section. This is performed in pages

of 4 KiB each. The following figure shows a 32 bit virtual address is translated to a 36 bit physical address.

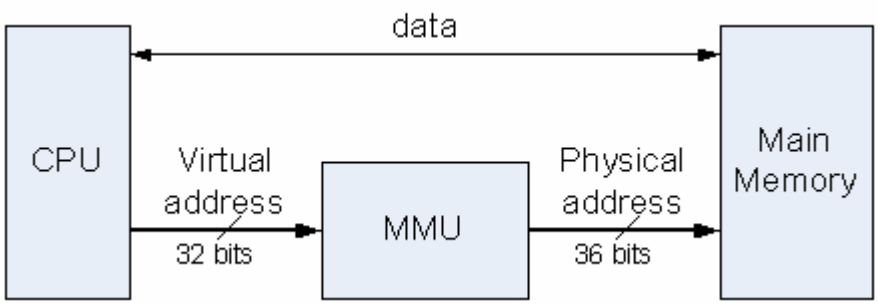


Figure 14: Block Diagram of a SPARC V8 System with MMU

The use of 36 bits for physical address provides a 64GiB physical address space to support large physical memories and memory mapping of 32 bit. A physical address is logically composed of an offset into a 4KiB page and a Physical Page Number.

Pages are always aligned on 4KiB boundaries; hence, the lower-order 12 bits of a physical address are always the same as the low-order 12 bits of the virtual address, and do not require translation. For every valid virtual page resident in memory there is a corresponding Page Table Entry that contains the physical page number for that virtual page. Translating a virtual address to a physical address replaces the virtual page number with the physical page number.

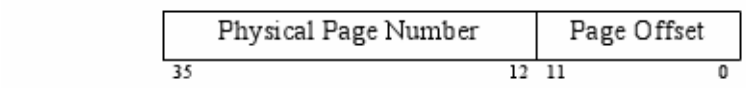


Figure 15: MMU physical address composition

All the address translation information required by the SPARC Reference MMU resides in physically addressed data structures in main memory. The MMU fetches translations from these data structures, as required, by accessing main memory. Mapping a virtual address space is accomplished by up to three levels of page tables, in order to efficiently support sparse addressing. The first and second levels of these tables typically (though not necessarily) contain descriptors (called Page Table Descriptors) which point to the next-level tables. A third-level table entry is always a Page Table Entry (PTE) which points to a physical page. A first- or second-level entry may also be a PTE. A representation of the full three levels of mapping is shown below:

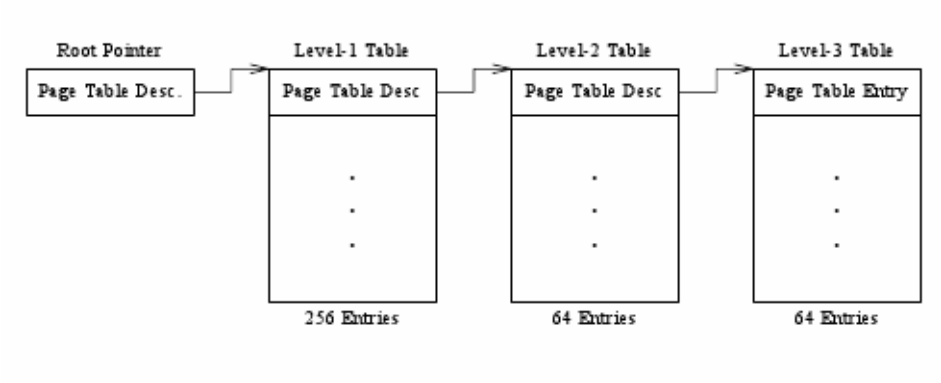


Figure 16: Reference MMU three-level mapping

The Root Pointer Page Table Descriptor is unique to each context and is found in the Context Table. The figure below shows the fields composition of a virtual address:

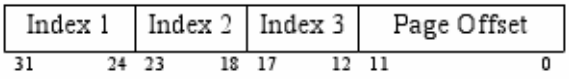


Figure 17: MMU virtual address composition

Each index field provides an offset into the corresponding level of page table. A full set of tables is rarely required.

5.2.3 Memory protection

The MMU also provides memory protection. A mal-functioning process should not write in the address space of another process. This implies protecting memory against unauthorized access. When an application require the access to a specified memory address, instead the application make the access directly, the access should be first verified if the address belongs to the application memory segment, what type of access the application is trying to do, if is a read/write, read only or a write only access, the access type is dependent of the level of the memory protection.

The level of protection is usually described in a page descriptor that stores all the information about the page, this information usually contains the type of access, if the page is present in memory or not. In the SPARC Reference MMU this descriptor is a Page Table Entry (PTE) and it specifies both the physical address of a page and its access permissions.

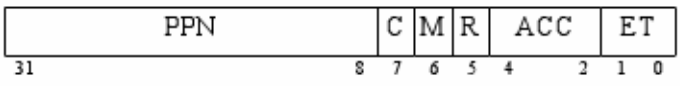


Figure 18: Composition of a Page Table Entry

The relevant PTE fields are defined as follow:

PPN Physical Page Number - the high-order 24 bits of the 36-bit physical address of the page. The PPN appears on bits 35 through 12 of the physical address bus when a translation completes.

ACC Access Permissions - these bits indicate whether access to this page is allowed for the transaction being attempted.

ET Entry Type - this field differentiates a PTE from a PTD.

From the above description we could see that for the memory protection the most relevant field in the page table entry is the ACC. The ACC field has the following interpretation:

Accesses Allowed	User access	Supervisor access
0	Read Only	Read Only
1	Read/Write	Read/Write
2	Read/Execute	Read/Execute
3	Read/Write/Execute	Read/Write/Execute
4	Execute Only	Execute Only
5	Read Only	Read/Write
6	No Access	Read/Execute
7	No Access	Read/Write/Execute

Table 3: ACC access types

Memory Access permissions are checked for each translation; if the requested access violates those permissions, a trap is generated and the appropriate error processing actions should be performed.

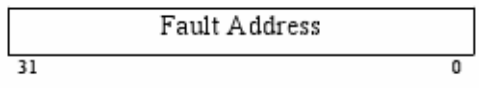


Figure 19: MMU fault address register

The EBE field records the type of bus error. Internal error indications are set when the MMU detects an internal inconsistency. This should be considered a fatal error by software, requiring a system reset.

The highest priority fault is recorded in the Fault Type field. Reading the Fault Status Register clears it. Writes to the Fault Status Register are ignored.

5.3 RTEMS Improvement Memory Management Module

Since the RTEMS Improvement project due the usage of standards like MISRA-C and Galileo SW-DAL B will not use dynamic memory allocation, even if the memory allocation has deterministic time, the main objective of the memory management module that will be developed in the scope of the RTEMS Improvement project will be for memory protection. Taking in account the project restrictions the segmentation model appears to be more useful to implement the memory management module than the paging Model. Besides this, the module will use essentially the hardware MMU features that deal with memory protection like those that verifies the access rights in the MMU page table entries.

The Memory Management module, like all the others RTEMS modules, will be developed and tested according to the Galileo Software Standards SW-DAL B Level.

When the phase of the Memory Management module starts the RTEMS Improvement main activities will change and Figure 7 will stay like the exposed on Figure 20.

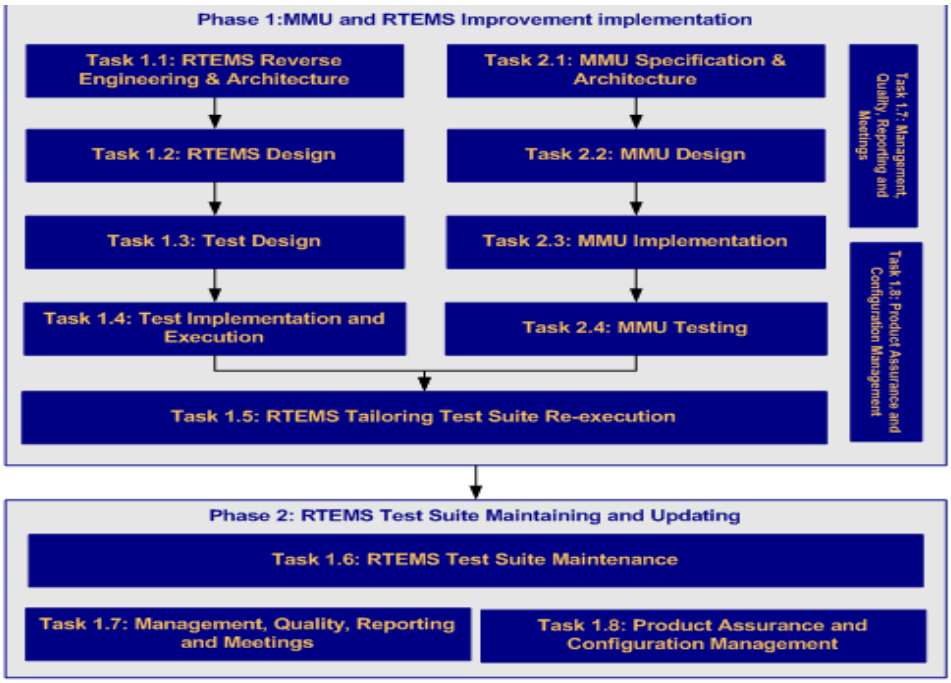


Figure 20: RTEMS Improvement main activities with the Memory Management module integrated.

After this module is implemented the RTEMS Improvement road map exposed in the Figure 6 will be like the exposed on Figure 21.

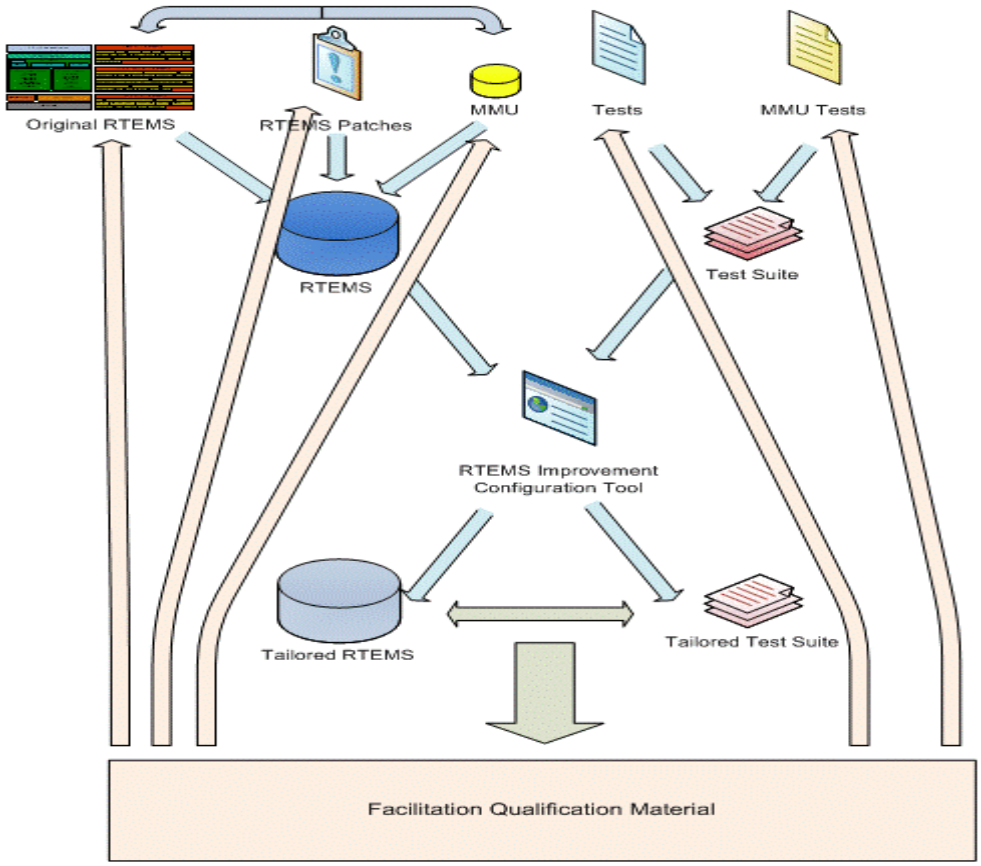


Figure 21: RTEMS Improvement road map with MMU

5.4 Summary

This chapter has presented a brief overview of the memory management and protection models. The chapter also indicates how RTEMS calculate the memory that will be used by the application and ends indicating a possible vision to the implementation of the memory model that will be produced by the RTEMS Improvement project and the changes to the RTEMS Improvement development plan when the develop of the Memory Management module start.

Chapter 6

Conclusion

The present report is result of the work developed in the RTEMS Improvement project, where the author has participated in the several steps of the project, which includes a participation in the design and development of test suite (task 1.3 and 1.4), a light participation in the selection of RTEMS qualification modules (task 1.2) and will include a participation in the definition of the Memory Management module for the RTEMS Operating System (task 2.1 and 2.2).

The RTEMS Improvement project is being developed by the RTEMS Centre team on the Edisoft facilities.

The author has also been involved in producing scripts to generate documentation in an automatic fashion, and was his responsibility the changes that were made to gnu libgcov sources, to GCOV work on target, turning possible to get statement coverage and decision coverage on target boards and processor simulators.

To increase the necessary knowledge to perform part of the work it has been a formation to develop and design hardware components in VHDL for FPGA (Field Programmable Gate Arrays).

This experience have also contributed to learn, as software engineer, new coding standards associated to development to embedded systems, one of this standards is MISRA-C coding standards. MISRA stands for "**The Motor Industry Software Reliability Association",** www.misra-c.com. MISRA-C is a book where we can find the basic that we should follow when we are developing for embedded systems, this standards could also be applied to other systems since it is basically a set of simple rules to make the code more easily to read and more understandable.

During the test suite development phase, the RTEMS Centre team has noticed the necessity that some embedded systems have in memory protection, currently from the results obtained in the simulators, RTEMS do not offer any kind of memory protection, instead it gives to the application extra memory expecting that if one interruption occurs

that extra memory be sufficient to store all necessary data that could result from the interrupt.

Extensive Abstract in Portuguese

O RTEMS (**R**eal **T**ime **E**xecutive for **M**ultiprocessor **S**ystems) é um sistema operativo de tempo real "open source" desenhado e desenvolvido para ser competitivo com sistemas comerciais idênticos. O RTEMS encontra-se actualmente numa fase de desenvolvimento bastante activa, nomeadamente para as aplicações espaciais. Tendo em conta este aspecto foi criado um centro de investigação, o Edisoft RTEMS Centre, em colaboração com a Agência Espacial Europeia (ESA) sobre o sistema operativo RTEMS, com o intuito de ajudar a comunidade espacial europeia na utilização deste sistema operativo.

As primeiras actividades da equipa de desenvolvimento do RTEMS Centre consistiram na criação de ferramentas de suporte e auxílio à configuração e compilação do sistema operativo RTEMS. Numa fase mais avançada do RTEMS Centre foi iniciado o projecto RTEMS Improvement, que visa disponibilizar uma versão do sistema operativo RTEMS, esta versão tem como objectivo facilitar a qualificação de aplicações, e está optimizada para funcionar com os processadores da família SPARC, nomeadamente os processadores ERC32, LEON2 e LEON3 na a sua utilização em aplicações espaciais. O processo completo de qualificação das aplicações espaciais só poderá ser concluído conjugando o sistema com os componentes de software da aplicação e com o hardware onde a mesma irá correr.

Para que possa ser produzida a versão adaptada ou ajustada do sistema operativo RTEMS é necessário seguir determinadas normas quer para a produção de documentação, quer para a correcção e modificação do próprio código fonte. Para o projecto do RTEMS Improvement foi necessário seguir um conjunto de normas base resultantes da conjugação das normas definidas: pela Galileo Industries, Galileo Software standards (**GSWS** [RD1]), que impõe modelos para a documentação e para o código fonte; e pelas normas definidas no âmbito do manual MISRA-C [RD0], da MIRA Limited, que impõe modelos de codificação normalizados para aplicações em sistemas embebidos e de missão crítica.

Para além de toda a documentação associada ao projecto foi necessária a criação de uma nova bateria de testes. Esta foi criada com objectivo de cobrir todos os requisitos

impostos à RTEMS Tailored version, versão do sistema operativo RTEMS resultante do projecto RTEMS Improvement, para que a referida versão do RTEMS possa efectivamente facilitar o processo de qualificação das aplicações espaciais, pois este é um item essencial do projecto: produzir uma versão do sistema operativo RTEMS que facilite a qualificação das aplicações espaciais.

Numa fase mais avançada do projecto do RTEMS Improvement, e quando todas as etapas anteriores estiverem concluídas será desenvolvido um módulo de gestão de memória para o RTEMS para a classe de processadores LEON3. Este desenvolvimento numa fase mais avançada deve-se ao facto deste módulo ser um objectivo secundário do projecto, relembrando que o principal objectivo do projecto visa a facilitação da qualificação de aplicações baseadas no sistema operativo RTEMS para o espaço.

Devido às imposições colocadas pela utilização das normas da Galileo industries a RTEMS Tailored version não pode conter código que não seja executado, *dead code*, o que implicou na realização deste projecto a necessidade de remover todo o código fonte que demonstrasse não ter sido executado, com excepção do código fonte que foi considerado código defensivo. Foram ainda eliminadas todas as BSPs (Board Support Packages) para os processadores que não foram utilizados no desenvolvimento do projecto.

Para obter a cobertura do código fonte que foi executado nos processadores-alvo foi necessário adaptar uma biblioteca da ferramenta “open source” GNU GCOV. Esta biblioteca é incluída na aplicação para processar os resultados finais das contagens das linhas de código fonte que foram executadas. As alterações efectuadas à biblioteca acima citada foram necessárias, porque originariamente a ferramenta utiliza funcionalidades que não se encontram presentes na maioria dos sistemas operativos para sistemas embebidos, não facultando assim a cobertura de código fonte para as aplicações embebidas a funcionar no sistema alvo.

Conclusão

O projecto RTEMS Improvement tem como missão a apresentação de uma versão ajustada do sistema operativo RTEMS, a RTEMS Tailored version, esta versão ao seguir a norma Galileo SoftWare Standards (GSWS [RD1]) no seu nível B, SW-DAL B, responde aos mais apertados requisitos apresentados para o desenvolvimento de aplicações para sistemas de tempo real e de missão crítica. A RTEMS Tailored version pretende dar e ser resposta às necessidades apresentadas pela comunidade espacial europeia.

Bibliography

[RD0] MIRA Limited, *MISRA-C: 2004 Guidelines for the use of the C language in critical systems*, MIRA Limited, Watling Street, Nuneaton, Warwickshire CV10 0TU, UK, © MIRA Limited, 2004.

[RD1] *Galileo Software standards* (GSWS), Galileo Industries Doc. No.: GAL-SPE-GLI-SYST-A/0092, ISSUE: 7, DATE: 24/05/2004

[RD2] Kernighan, Brian W., Ritchie, Dennis M., *THE C PROGRAMMING LANGUAGE, SECOND EDITION*, PRENTICE HALL SOFTWARE SERIES, 1988

[RD3] RTEMS Improvement RTEMS managers candidate evaluation report.

[RD4] SPARC International, The SPARC Architecture Manual Version 8.

[RD5] GNU website, www.gnu.org.

[RD6] Silva, H., Constantino, A., Coutinho, M., Freitas, D., Faustino, S., Mota, M., Colaço, P., Sousa, J., Dias, L., Damjanovic, B., Zulianello, M., Rufino, J.: RTEMS CENTRE – Support and Maintenance CENTRE to RTEMS Operating System, DAta Systems in Aerospace (2009)

[RD7] Silva, H., RTEMS CENTRE Final presentation and Final Report, ESTEC (European Space Research and Technology Centre), Noordwijk - Netherlands (2008)

[RD8] RTEMS CENTRE website: http://rtemscentre.edisoft.pt

[RD9] RTEMS website: http://www.rtems.com

[RD10] Constantino, A., Freitas, D., Mota, M., Silva, H.: RTEMS CENTRE Software System Specification, RTEMS CENTRE project (2008)

[RD11] Coutinho, M.: RTEMS Managers Candidate Evaluation Report, RTEMS Improvement project (2009)

[RD12] Coutinho, M.: RTEMS Improvement Generic Test Report, RTEMS Improvement project (2009)

[RD13] Freitas, D.: RTEMS Improvement Software Budget Report, RTEMS Improvement project (2009)

[RD14] Dias, L.: RTEMS Improvement Preliminary Software Criticality Analysis Report, RTEMS Improvement project (2009)

[RD15] Colaço, P., Coutinho, M.: RTEMS Improvement Software Design Document, RTEMS Improvement project (2009)

[RD16] Coutinho, M.: RTEMS Improvement Software Requirements Document, RTEMS Improvement project (2009)

[RD17] Polock, D., Zöbel D.: Conformance Testing of Priority Inheritance Protocols, IEEE (2000)

[RD18] Sha, L., Rajkumar, R., Lehoczky, J. P.: Priority Inheritance Protocols: An Approach to Real-Time Synchronization, IEEE (1990)

[RD19] Lam, K., Son, S. H., Hung S.: A Priority Ceiling with Dynamic Adjustment of Serialization Order, IEEE (1997)

[RD20] Sha, L., Rajkumar, R., Son, S. H., Chang, C.: A Real-Time Locking Protocol, IEEE (1991)