Graduation Project Documentation

July 2023

Faculty of Engineering

Computer Engineering Department

Under Supervision of:

Prof. Dr. Amr El-Sayed

Dr. Ahmed Moro

Eng. Mohned Mohamed

Eng. Kerollos Gamal

Security Based OSEK OS

Prepared by:

Ahmed Mohamed ElShahat

Aliaa Mohamed ElFeky

Esraa Ahmed Taha

Omar Ahmed Kadry

Ziad Ahmed Hossam

Project number: 5

Sponsored by:

SIEMENS

Contributions

Ahmed Mohamed El-Shahat	• Task Management • Schedule • Alarm Management • Linker Script • Startup Code • Hash Module • Bridge Module • Unit Testing • Integration Testing • Documentation
Esraa Ahmed Taha	• Task Management • Event Handling • Resource Management • Make File • Bridge Module • Configurator tool • User Guide • Documentation
Ziad Ahmed Hossam El-Din	• Task Management • Event Handling • Resource Management • Alarms • AES Module • Documentation
Aliaa Mohamed El-Feky	• Task Management • Timer Implementation • Alarms • Unit Testing • Integration Testing • Documentation
Omar Ahmed Kadry	• Task Management • Bridge Module • Startup Code • User Guide • Unit Testing • Integration Testing • Demo project • Documentation

Table of Contents

1. Abstract

The purpose of this project is to develop a Single Core AUTOSAR Operating System designed to make multitasking possible on a hardware security module (HSM) in an Electronic control unit (ECU). The HSM provides cryptographic and security-related functions, such as secure booting, secure communication, and secure storage of sensitive data.

The operating system is developed with a focus on reliability and security, and will be designed to enable the development of secure and reliable automotive software. The system will be compatible with the AUTOSAR standard, allowing it to be used with a wide range of automotive electronic control units (ECUs) from different manufacturers and a wide range of different compilers too.

This document will provide a detailed specification of the system requirements, including functional and non-functional requirements, as well as any constraints or assumptions that need to be taken into account during development. The document will serve as a guide for the development team, ensuring that the system is developed to meet the needs of the automotive industry and the specific requirements of the HSM in the ECU.

2. Introduction

2.1 Background of project work

The rapidly growing connectivity of vehicles is opening up numerous opportunities for new functions and attractive business models. At the same time, the potential for cyber-attacks on vehicle networks is also growing. Such attacks threaten the functional safety of the vehicle and could cause financial damage.

This has attracted automotive cyber attackers to try to control the car remotely or get sensitive information that is supposed to be confidential either for the customer or the manufacturer

Requirements for security of information in the vehicle are growing along with the complexity of vehicle functions. In addition to protecting internal vehicle data, the vehicle’s connections to the outside world in particular require heightened protection against unauthorized access.

Hence, more Cryptographic operations and calculations are required to be done on a hardware that’s Physically secured to be trusted enough as the trust anchor for the rest of the ECUs, Hence The requirement of a OS to multitask operations on the different Cryptographic Hardware Accelerators on the HSM.

2.2 Establish the context

This project is supposed to provide the HSM developers with a lite version of OSEK OS that has only the required OS modules for HSM core. The final product will be seen as APIs that allow controlling a user-defined set of OS objects to make the best use of the HSM Hardware accelerators.

The implementation of this project follows the AUTOSAR standards, hence it will be portable on future versions of HSM and different OS application developed by different Compilers, different Hardware Targets.

The product faces some risks including:

1) Performance Issues: The hardware accelerators on the HSM are relatively fast which means that OS overhead needs to be minimum in order to efficiently multitask. This has caused the need for continuous optimization for each part of the code causing slower development rates.
2) High costs: The HSM is considered to be a new technology that exists on expensive MCUs such as Aurix TriCore and RH850. While the project didn't need the whole MCU, the HSM wouldn't exist separately.
3) Delayed Acquirement of the board due to customs guidelines and fear of new infamous electronic devices in Egypt.
4) The Confidentiality of software and documentation for both the board and the compiler which may have caused a lack of clarity until a late stage in the process.

2.3 Development Process

The project followed a simple Agile-like software development methodology that consisted of running each OS module through five main stages

- Discussion on its importance to the HSM core.
- Design & allowed Interactions.
- Implementation and unit testing.
- Refactoring and optimization.
- Integration testing with other OS modules on the Hardware (using Trace32).

Using a student license for Git Kraken Software we were able to work concurrently on different parts of the project with minimum number of conflicts.

Each member worked on each part of the Operating system in order to gather the most ideas for the final design of the OS kernel.

2.4 Motivation

2.4.1 Operating System on HSM Core

In the previous the host core was requesting tasks or cryptographic operations from HSM core in sequential form of requests which we call bare-metal by the evolution of the HSM Core technology and increase the Hardware Accelerators, it has become harder to develop bare-metal applications that would utilize these resources well enough.
Which is why an Operating System that can utilize Hardware accelerators by multitasking, managing Priority of Tasks and handling the need of periodic and synchronization for these Resources is required. The operating system follows AUTOSAR OS standard (which is built on OSEK OS standards).

2.4.1.1 Operating system components

Operating System follow scalability class one (**ECC2**) by implementation of components as Task Management, Event Handling, Resource Management, Timers and Alarms, Allows Multiple tasks activation.

The following conformance classes are defined:

- BCC1 (only basic tasks, limited to one activation request per task and one task per priority, while all tasks have different priorities)
- BCC2 (like BCC1, plus more than one task per priority possible and multiple requesting of task activation allowed)
- ECC1 (like BCC1, plus extended tasks)
- ECC2 (like ECC1, plus more than one task per priority possible and multiple requesting of task activation allowed for basic tasks)

The portability of applications can only be assumed if the minimum requirements are not exceeded. The minimum requirements for Conformance Classes are shown in the Figure 3-3.

	BCC1	BCC2	ECC1	ECC2
Multiple requesting of task activation	no	yes	BT[3]: no ET: no	BT: yes ET: no
Number of tasks which are not in the *suspended* state	8		16 (any combination of BT/ET)	
More than one task per priority	no	yes	no (both BT/ET)	yes (both BT/ET)
Number of events per task	—		8	
Number of task priorities	8		16	
Resources	RES_SCHEDULER	8 (including RES_SCHEDULER)		
Internal resources	2			
Alarm	1			
Application Mode	1			

Figure 3-3 The minimum requirements for Conformance Classes

3. Requirements

3.1 Specific Requirements

3.1.1 HSM Control

In order to control the behavior of the HSM inside the ECU of the car, we need to provide a mechanism for managing the execution of tasks and resources. This may include features such as task prioritization, resource allocation, and interrupt handling. For example, we needed to implement a task scheduler to ensure that high-priority tasks are executed first, or a resource allocation mechanism to ensure that multiple tasks do not attempt to access the same resource simultaneously.

3.1.2 OSEK Compliance

The OSEK/VDX standard defines a number of requirements for real-time operating systems, including features such as task management, interrupt handling, and communication between tasks. To ensure OSEK compliance, we needed to implement these features and ensure that they meet the relevant standards and specifications.

3.1.3 Security

In order to manage and execute requests that come from the hosts, our project will provide a secure mechanism for handling sensitive data and ensuring that only authorized hosts are able to access the system. This may include features such as encryption, authentication, and access control, to protect against unauthorized access and data theft.

3.1.4 Real-time Behavior

Real-time behavior is essential for many automotive applications, as tasks must be executed within specified time constraints in order to ensure safe and efficient operation. To ensure real-time behavior, we needed to implement a task scheduler and interrupt handling mechanism, to ensure that tasks are executed in a timely and predictable manner.

3.1.5 Performance

High performance and low latency are critical for many automotive applications, as delays or bottlenecks in the system can cause significant safety and performance issues. To ensure high performance, we needed to optimize code and carefully manage system resources, to ensure that requests are executed quickly and efficiently.

3.1.6 Compatibility

Our project will be compatible with the specific hardware and software components of the car, including the ECU and any other relevant components. This may require to work closely with the manufacturer of the car or ECU, to ensure that your implementation is compatible and meets all relevant standards and specifications.

3.2 System Requirements

3.2.1 Functional Requirements

3.2.1.1 Task Management Module

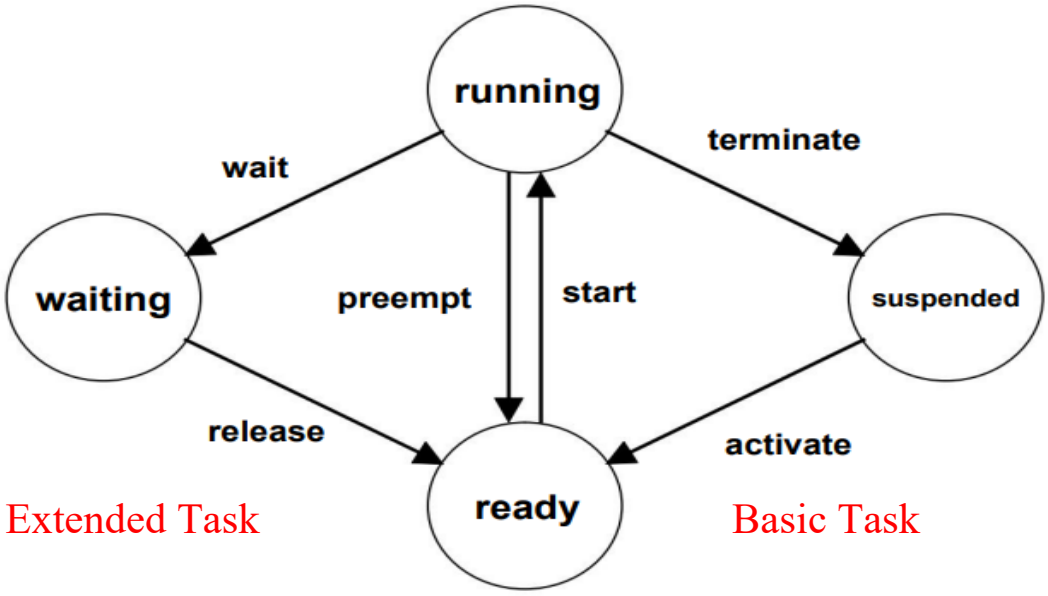
Task Concept

Complex control software can conveniently be subdivided in parts executed according to their real-time requirements. These parts can be implemented by the means of tasks.
A task provides the framework for the execution of functions.
Two different task concepts are provided by the OSEK operating system:

- Basic Tasks
- Extended Tasks

Basic Tasks vs Extended Tasks

The main difference between basic and extended tasks that the extended task being allowed to use the operating system call WaitEvent, which may result in a waiting state.
The waiting state allows the processor to be released and to be reassigned to a lower-priority task without the need to terminate the running extended task.



Transition	Former state	New state	Description
Activate	suspended	ready	A new task is set into the ready state by a system service. This task put into Ready Queue.
start	ready	running	A ready task selected by the scheduler is executed.
wait	running	waiting	The transition into the waiting state is caused by a system service. To be able to continue execution, the waiting task requires an event.
release	waiting	ready	At least one event has occurred which a task has waited for.
preempt	running	ready	The scheduler decides to start another task. (This task has higher priority than the running task) The running task is put into the ready state.
terminate	running	suspended	The running task causes its transition into the suspended state by a system service.

Running

In the running state, the CPU is assigned to the task, so that its instructions can be executed. Only one task can be in this state at a time.

Ready

All functional prerequisites for a transition into the running state exist, and the task only waits for allocation of the processor. The scheduler decides which ready task is executed next.

Waiting

A task cannot continue execution because it shall wait for at least one event.

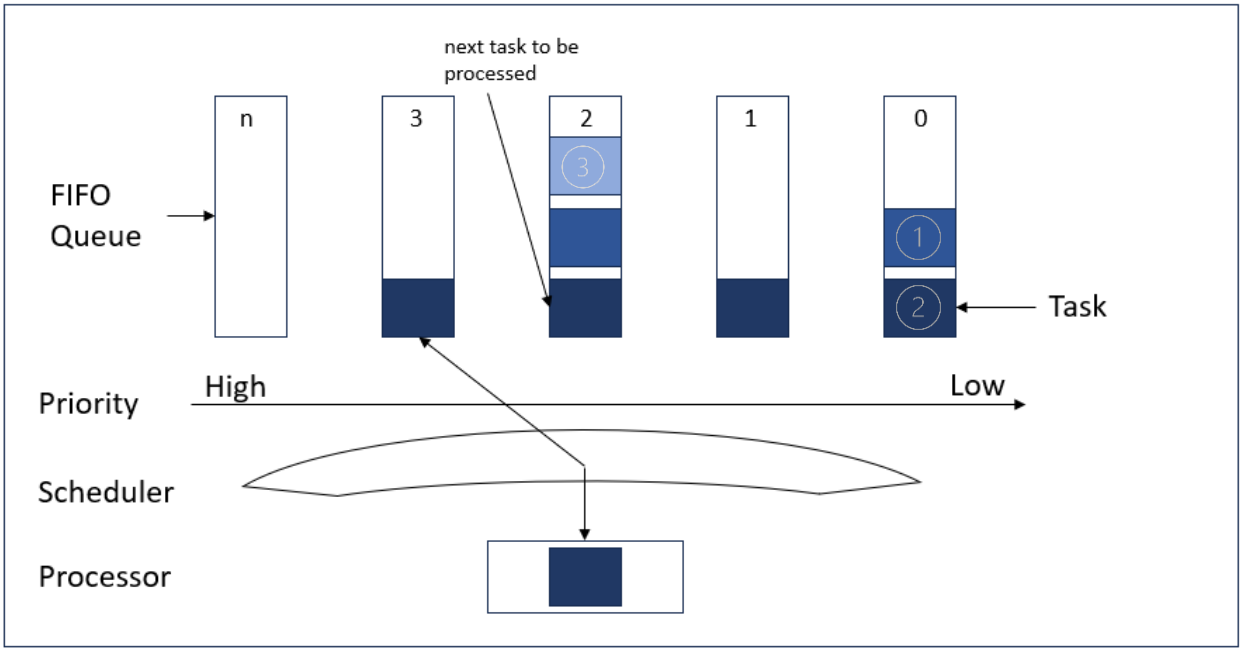
Suspended

In the suspended state the task is passive and can be activated.

Task priority

The lowest priority value is assigned as 0, and higher values correspond to higher priorities.
The system does not support dynamic priority management to optimize efficiency.
Consequently, task priorities are defined statically, and users are unable to modify them during execution.
However, in specific cases, the operating system may assign a task a higher priority as part of the Resource Management: Priority Ceiling Protocol.
Tasks with the same priority level are activated based on their order of arrival, allowing subsequent tasks of identical priority to start even if there are extended tasks in the waiting state.
When a preempted task occurs, it is treated as the first (oldest) task in the ready list of its current priority. Similarly, when a task is released from the waiting state, it is treated as the last (newest) task in the ready queue of its priority.

Scheduling

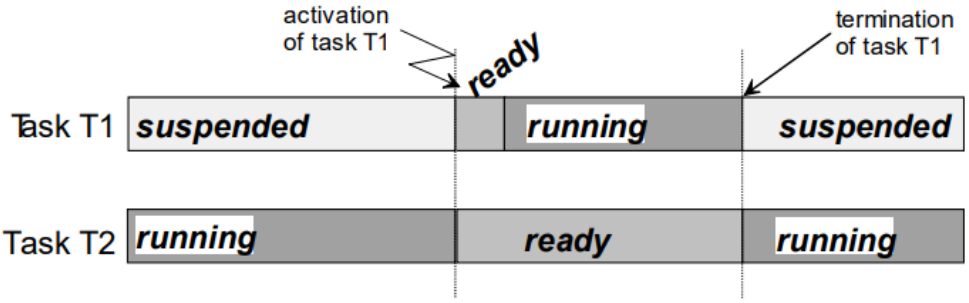


The scheduler determines the next task to transition from the ready state to the running state based on task priority.

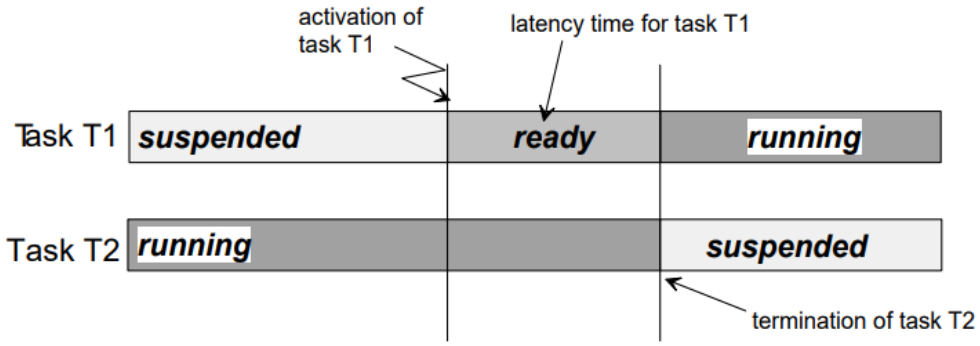
Scheduling Policy

Mixed Preemptive Scheduling
It is a scheduling algorithm that combines features of both preemptive and non-preemptive scheduling. In this case scheduling policy depends on the preemption properties of the running task. If the running task is non preemptable, then non preemptive scheduling is performed. If the running task is preemptable, then preemptive scheduling is performed.

Case 1: Task T2 with the lower priority allow to be preempt from another Task and Task T1 with higher priority.
In this case task T2 does not delay the scheduling of task T1.



Case 2: task T2 with the lower priority doesn't allow to be preempt from another Task and Task T1 with higher priority.
In this case task T2 delays task T1 up to the next point of rescheduling (in this case termination of task T2).



3.2.1.2 Resource Management Module

Resource Concept

Resource management is used to co-ordinate concurrent accesses of several tasks with different priorities to shared resources, management entities (scheduler), program sequences, and memory or hardware areas.
Resource management ensures that:

- Two tasks cannot occupy the same resource at the same time.
- Priority inversion cannot occur.
- Deadlocks do not occur by use of these resources.
- Access to resources never results in a waiting state.

The functionality of resource management is useful in the following cases:

- Preemptable tasks
- Non preemptable tasks, if the user intends to have the application code executed under other scheduling policies, too.

Restrictions when using resources

TerminateTask, ChainTask, Schedule, WaitEvent shall not be called while a resource is occupied.
In case of multiple resource occupation within one task, the user has to request and release resources following the LIFO principle.

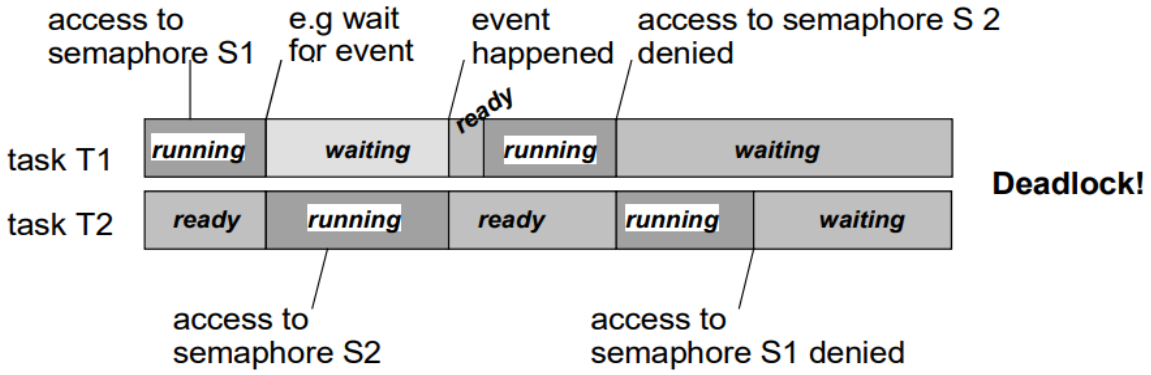
Problems with Synchronization Mechanisms

Priority Inversion

This means that a lower-priority task delays the execution of higher-priority task. The following scenario results in a Priority Inversion:
Task T1 has the highest priority.
Task T4 which has a low priority, occupies the semaphore S1.
T1 preempts T4 and requests the same semaphore. As the semaphore S1 is already occupied, T1 enters the waiting state.
Now the low-priority T4 is interrupted and preempted by tasks with a priority between those of T1 & T4.
T1 can only be executed after all lower-priority tasks have been terminated, and the semaphore S1 has been released again.
Although T2 and T3 do not use semaphore S1, they delay T1 with their runtime.

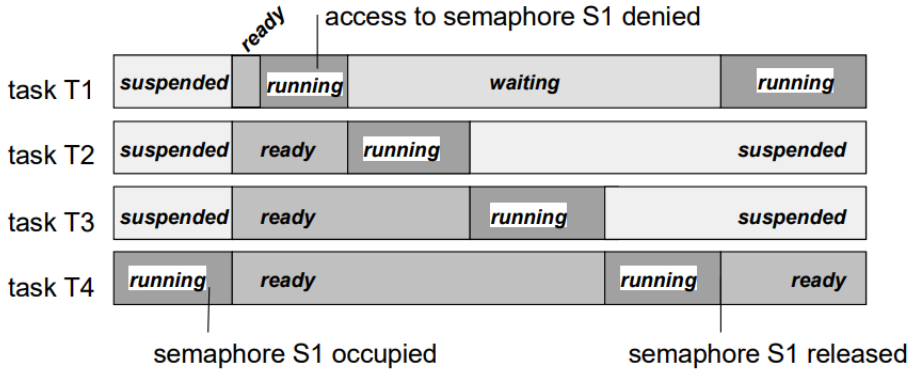
Deadlocks

It means the impossibility of task execution due to infinite waiting for mutually locked resources.
The following scenario results in a deadlock:
Task T1 occupies the semaphore S1 and subsequently cannot continue running, e.g. because it is waiting for an event.
The lower-priority task T2 is transferred into the running state. It occupies the semaphore S2.
If T1 gets ready again and tries to occupy semaphore S2, it enters the waiting state again.
If now T2 tries to occupy semaphore S1, this results in a deadlock.



Priority Ceiling Protocol

To avoid the problems of priority inversion and deadlocks the OSEK operating system requires

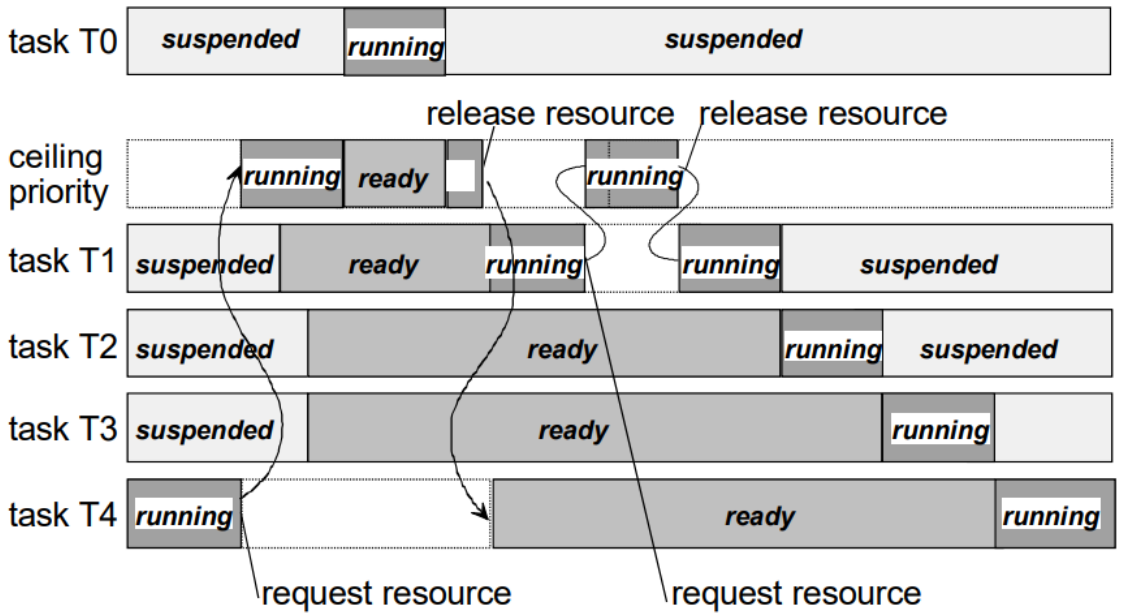


following behavior:

- At the system generation, to each resource its own ceiling priority is statically assigned. The ceiling priority shall be set at least to the highest priority of all tasks that access a resource or any of the resources linked to this resource.
- If a task requires a resource, and its current priority is lower than the ceiling priority of the resource, the priority of the task is raised to the ceiling priority of the resource.
- If the task releases the resource, the priority of this task is reset to the priority which was dynamically assigned before requiring that resource.

Priority ceiling results in a possible time delay for tasks with priorities equal or below the resource priority.

This delay is limited by the maximum time the resource is occupied by any lower priority task. Tasks which might occupy the same resource as the running task do not enter the running state, due to their lower or equal priority than the running task.
If a resource occupied by a task is released, other task which might occupy the resource can enter the running state. For preemptable tasks this is a point of rescheduling.
The mechanism of the priority ceiling:
Task T0 has the highest, and task T4 the lowest priority.
Task T1 and task T4 want to access the same resource.
The system shows clearly that no unbounded priority inversion is entailed. The high-priority task T1 waits for a shorter time than the maximum duration of resource occupation by T4.



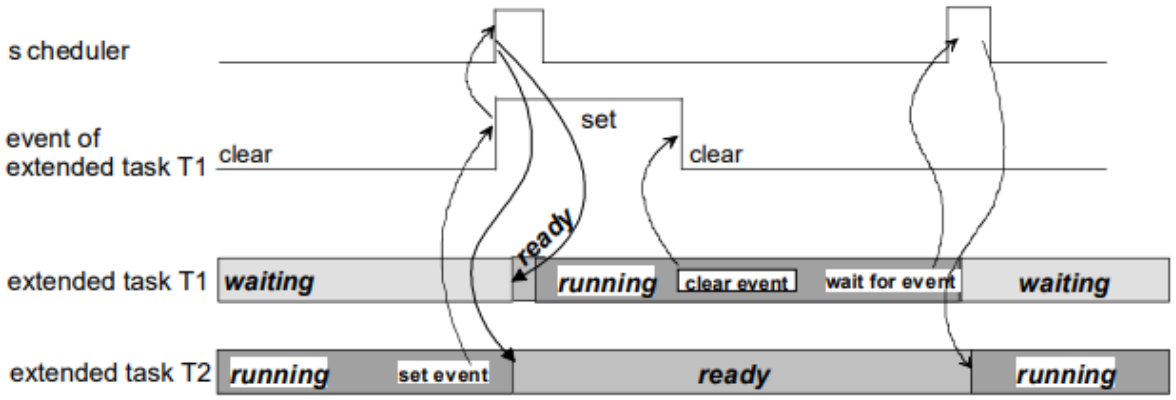
3.2.1.3 Event Mechanism Module

The primary purpose of events within OSEK/VDX is synchronization between tasks. For example, a task can begin a series of activities by activating a set of tasks and then wait for a set of events to occur.
When each activity is completed, it sets an event on which the activating task is waiting. This releases the original task from the waiting mode and allows it to continue with the knowledge that everything has been completed properly.
Events are objects managed by the operating system. They are not independent objects but assigned to extended tasks. Each extended task has a definite number of events. This task is called the owner of these events. An individual event is identified by its owner and its name (or mask).

When activating an extended task, these events are cleared by the operating system.

Events are the criteria for the transition of extended tasks from the waiting state into the ready state. The operating system provides services for setting, clearing and interrogation of events and for waiting for events to occur.

The receiver of an event is an extended task in any case. Consequently, it is not possible for a basic task to wait for an event.
An event can only be cleared by the task which is the owner of the event. Extended tasks may only clear events they own, whereas basic tasks are not allowed to use the operating system service for clearing events.
An extended task in the waiting state is released to the ready state if at least one event for which the task is waiting for has occurred. If a running extended task tries to wait for an event and this event has already occurred, the task remains in the running state.



Synchronization of preemptable extended tasks:
T1 has the higher priority and waits for an event. Task T2 sets this event for T1.
The scheduler is activated. Subsequently, T1 is transferred from the waiting state into the ready state. Due to the higher priority of T1 this results in a task switch, T2 being preempted by T1.
T1 resets the event. Thereafter T1 waits for this event again and the scheduler continues execution of T2

3.2.1.4 Alarms Management Module

The OSEK operating system provides services for processing recurring events. Such events may be for example timers that provide an interrupt at regular intervals, or encoders at axles that generate an interrupt in case of a constant change of a (camshaft or crankshaft) angle, or other regular application specific triggers.
The OSEK operating system provides a two-stage concept to process such events. The recurring events (sources) are registered by implementation specific counters. Based on counters, the OSEK operating system software offers alarm mechanisms to the application software.

3.2.1.5 Counters Module

A counter is represented by a counter value, measured in "ticks", and some counter specific constants.
The OSEK operating system does not provide a standardised API to manipulate counters directly.
The OSEK operating system takes care of the necessary actions of managing alarms when a counter is advanced and how the counter is advanced.
The OSEK operating system offers at least one counter that is derived from a (hardware or software) timer.

3.2.1.6 Bridge Module

Host Core can communicate with HSM core by a module exists in between them called Bridge. It has a window such as cache which can store up to 64 KB of a specific addresses in Host core. We chose the Bridge module as a gate of information between Host & HSM cores because there is a FIREWALL after it which checks and authenticates the data transferred from Bridge module.
It has 2 status registers called HSM2HTS & HT2HSMS, which are responsible for synchronization between 2 cores.
HSM Core can generate 32 interrupts to **Host** core which are mapped onto 2 lines INT0 & INT1. Host core can also generate interrupt to HSM core if and only if HSM core has enabled that line.

3.2.1.7 AES Module

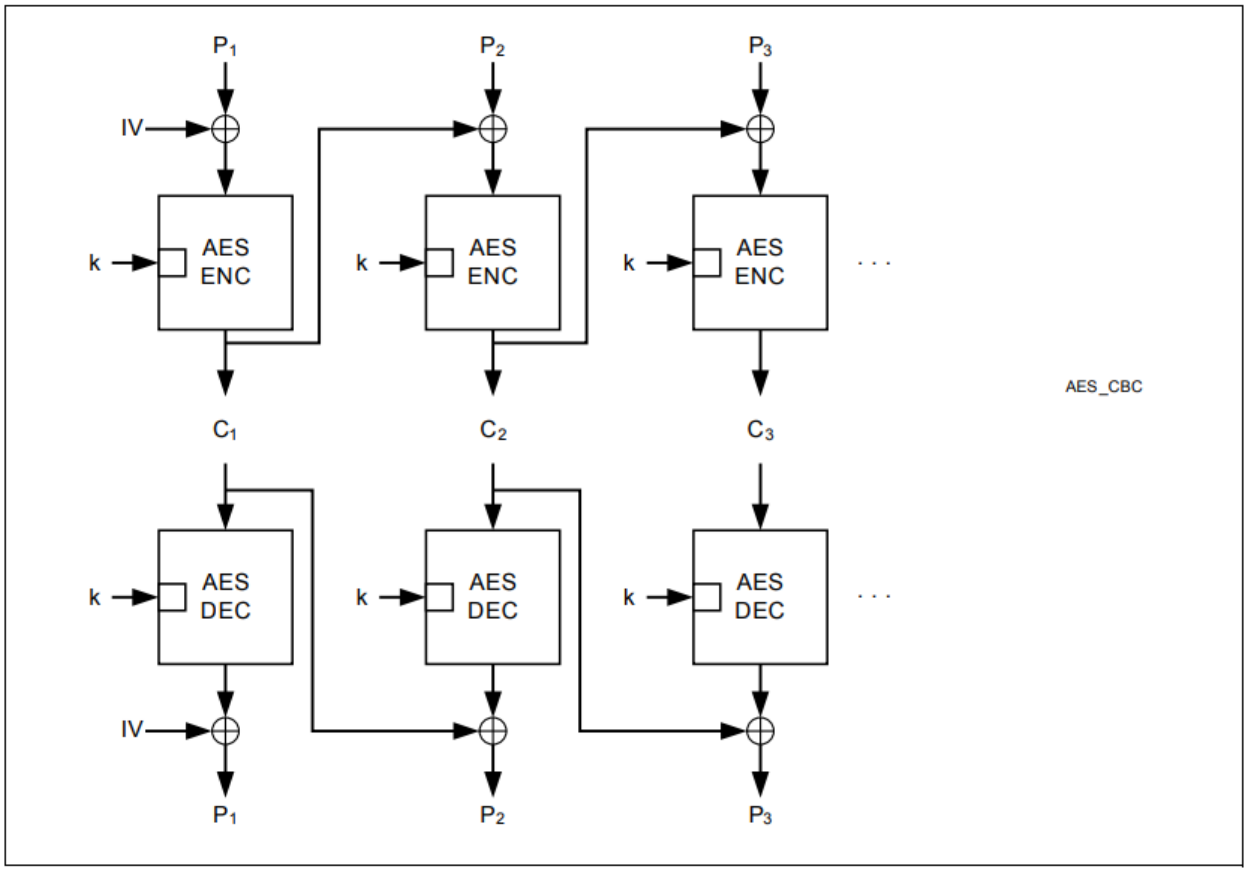
AES, which stands for "advanced encryption system," is one of the most prevalently used types of encryption algorithms and was developed as an alternative to the DES algorithm. Also known as Rijndael, AES became an encryption standard on approval by NIST in 2001. Unlike DES, AES is a family of block ciphers that consists of ciphers of different key lengths and block sizes. AES works on the methods of substitution and permutation. First, the plaintext data is turned into blocks, and then the encryption is applied using the encryption key. The encryption process consists of various sub-processes such as sub bytes, shift rows, mix columns, and add round keys. Depending upon the size of the key, 10, 12, or 14 such rounds are performed. It's worth noting that the last round doesn't include the sub-process of mix columns among all other sub-processes performed to encrypt the data.

The Advantage of Using the AES Encryption Algorithm

What all of this boils down to is to say that AES is safe, fast, and flexible. AES is a much quicker algorithm compared to DES. The multiple key length options are the biggest advantage you have as the longer the keys are, the harder it is to crack them.
Today, AES is the most widely used encryption algorithm — it's used in many applications, including:

- Wireless security,
- Processor security and file encryption,
- SSL/TLS protocol (website security),
- Wi-Fi security,
- Mobile app encryption,
- Most VPNs (virtual private network), etc.

One of the best encryption algorithms is CBC mode:



3.2.1.8 HASH Module

Hardware Security Modules (HSMs) support various hashing algorithms for secure cryptographic operations. Here are some commonly used hashing algorithms in HSMs:

1. SHA-1 (Secure Hash Algorithm 1): SHA-1 is a widely adopted hashing algorithm that produces a 160-bit hash value. However, SHA-1 is considered insecure for cryptographic purposes due to vulnerabilities, and its use is discouraged.

2. SHA-2 (Secure Hash Algorithm 2): SHA-2 is a family of hashing algorithms that includes SHA-224, SHA-256, SHA-384, and SHA-512. These algorithms produce hash values of different lengths: 224, 256, 384, and 512 bits, respectively. SHA-256, in particular, is widely used and considered secure for various cryptographic applications.

3. SHA-3 (Secure Hash Algorithm 3): SHA-3 is a newer family of hashing algorithms that was selected as the winner of the NIST hash function competition in 2012. It includes SHA3-224, SHA3-256, SHA3-384, and SHA3-512, which produce hash values of different lengths. SHA-3 is designed to provide improved security and resistance against certain types of attacks.

4. MD5 (Message Digest Algorithm 5): MD5 is an older and widely used hashing algorithm that produces a 128-bit hash value. However, MD5 is no longer considered secure for cryptographic purposes due to vulnerabilities, particularly collision attacks.

It's important to note that the choice of hashing algorithm in an HSM depends on the specific requirements of the cryptographic operation and the desired level of security. While SHA-2 and SHA-3 algorithms are generally recommended for their stronger security properties, the selection may also be influenced by industry standards, compatibility with existing systems, and regulatory compliance.

HSMs provide a secure and trusted environment for executing these hashing algorithms, ensuring the confidentiality and integrity of the data and cryptographic operations. The hardware-based security mechanisms in HSMs protect against various attacks, including tampering, side-channel attacks, and unauthorized access to cryptographic keys, enhancing the overall security of the hashing operations performed within the module.

What Are the Types of Ciphers Being Used?

Two types of ciphers can be used in symmetric algorithms. These two types are:

- Stream Ciphers
- Block Ciphers

Block Ciphers

On the other hand, block ciphers dissect the raw information into chunks of data of a fixed size. The size depends on the exact cipher being used. A 128-bit block cipher will break the plaintext into blocks of 128-bit each and encrypt those blocks instead of a single digit. These ciphers are slower but much more tamper-proof and are used in some of the most common algorithms being employed today.

3.2.2 Nonfunctional Requirements

3.2.2.1 Performance

Our system is designed to provide high performance and low latency, to ensure that requests are executed quickly and efficiently.

3.2.2.2 Usability

Out system is designed to be easy to use and understand, with clear and intuitive interfaces and documentation.

3.2.2.3 Security

Our system is designed to provide a secure mechanism for managing and executing requests that come from the hosts, including features such as encryption, authentication, and access control.

3.2.2.4 Reliability

Our system is designed to be reliable and robust, with built-in mechanisms for error detection and recovery.

3.2.2.5 Scalability

Our system is designed to be scalable, to support a growing number of users and requests over time.

3.2.2.6 Compliance

Our system is designed to comply with relevant industry standards and regulations, including OSEK/VDX, AUTOSAR, and other relevant standards and best practices.

3.2.2.7 Availability

Our system is designed to be highly available, with built-in mechanisms for fault tolerance and disaster recovery.

3.2.2.8 Interoperability

Our system is designed to be interoperable with other systems and components, to ensure that it can integrate seamlessly with other parts of the car and the larger ecosystem.

3.2.2.9 Maintainability

Our system is designed to be easy to maintain and update, with clear documentation, modular code, and well-defined interfaces.

3.3 Functional Requirements Specification

This section describes specific features of the software project. If desired, some requirements may be specified in the use-case format and listed in the Use Cases Section.

3.3.1 Stakeholders

The stakeholders of this project are:

1) OS developer
2) HSM developer

3) Customers
4) Original equipment manufacturer (OEM)

3.3.2 Actors and Goals

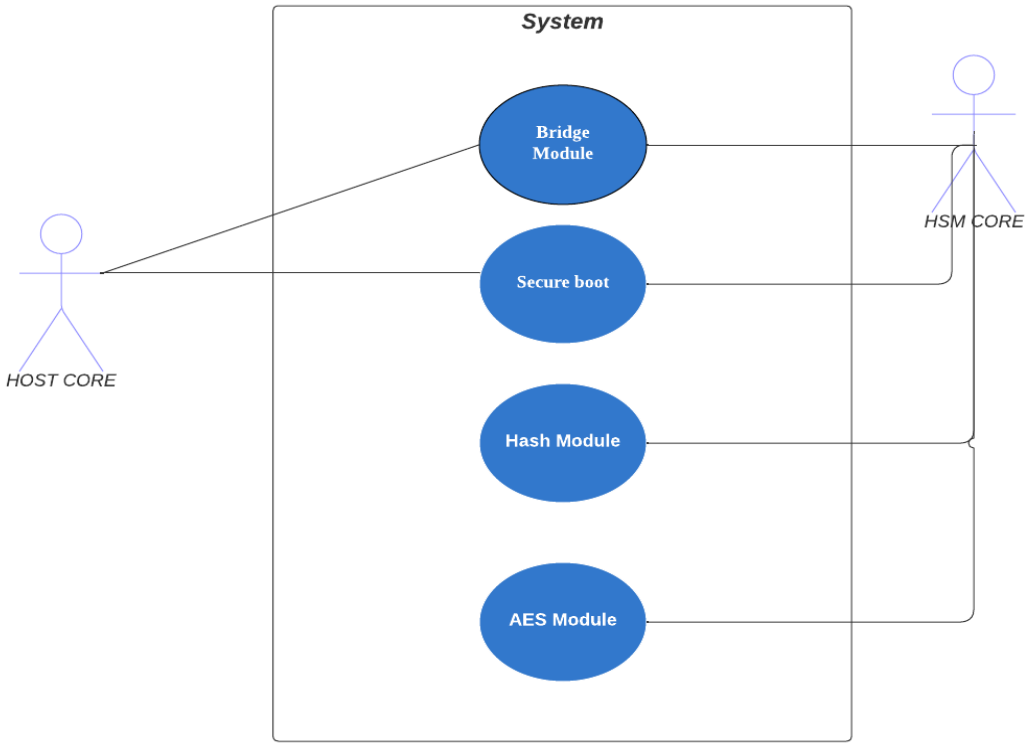
1) OS developer: who will provide the Operating System
2) HSM developer: who will be provided the OS
3) Customers: the host developers using the HSM to serve their needs on other cores and ECUs.
4) Original equipment manufacturer (OEM): Embedding the ECUs that include the HSM into their Electronic Vehicles.

3.3.3 Use Cases

A) Use Case Description

- Host core can communicate HSM core by bridge module function (bidirectional communication)
- Host and HSM can secure boot themselves before start up, while HSM core responsible for Hashing, Encryption and Decryption operations.

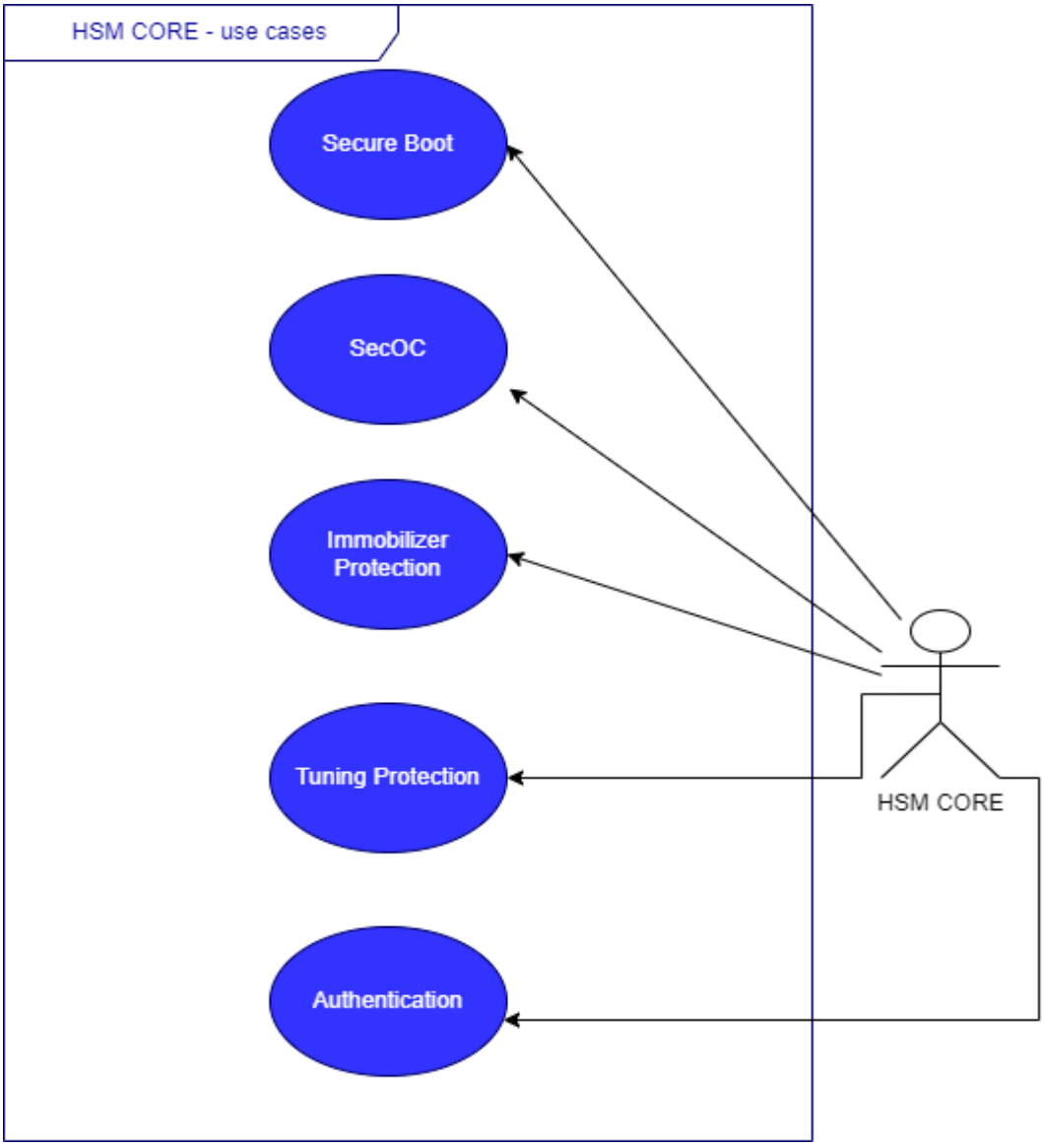
A) Use Case Diagram



B) Use Case Description

- HSM can provide secure communication by encrypting and decrypting the data exchanged over these channels.
- The HSM is responsible for verifying the authenticity and integrity of the software running on the ECU.
- The HSM can be used to verify the authenticity and integrity of software updates that are installed on the ECU.

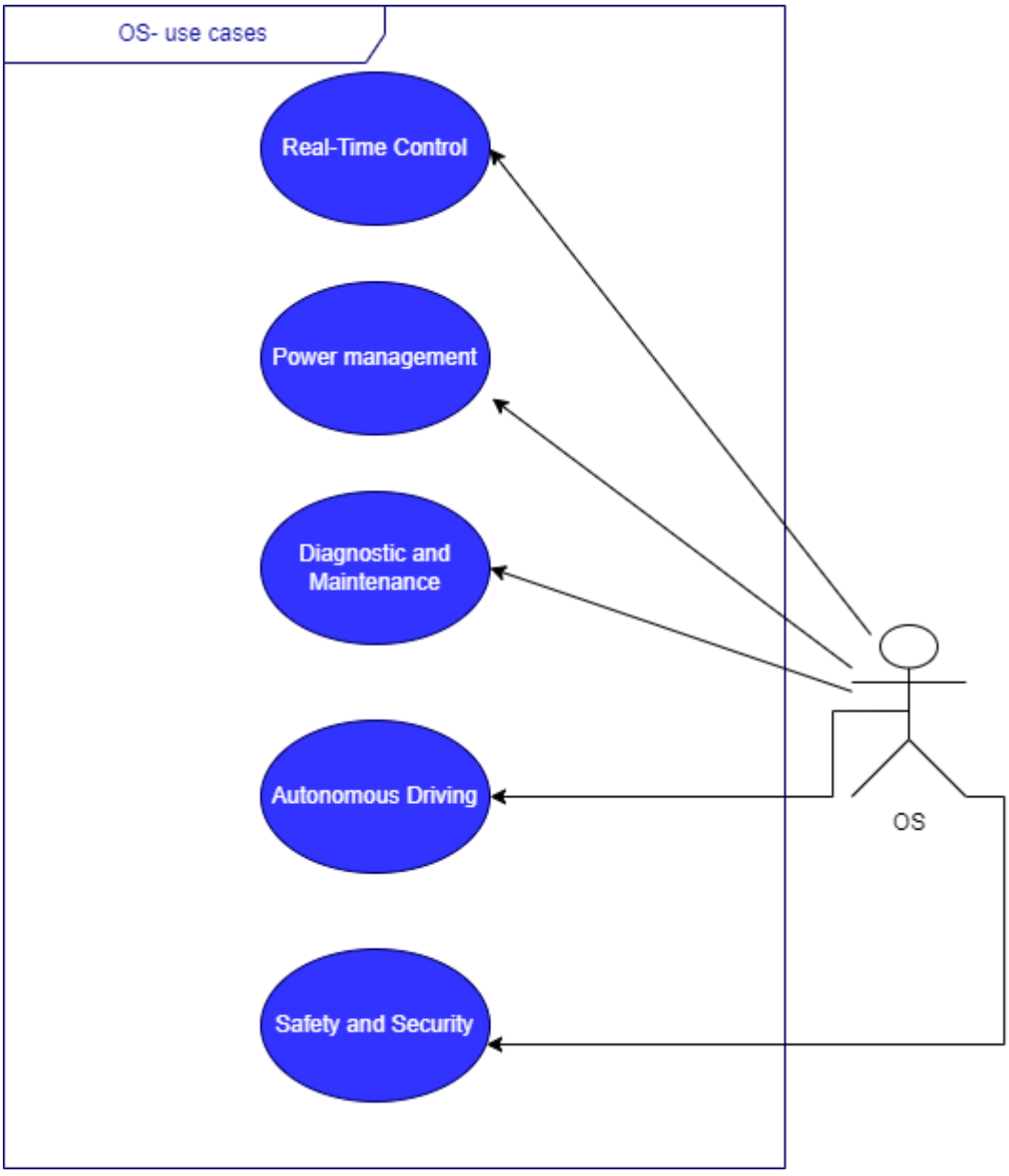
B) Use Case Diagram



C) Use Case Description

- OSEK-compliant operating systems are designed to provide real-time control of embedded systems. This can enable the ECU to respond quickly and accurately to changes in the environment or user input.
- An OSEK-compliant operating system can manage the power consumption of various components inside the ECU.
- An OSEK-compliant operating systems can provide diagnostic and maintenance functions for the ECU.

C) Use Case Diagram



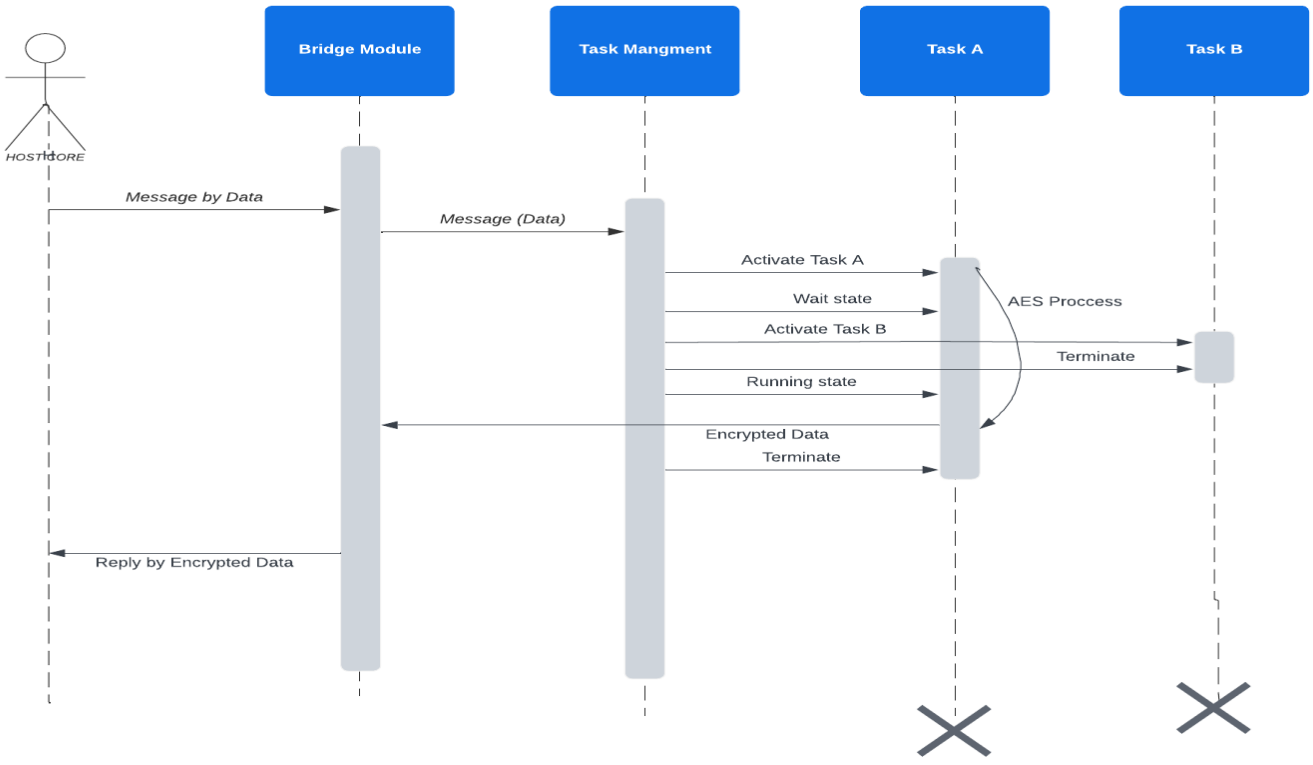
4. Design

4.1 Sequence Diagrams

4.1.1 Task Management

The following diagram explain the role of Task Management in managing the priorities of tasks and preemption between them. Task A is preemptive task that is in RUNNING state, as it is waiting for an Even then it leaves the CPU and changes its state to WAITING state. Task B takes the CPU and become RUNNING, after it terminate and the Event is set, Task A changes to RUNNING state again

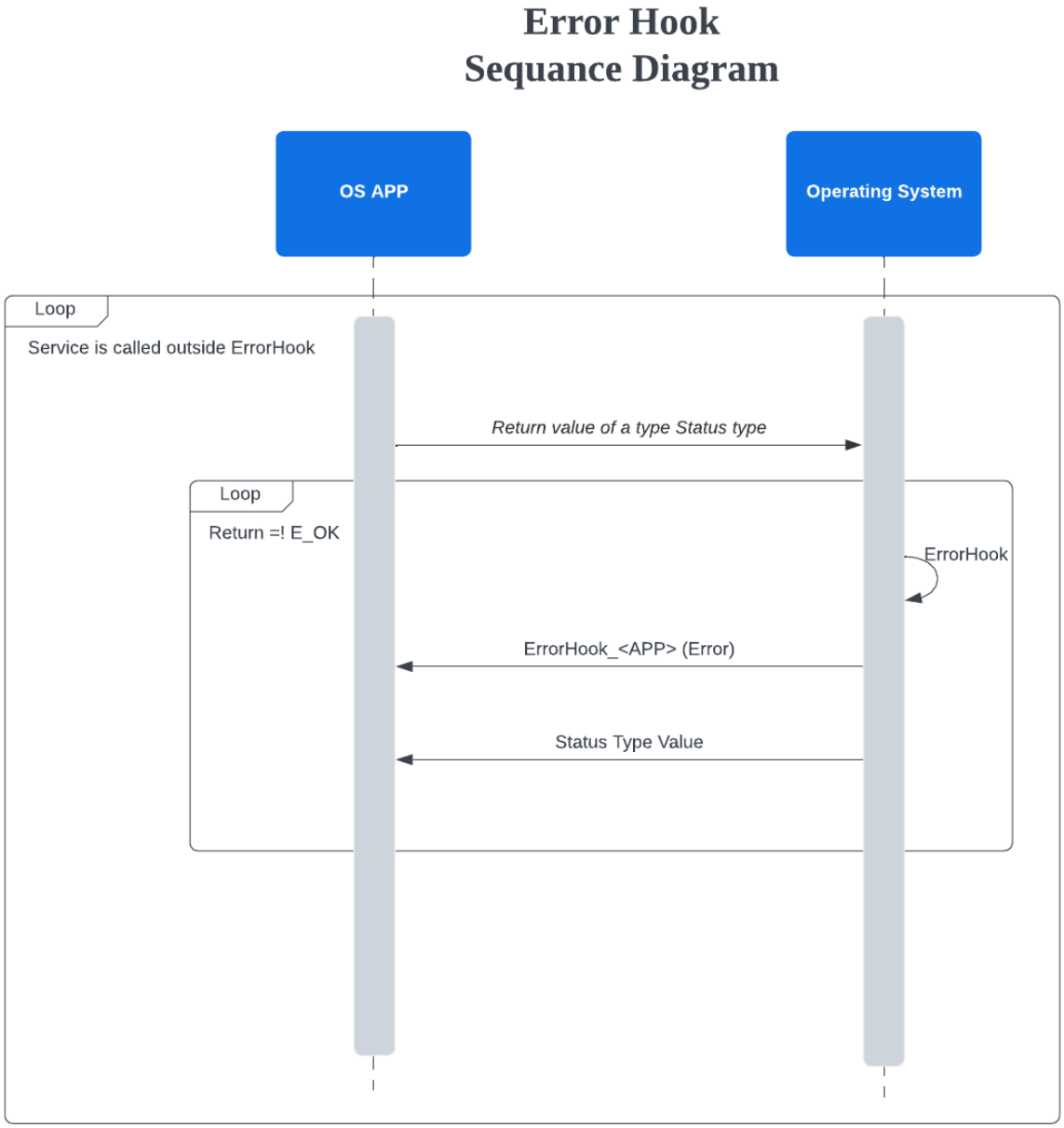
Task Management Sequence Diagram



4.1.2 ErrorHook

The following diagram illustrate the ErrorHook function, when error happened from the user the ErrorHook alert the user by the Status type of this error and its conditions when the error happen outside the ErrorHook and when return Status type is not E_OK

ErrorHook Sequence Diagram



4.2 Interface Specification

4.2.1 Data types

TaskType

This data type identifies a task.

TaskRefType

This data type points to a variable of TaskType.

TaskStateType

This data type identifies the state of a task.

TaskStateRefType

This data type points to a variable of the data type TaskStateType.

ResourceType

Data type for a resource.

EventMaskType

Data type of the event mask.

EventMaskRefType

Reference to an event mask.

TickType

This data type represents count values in ticks.

TickRefType

This data type points to the data type TickType.

AlarmBaseType

This data type represents a structure for storage of counter characteristics. The individual elements of the structure are:

maxallowedvalue	• Maximum possible allowed count value in ticks
ticksperbase	• Number of ticks required to reach a counter-specific (significant) unit.
mincycle	• Smallest allowed value for the cycle-parameter of SetRelAlarm/SetAbsAlarm) (only for systems with extended status).

All elements of the structure are of data type TickType.

AlarmBaseRefType

This data type points to the data type AlarmBaseType.

AlarmType

This data type represents an alarm object.

4.2.2 System Services Description

Prototype

Interface in C-like syntax

Inputs

List of all input parameters

Outputs

List of all output parameters

Return

List all possible return values
List the description of each |
Mention if the status is standard or extended

Status	Description	S	E

Function

Description: Explanation of the functionality of the operating system service.

Particularities: Explanation of restrictions relating to the utilization of the operating system service

Hook Routines Allowed

Error	Startup	Shutdown	PreTask	PostTask

4.2.3 ActivateTask

Prototype

StatusType ActivateTask(TaskType taskID);

Inputs

taskID: Name of the task that is to be activated.

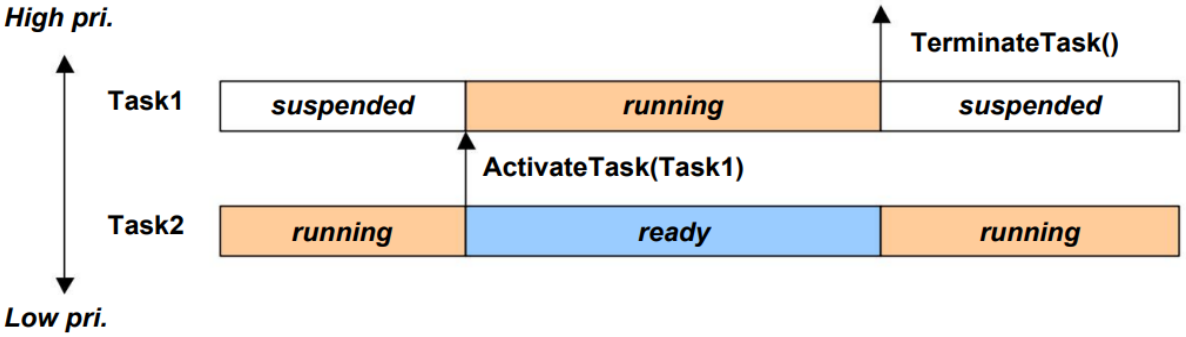
Outputs

None

Return

Status	Description	S	E
E_OS_ID	Input taskID is not a valid task.		✓
E_OS_LIMIT	Too many activations of taskID task. Only applicable in XCC2 conformance classes. Activation is ignored.		✓
E_OK	Service executed without error.	✓	✓

Function



This API service moves the task identified by the input taskID from the SUSPENDED state to the READY state. If the service is invoked from a preemptive task when the RES_SCHEDULER resource is not locked, the scheduler is entered, and preemption occurs if the activated task is a higher priority than the invoking task. If the service is invoked from a non-preemptive task from the ISR level or from within a critical section in which RES_SCHEDULER is locked, the scheduler will not be invoked, and preemption will not occur.

Hook Routines Allowed

Error	Startup	Shutdown	PreTask	PostTask
	✓			

4.2.4 TerminateTask

Prototype

StatusType TerminateTask(void);

Inputs

None

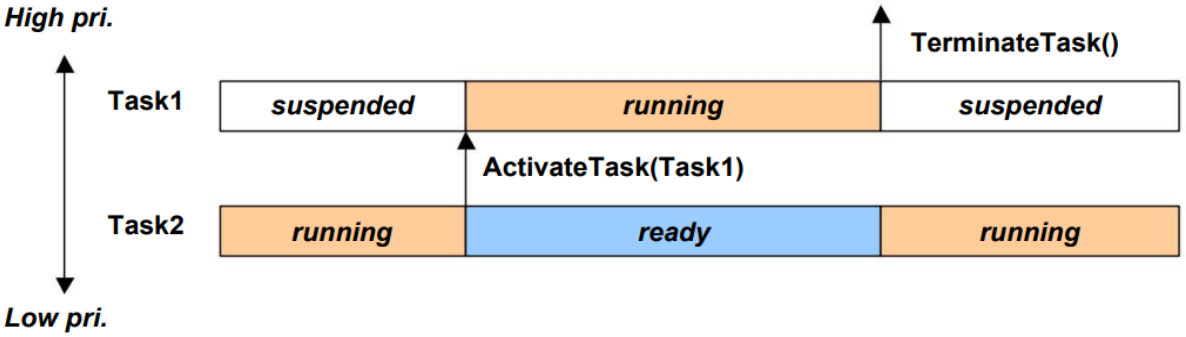
Outputs

None

Return

Status	Description	S	E
E_OS_RESOURCE	Invoking task still occupies resources and termination is not safe.		✓
E_OS_CALLEVEL	Service was invoked from the interrupt level.		✓
E_OK	Service executed without error.	✓	✓

Function



This function terminates the task that invoked the service, transferring it from the RUNNING to the SUSPENDED state. The service then forces a rescheduling of the application. All resources locked by the invoking task must be released using ReleaseResource() prior to invoking this service. This service only returns in the extended status state when an error occurs. All tasks must end with either TerminateTask() or ChainTask().

Hook Routines Allowed

Error	Startup	Shutdown	PreTask	PostTask

4.2.5 ChainTask

Prototype

StatusType ChainTask(TaskType taskID);

Inputs

taskID: Name of the task that is to be activated.

Outputs

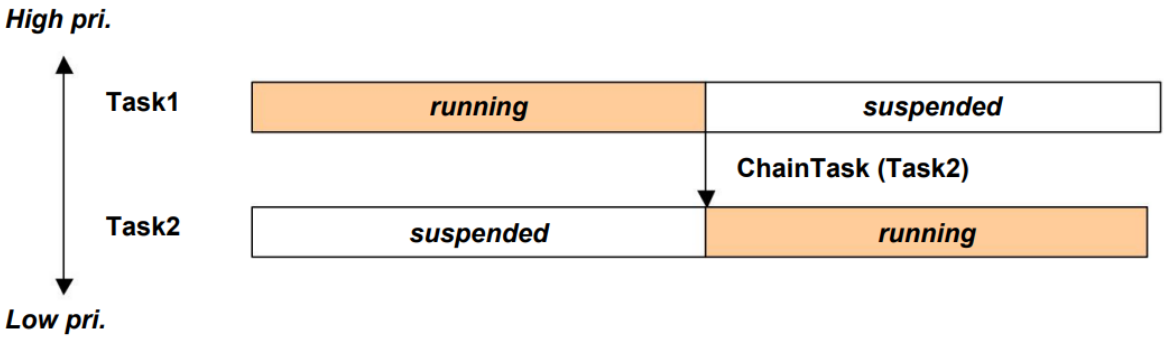
None

Return

Status	Description	S	E
E_OS_ID	Input taskID is not a valid task.		✓
E_OS_LIMIT	Too many activations of taskID task. Only applicable in XCC2 conformance classes. Activation is ignored.		✓
E_OS_RESOURCE	Invoking task still occupies resources and termination is not safe.		✓
E_OS_CALLEVEL	Service was invoked from the interrupt level.		✓
E_OK	Service executed without error.	✓	✓

Function

This service terminates the invoking task, moves the task identified by the input taskID from the SUSPENDED state to the READY state, and reschedules the application.
Because the invoking task terminates first, invoking this service with taskID set to the same ID as the invoking task does not cause multiple activations.
All resources locked by the invoking task must be released using ReleaseResource() prior to invoke this service.
This service only returns in the extended status state when an error occurs.All tasks must end with either ChainTask() or TerminateTask() APIs.



Hook Routines Allowed

Error	Startup	Shutdown	PreTask	PostTask

4.2.6 Schedule

Prototype

StatusType Schedule(void);

Inputs

None

Outputs

None

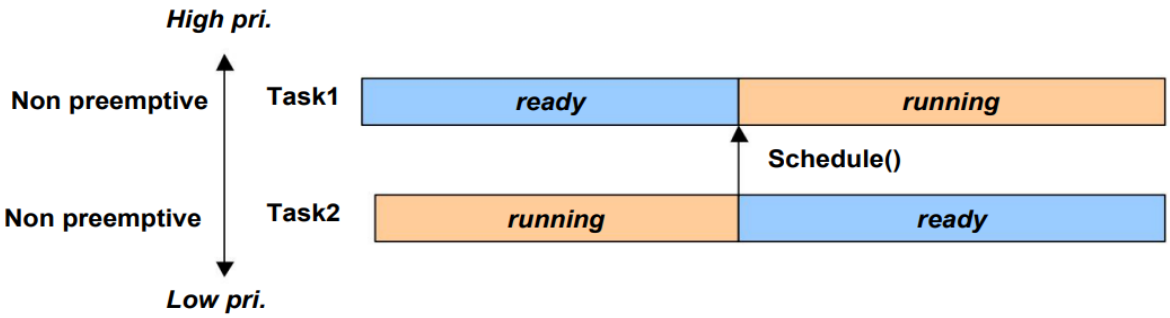
Return

Status	Description	S	E
E_OS_CALLEVEL	Service was invoked from the interrupt level.		✓
E_OK	Service executed without error.	✓	✓

Function

This service is used in non-preemptive tasks to allow cooperative multitasking by forcing the scheduler to run a higher priority task if there is one in the READY state. Although it is possible to call this service from a full-preemptive task, no action is taken.

Hook Routines Allowed



Error	Startup	Shutdown	PreTask	PostTask

4.2.7 GetTaskID

Prototype

StatusType GetTaskID(TaskRefType taskIDRef);

Inputs

None

Outputs

taskIDRef: Reference to a variable of type TaskType that contains the identifier of the task that is currently running.
If no task is running, the variable is set to INVALID_TASK.

Return

Status	Description	S	E
E_OK	Service executed without error.	✓	✓

Function

This service provides the application with the identifier of the task that is presently running. It is intended to be used in hook routines and library functions to check the task from which it was invoked

Hook Routines Allowed

Error	Startup	Shutdown	PreTask	PostTask
✓			✓	✓

4.2.8 GetTaskState

Prototype

StatusType GetTaskState(TaskType taskID, TaskStateRefType stateRef);

Inputs

taskID: Name of the task that is to be checked.

Outputs

stateRef: Reference to a variable of type TaskStateType into which the service will place the current state of the task identified by the input taskID. The value will be one of the following constants: RUNNING, WAITING, READY, SUSPENDED

Return

Status	Description	S	E
E_OS_ID	Input taskID is not a valid task.		✓
E_OK	Service executed without error.	✓	✓

Function

This service identifies the current state of the task identified by the taskID parameter and places this value into the variable referenced by stateRef. If this service is invoked by a task that can be preempted, the result could be invalid by the time it is evaluated. In this case, it is recommended that interrupts are disabled prior to invoking the service and until the result is analyzed.

Hook Routines Allowed

Error	Startup	Shutdown	PreTask	PostTask
✓			✓	✓

4.2.9 GetResource

Prototype

StatusType GetResource(ResourceType resID);

Inputs

resID: Name of the resource that is to be locked.

Outputs

None

Return

Status	Description	S	E
E_OS_ID	Requested resource is not a valid resource.		✓
E_OS_ACCESS	The application has attempted to get a resource that is already locked.		✓
E_OK	Service executed without error.	✓	✓

Function

This service allows the application to lock a resource and to enter into a critical section that will disable all other tasks that need access to the resource through the priority ceiling protocol. The corresponding call to ReleaseResource() should appear within the same function. The resource must be released prior to the task being placed in either the SUSPENDED or the WAITING state.

Hook Routines Allowed

Error	Startup	Shutdown	PreTask	PostTask

4.2.10 ReleaseResource

Prototype

StatusType ReleaseResource(ResourceType resID);

Inputs

resID: Name of the resource that is to be unlocked.

Outputs

None

Return

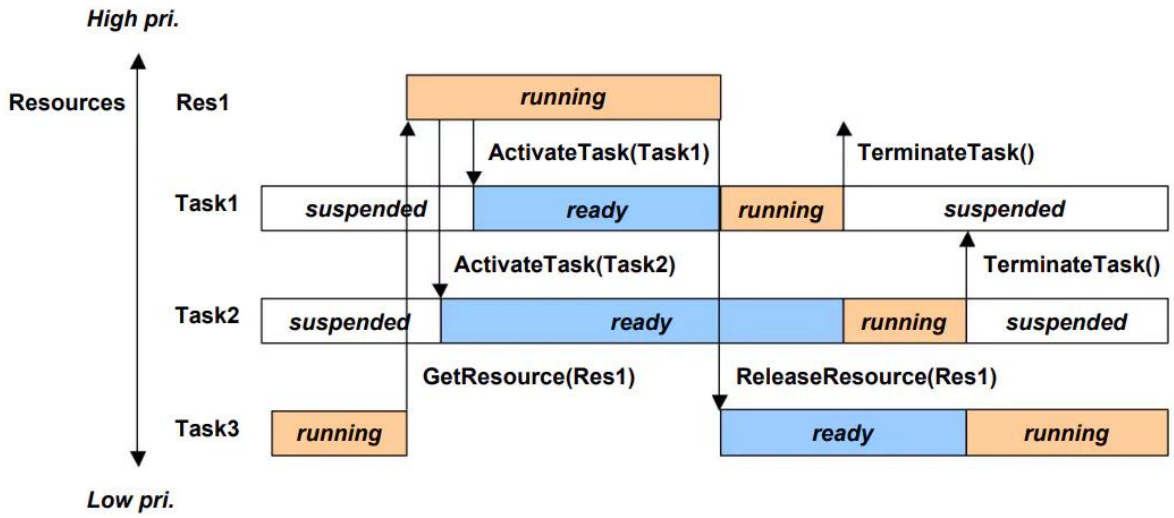
Status	Description	S	E
E_OS_ID	Requested resource is not a valid resource.		✓
E_OS_FUNC	The application has attempted to release a resource that is not locked or another resource has to be released first.		✓
E_OS_ACCESS	The application has attempted to release a resource that has a lower ceiling priority than the statically assigned priority of the task or ISR that invoked the service.		✓
E_OK	Service executed without error.	✓	✓

Function

This service allows the application to unlock a resource and to leave a critical section.
The priority ceiling protocol returns the task that contains the critical section to the statically assigned priority.
The corresponding call to GetResource() should appear within the same function. The resource must be released prior to the task being placed in either the SUSPENDED or the WAITING state.

Hook Routines Allowed

Error	Startup	Shutdown	PreTask	PostTask



4.2.11 SetEvent

Prototype

StatusType SetEvent(TaskType taskID, EventMaskType mask);

Inputs

taskID: Name of the task that owns the event(s).

mask: Event mask of the events that are to be set.

Outputs

None

Return

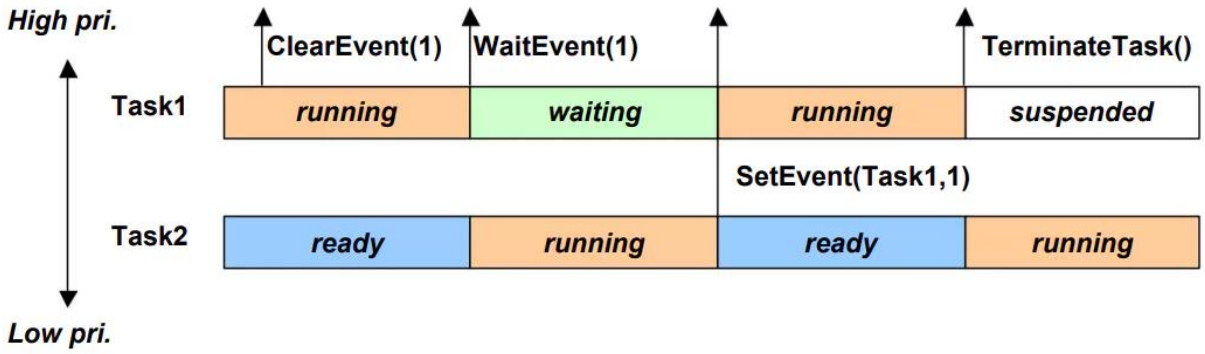
Status	Description	S	E
E_OS_ID	The task is not a valid task.		✓
E_OS_ACCESS	The task is not an extended task.		✓
E_ OS_STATE	The task is in the SUSPENDED state. Events cannot be set for tasks in the SUSPENDED state.		✓
E_OK	Service executed without error.	✓	✓

Function

The events defined by the input mask are set for the task defined by the input taskID.
Multiple events can be set for a given task with one call to this service.
If the task is in the WAITING state, and is waiting on one of the events that is set, the OS will move the task into the READY state.
If the task is in the READY state, the event is set but no action occurs for that task until it runs again and invokes WaitEvent() with one of these events set.
At that time, the task does not enter the WAITING state because the event is already set.

Hook Routines Allowed

Error	Startup	Shutdown	PreTask	PostTask



4.2.12 ClearEvent

Prototype

StatusType ClearEvent(EventMaskType mask);

Inputs

mask: Event mask of the events that are to be cleared.

Outputs

None

Return

Status	Description	S	E
E_OS_ACCESS	Service was not invoked from an extended task.		✓
E_OS_CALLEVEL	Service was invoked on the interrupt level.		✓
E_OK	Service executed without error.	✓	✓

Function

This service clears all of the events based on the events that are included in the input mask. This service can only be invoked from the extended task that owns the events. Typically, this service will be invoked after the task returns from the WAITING state and has processed a particular event or set of events.

Hook Routines Allowed

Error	Startup	Shutdown	PreTask	PostTask

4.2.13 WaitEvent

Prototype

StatusType WaitEvent(EventMaskType mask);

Inputs

mask: Event mask of the events to be waited for.

Outputs

None

Return

Status	Description	S	E
E_OS_ACCESS	Service was not invoked from an extended task.		✓
E_OS_RESOURCE	The extended task from which the service was invoked still has resources locked.		
E_OS_CALLEVEL	Service was invoked on the interrupt level.		✓
E_OK	Service executed without error.	✓	✓

Function

This service checks the status of all of the events defined by the input value mask. If any of the events are set, the service immediately returns; otherwise, the service puts the extended task into the WAITING state and invokes the scheduler. The service can only be invoked from the extended task that owns the events.

Hook Routines Allowed

Error	Startup	Shutdown	PreTask	PostTask

4.2.14 GetEvent

Prototype

StatusType GetEvent(TaskType taskID, EventMaskRefType maskRef);

Inputs

taskID: Name of the extended task that is to being queried.

Outputs

maskRef: Reference to a variable of type EventMaskType into which the current status of the events for the task is placed.

Return

Status	Description	S	E
E_OS_ID	The task is not a valid task.		✓
E_OS_ACCESS	The task is not an extended task.		✓
E_OS_STATE	The task is in the SUSPENDED state. Events are not valid for tasks in the SUSPENDED state.		✓
E_OK	Service executed without error.	✓	✓

Function

The current status of all events for the task defined by the input taskID are placed in the variable referenced by the eventRef input. The value placed here is the current state, either set or cleared, and does not indicate whether the task is waiting for the event.

Hook Routines Allowed

Error	Startup	Shutdown	PreTask	PostTask
✓			✓	✓

4.2.15 SetRelAlarm

Prototype

StatusType SetRelAlarm(AlarmType alarmID, TickType increment, TickType cycle);

Inputs

alarmID: Name of the alarm that is to be set.
increment: The incremental value in counter ticks relative to the current counter value at which the alarm is to expire the first time.
cycle: If this input is not 0, then the alarm is a cyclic alarm with cycle ticks.

Outputs

None

Return

Status	Description	S	E
E_OS_ID	The alarm is not a valid alarm.		✓
E_OS_VALUE	Either the value of the input increment is less than 0 or greater than the alarm base value maxallowedvalue, or the input cycle is less than mincycle or greater than maxallowedvalue.		✓
E_OS_CALLEVEL	The alarm has previously been set and has not expired prior to a second attempt to set the alarm.		✓
E_OK	Service executed without error.	✓	✓

Function

This service sets an alarm to expire at an incremental value of the counter to which it is assigned. When the counter reaches the value defined by the input increment plus the current value of the counter, the alarm expires. Typically, this service is used for a time-based alarm to set the next time that an action, such as a periodic task, would occur. A cyclic alarm is defined if the input cycle is not equal to 0. When the alarm expires, the value of the input cycle is added to the current alarm value and used as the next set point at which to expire. If the alarm is currently in use, this service will fail. To restart an alarm that is currently in use, first invoke CancelAlarm().

Hook Routines Allowed

Error	Startup	Shutdown	PreTask	PostTask

4.2.16 SetAbsAlarm

Prototype

StatusType SetAbsAlarm(AlarmType alarmID, TickType start, TickType cycle);

Inputs

alarmID: Name of the alarm that is to be set.
start: The absolute value in counter ticks at which the alarm is to expire the first time.
cycle: If this input is not 0, then the alarm is a cyclic alarm with cycle ticks.

Outputs

None

Return

Status	Description	S	E
E_OS_ID	The alarm is not a valid alarm.		✓
E_OS_VALUE	Either the value of the input increment is less than 0 or greater than the alarm base value maxallowedvalue, or the input cycle is less than mincycle or greater than maxallowedvalue.		✓
E_OS_STATE	The alarm has previously been set and has not expired prior to a second attempt to set the alarm.		✓
E_OK	Service executed without error.	✓	✓

Function

This service sets an alarm to expire at an absolute value of the counter to which it is assigned. When the counter reaches the value defined by the input start, the alarm expires. If the counter has already passed the value defined by start, the alarm expires only after the counter rolls over one time. If the counter is very close to the start value, the counter can expire prior to the OS returning from this service. Typically, this service is not used for a timebased alarm, but only for an alarm based on a position input, such as an engine crank angle. A cyclic alarm is defined if the input cycle is not equal to 0. When the alarm expires, the value of the input cycle is added to the current alarm value and used as the next set point at which to expire. If the alarm is currently in use, this service fails. To restart an alarm that is currently in use, first invoke CancelAlarm().

Hook Routines Allowed

Error	Startup	Shutdown	PreTask	PostTask

4.2.17 GetAlarm

Prototype

StatusType GetAlarm(AlarmType alarmID, TickRefType tickRef);

Inputs

alarmID: Name of the alarm that is to be checked.

Outputs

tickRef: Reference to a variable of type TickType into which the service places the remaining counter ticks before the alarm expires.

Return

Status	Description	S	E
E_OS_ID	The alarm is not a valid alarm.		✓
E_OS_NOFUNC	The alarm is not presently used. This typically indicates that the alarm has not been set or has already expired.	✓	✓
E_OK	Service executed without error.	✓	✓

Function

This service checks the requested alarm to determine if it has been set and has not yet expired. If this is true, it then sets the variable referred to by the parameter tickRef to the number of counter ticks remaining before the alarm expires.

Hook Routines Allowed

Error	Startup	Shutdown	PreTask	PostTask
✓			✓	✓

4.2.18 CancelAlarm

Prototype

StatusType CancelAlarm(AlarmType alarmID);

Inputs

alarmID: Name of the alarm that is to be canceled.

Outputs

None.

Return

Status	Description	S	E
E_OS_ID	Input alarmID is not a valid alarm.		✓
E_OS_NOFUNC	Alarm defined by alarmID is not in use.	✓	✓
E_OK	Service executed without error.	✓	✓

Function

This service cancels an alarm that has been set but has not yet been triggered. If the application wants to restart an alarm that is currently running, it must first invoke this service before it can reset the value of the alarm.

Hook Routines Allowed

Error	Startup	Shutdown	PreTask	PostTask

4.2.19 GetAlarmBase

Prototype

StatusType GetAlarmBase(AlarmType alarmID, AlarmBaseRefType baseRef);

Inputs

alarmID: Name of the alarm that is to be checked.

Outputs

baseRef: Reference to a structure of type AlarmBaseType into which the service places the current characteristics of the alarm identified by the input alarmID.

Return

Status	Description	S	E
E_OS_ID	The alarm is not a valid alarm		✓
E_OK	Service executed without error.	✓	✓

Function

This service sets the values in the structure defined by baseRef to the current characteristics of the alarm. The members of the structure follow. maxallowedvalue The maximum allowed value of the alarm counter in ticks. ticksperbase The number of counter ticks required to reach a counter-specific unit. This value is vaguely defined in the specification and is typically not used. mincycle The smallest allowed value for the cycle parameter when setting the alarm.

Hook Routines Allowed

Error	Startup	Shutdown	PreTask	PostTask
✓			✓	✓

4.2.20 IncrementCounter

Prototype

StatusType IncrementCounter (CounterType counterID);

Inputs

counterID: Name of the Counter to be incremented.

Outputs

None.

Return

Status	Description	S	E
E_OS_ID	The CounterID was not valid or counter is implemented in hardware and can not be incremented by software.		✓
E_OK	Service executed without error.	✓	✓

Function

This service increments a software counter with which a number of alarms is linked to be expired if their ticks end.

Hook Routines Allowed

Error	Startup	Shutdown	PreTask	PostTask
✓			✓	✓

5. Testing and Validation

5.1 Introduction

In the field of automotive engineering, operating systems play a crucial role in ensuring the proper functioning of various critical components and systems in a vehicle. To ensure the reliability, safety, and interoperability of these operating systems, industry standards have been developed to guide their development and implementation. Two such standards are the OSEK (Open System and the Embedded Kernel) and AUTOSAR (AUTomotive Open System ARchitecture) requirement specifications.

The OSEK specification provides a standard for real-time operating systems in embedded systems, including automotive applications. It defines a set of APIs (Application Programming Interfaces) that enable the development of portable and modular embedded applications. On the other hand, AUTOSAR is a standardized software architecture for automotive applications that promotes the reuse of software components and interoperability between different automotive systems.

In this context, the use of OSEK and AUTOSAR requirement specifications is essential when developing an operating system on an HSM (Hardware Security Module) module for automotive applications. These specifications provide a common set of guidelines for the development of reliable and secure operating systems that meet the stringent safety and compatibility requirements of the automotive industry. By using these specifications to guide the development of an API for the operating system, developers can ensure that the resulting system is robust, efficient, and capable of supporting a wide range of automotive applications.

5.2 OSEK Requirements

The requirements in OSEK on which the test cases was implemented:

Task Management

[TM_OSEK_000]	OSEK OS system provides a Fixed Priority Scheduling task switching mechanism.
[TM_OSEK_001]	Tasks on the same priority level are put in a Dynamic FIFO queue for that priority.
[TM_OSEK_002]	OSEK OS system provides an IDLE-mechanism which is active when no other system/application functionality is active.
[TM_OSEK_003]	OSEK OS provides two different task concepts, basic tasks & extended tasks.
[TM_OSEK_004]	Processor releases basic tasks if & only if: 1- It terminates. 2- Is preempted by a higher priority task (switch to another task). 3- An interrupt occurs while interrupts are enabled (switch to ISR). 4- A resource is released.

	5- Calling SetEvent for a waiting ext-task of higher priority. 6- Return from interrupt level to task level.
[TM_OSEK_005]	An Ext-Task is released for the same reasons a basic task may be released as mentioned in [TM_OSEK_004] in addition to that ext-Task calling WaitEvent
[TM_OSEK_006]	A preempted task is considered to be the first (oldest) task in the ready list of its current priority.
[TM_OSEK_007]	Activation of a task can be achieved by the call of ActivateTask or ChainTask API for that task.
[TM_OSEK_008]	A basic task that allows reactivation (determined by a task specific attribute) may be reactivated by itself or by another running system object by calling ActivateTask or ChainTask API for that task.
[TM_OSEK_009]	Extended tasks don't allow reactivations.
[TM_OSEK_010]	Termination of a task is only possible if the task terminates itself directly using either TerminateTask or ChainTask APIs.
[TM_OSEK_011]	The scheduler is activated whenever a task switch is possible as specified in [TM_OSEK_004], [TM_OSEK_005].
[TM_OSEK_012]	Schedular is considered a resource {RES_SCHEDULER} that tasks may occupy to disable preemption and release at the termination point.
[TM_OSEK_013]	The value 0 is defined as the lowest priority of a task. Accordingly bigger numbers define higher priorities
[TM_OSEK_014]	Priority of a task is defined statically. Meaning user can't change it during execution.
[TM_OSEK_015]	In particular cases the OS can treat a task with a defined higher priority {exePri}.
[TM_OSEK_016]	A task being released from the waiting state is treated like the newest task in the ready queue of its priority.
[TM_OSEK_017]	The task context is saved so that the preempted task can be continued at the location where it was preempted.
[TM_OSEK_018]	OSEK OS has the ability to differ between preemptable and non-preemtable tasks (determined by a task specific attribute)
[TM_OSEK_019]	The task type (basic or extended) is independent from the task's scheduling type (preemptable or non preemptable).
[TM_OSEK_020]	ActivateTask API may be called from interrupt level and from task leve
[TM_OSEK_021]	Rescheduling after the call to ActivateTask depends on the place it is called from (ISR, non preemptable task, preemptable task).
[TM_OSEK_022]	If the service TerminateTask is called successfully, it enforces a rescheduling.
[TM_OSEK_023]	Ending a task function must by call to TerminateTask or ChainTask
[TM_OSEK_024]	If ChainTask API is called The task is not transferred to the suspended state, but will immediately become ready again.
[TM_OSEK_025]	If the service ChainTask is called successfully, it enforces a rescheduling.

[TM_OSEK_026]	GetTaskID This service is intended to be used by library functions and hook routines.
[TM_OSEK_027]	In GetTaskID API If <TaskID> can't be evaluated (no task currently running), the service returns INVALID_TASK as TaskType.
[TM_OSEK_028]	In GetTaskState API The service may be called from interrupt service routines, task level, and some hook routines
[TM_OSEK_029]	ActivateTask API force The task <TaskID> is transferred from the suspended state into the ready state
[TM_OSEK_030]	TerminateTask API, This service causes the termination of the calling task. The calling task is transferred from the running state into the suspended state
[TM_OSEK_031]	ChainTask API, This service causes the termination of the calling task. After termination of the calling task a succeeding task <TaskID> is activated
[TM_OSEK_032]	Schedule API, used when a higher-priority task is ready, the internal resource of the task is released, the current task is put into the ready state, its context is saved and the higher-priority task is executed. Otherwise the calling task is continued.
[TM_OSEK_033]	GetTaskID API returns the information about the TaskID of the task which is currently running.
[TM_OSEK_034]	GetTaskState API Returns the state of a task (running, ready, waiting, suspended) at the time of calling GetTaskState

Alarm Management

[AC_OSEK_000]	Counters and alarms are defined statically. The assignment of alarms to counters, as well as the action to be performed when an alarm expires, is defined statically, too.
[AC_OSEK_001]	A counter is represented by a counter value, measured in" ticks", and some counter specific constants.
[AC_OSEK_002]	The OSEK OS does not provide a standardised API to manipulate counters directly.
[AC_OSEK_003]	The OSEK OS manages alarms, when a counter is advanced and how the counter is advanced.
[AC_OSEK_004]	The OSEK OS offers at least one counter that is derived from a (hardware or software) timer.
[AC_OSEK_005]	The OSEK operating system provides services to activate tasks, set events or call an alarm-callback routine when an alarm expires
[AC_OSEK_006]	An alarm-callback routine is a short function provided by the application.
[AC_OSEK_007]	An alarm will expire when a predefined counter value is reached.
[AC_OSEK_008]	Alarms can be defined to be either single alarms or cyclic alarms.
[AC_OSEK_009]	More than one alarm can be attached to a counter.

[AC_OSEK_010]	An alarm is statically assigned at system generation time to: A - one counter B - one task or one alarm-callback routine
[AC_OSEK_011]	Alarm-callback routines run with category 2 interrupts disabled.
[AC_OSEK_012]	Dynamic parameters are the counter value when an alarm shall expire, and the period for cyclic alarms.
[AC_OSEK_013]	Alarm-callback routines can have neither parameter nor return value.
[AC_OSEK_014]	The following format of the alarm-callback prototype shall apply: ALARMCALLBACK(AlarmCallbackRoutineName);
[AC_OSEK_015]	Sharing a task/ISR/alarm between different modes is recommended if the same functionality is needed again.
[AC_OSEK_016]	11Counters are - if possible - set to zero by the system initialisation before alarms are autostarted. Exception: calendar timers etc. For autostarted alarms, all values are relative values.
[AC_OSEK_017]	The value of <Cycle> in case of cyclic alarm. In case of single alarms, cycle shall be zero.
[AC_OSEK_018]	If the relative value <increment> is very small, the alarm may expire, and the task may become ready or the alarm-callback may be called before the system service returns to the user.
[AC_OSEK_019]	The alarm <AlarmID> must not already be in use.
[AC_OSEK_020]	To change values of alarms already in use the alarm shall be cancelled first.
[AC_OSEK_021]	The value of <increment> must be [0-maxallowedvalue].
[AC_OSEK_022]	The value of <cycle> must be [mincyle-maxallowedvalue] but not equal 0.
[AC_OSEK_023]	When <start> ticks are reached, the task assigned to the alarm <AlarmID> is activated or the assigned event (only for extended tasks) is set or the alarm-callback routine is called.
[AC_OSEK_024]	If the absolute value <start> is very close to the current counter value, the alarm may expire, and the task may become ready or the alarm-c
[AC_OSEK_025]	If the absolute value <start> already was reached before the system call, the alarm shall only expire when the absolute value <start> is reached again, i.e. after the next overrun of the counter.
[AC_OSEK_026]	The value of <start> must be [0-maxallowedvalue].

Event Handling

[EV_OSEK_000]	Extended Tasks has have an additional state {waiting state}
[EV_OSEK_001]	The Waiting state may be visited by Ext-Tasks using WaitEvent API
[EV_OSEK_002]	Each extended task has a definite number of events {8 events maximum}
[EV_OSEK_003]	An individual event is identified by its owner and its name (or mask).

[EV_OSEK_004]	When activating an extended task, these events are cleared by the OS.
[EV_OSEK_005]	Only the owner extended task is able to clear its events and to wait for the reception (= setting) of its events
[EV_OSEK_006]	Any task or ISR of category 2 can set an event for a not suspended extended task
[EV_OSEK_007]	It is not possible for an interrupt service routine or a basic task to wait for an event.
[EV_OSEK_008]	An extended task in the waiting state is released to the ready state if at least one event for which the task is waiting for has occurred.
[EV_OSEK_009]	If a running extended task tries to wait for an event and this event has already occurred, the task remains in the running state.
[EV_OSEK_010]	In SetEvent API The service may be called from an interrupt service routine and from the task level, but not from hook routines.
[EV_OSEK_011]	Any events not set in the event mask remain unchanged.
[EV_OSEK_012]	The system service ClearEvent is restricted to extended tasks which own the event.
[EV_OSEK_013]	In SetEvent The events of task <TaskID> are set according to the event mask <Mask>. Calling SetEvent causes the task <TaskID> to be transferred to the ready state, if it was waiting for at least one of the events specified in <Mask>.
[EV_OSEK_014]	In CleartEvent The events of the extended task calling ClearEvent are cleared according to the event mask <Mask>.
[EV_OSEK_015]	This service returns the current state of all event bits of the task <TaskID>, not the events that the task is waiting for.
[EV_OSEK_016]	The service may be called from interrupt service routines, task level and some hook routines
[EV_OSEK_017]	The state of the calling task is set to waiting, unless at least one of the events specified in <Mask> has already been set.
[EV_OSEK_018]	This call of WaitEvent enforces rescheduling, if the wait condition occurs. If rescheduling takes place
[EV_OSEK_019]	This service shall only be called from the extended task owning the event.

Hook Routine

[HR_OSEK_000]	The OSEK OS provides system specific hook routines to allow user-defined actions within the OS internal processing
[HR_OSEK_001]	Hook routines have higher priority than all tasks.
[HR_OSEK_002]	Hook routines can't be interrupted by category 2 interrupt routines
[HR_OSEK_003]	Hook routines are configurable to exist within code or not
[HR_OSEK_004]	StartupHook is called after the OS start-up and before the scheduler is running.

[HR_OSEK_005]	ShutdownHook is called when a system shutdown is requested by the application or by the OS in case of a severe error.
[HR_OSEK_006]	PostTaskHook is called each time directly before the old task leaves the running state
[HR_OSEK_007]	PreTaskHook is called each time directly after a new task enters the running state
[HR_OSEK_008]	ErrorHook is called when an API returns a StatusType value other than E_OK
[HR_OSEK_009]	When ShutdownOS API is called while a task is running, postTaskHook is called before ShutdownHook.
[HR_OSEK_010]	ErrorHook is not called if a system service is called from the ErrorHook itself.
[HR_OSEK_011]	If an API service returns an error, the values of the output parameters are undefined
[HR_OSEK_012]	In ActivateTask API if No error, E_OK is return while Too many task activations of <TaskID>, E_OS_LIMIT is return while <TaskID> is invalid, E_OS_ID is return
[HR_OSEK_013]	In TerminateTask No return states type if the process sucsseeded
[HR_OSEK_014]	In TerminateTask if Task still occupies resources, E_OS_RESOURCE return while if Call at interrupt level, E_OS_CALLEVEL return
[HR_OSEK_015]	In ChainTask No return states type if the process sucsseeded
[HR_OSEK_016]	In ChainTask if Task still occupies resources, E_OS_RESOURCE return while if Too many task activations of <TaskID>, E_OS_LIMIT is return while if Call at interrupt level, E_OS_CALLEVEL return while if E_OS_LIMIT Task <TaskID> is invalid, E_OS_ID
[HR_OSEK_017]	In Shedule API if No error, E_OK is return while if Task still occupies resources, E_OS_RESOURCE return, if Call at interrupt level, E_OS_CALLEVEL is return
[HR_OSEK_018]	In GetTaskID API if No error, E_OK is return
[HR_OSEK_019]	In ActivateTask API if No error, E_OK is return while <TaskID> is invalid, E_OS_ID is return
[HR_OSEK_020]	In SetEvent API if No error, E_OK is return while if <TaskID> is invalid, E_OS_ID is return while if Referenced task is no extended task, E_OS_ACCESS is return while if Events cannot be set as the referenced task is in the suspended state, E_OS_STATE is return
[HR_OSEK_021]	In ClearEvent API if No error, E_OK is return while if Call not from extended task, E_OS_ACCESS, if Call at interrupt level, E_OS_CALLEVEL is return
[HR_OSEK_022]	In GetEvent API if No error, E_OK is return while if <TaskID> is invalid, E_OS_ID is return while if Referenced task is no extended task, E_OS_ACCESS is return while if Events cannot be set as the referenced task is in the suspended state, E_OS_STATE is return

[HR_OSEK_023]	In WaitEvent API if No error, E_OK is return while if Task still occupies resources, E_OS_RESOURCE return, if Call at interrupt level, E_OS_CALLEVEL is return while if Call not from extended task, E_OS_ACCESS

Resource Management

[RM_OSEK_000]	The resource management is used to co-ordinate concurrent accesses of several tasks with different priorities to shared resources.
[RM_OSEK_001]	Two tasks cannot occupy the same resource at the same time.
[RM_OSEK_002]	Priority inversion cannot occur.
[RM_OSEK_003]	Deadlocks do not occur by use of these resources.
[RM_OSEK_004]	Access to resources never results in a waiting state.
[RM_OSEK_005]	RM is useful in case of preemptable and non-preemtable tasks
[RM_OSEK_006]	RM is useful in case of resource sharing between tasks and interrupt service routines
[RM_OSEK_007]	RM is useful in case of resource sharing between interrupt service routine.
[RM_OSEK_008]	The OSEK operating system ensures also that an interrupt service routine is only processed if all resources which might be occupied by that interrupt service routine during its execution have been released.
[RM_OSEK_009]	OSEK strictly forbids nested access to the same resource.
[RM_OSEK_010]	TerminateTask, ChainTask, Schedule, WaitEvent shall not be called while a resource is occupied.
[RM_OSEK_011]	Interrupt service routine shall not be completed with a resource occupied.
[RM_OSEK_012]	In case of multiple resource occupation within one task, the user has to request and release resources following the LIFO principle (stack like).
[RM_OSEK_013]	The scheduler is treated like a resource which is accessible to all tasks
[RM_OSEK_014]	At the system generation, to each resource its own ceiling priority is statically assigned.
[RM_OSEK_015]	The ceiling priority shall be set at least to the highest priority of all tasks that access a resource or any of the resources linked to this resource
[RM_OSEK_016]	The ceiling priority shall be lower than the lowest priority of all tasks that do not access the resource, and which have priorities higher than the highest priority of all tasks that access the resource.
[RM_OSEK_017]	If a task requires a resource, and its current priority is lower than the ceiling priority of the resource, the priority of the task is raised to the ceiling priority of the resource.
[RM_OSEK_018]	If the task releases the resource, the priority of this task is reset to the priority which was dynamically assigned before requiring that resource.
[RM_OSEK_019]	Priority ceiling results in a possible time delay for tasks with priorities equal or below the resource priority. This delay is limited by the maximum time the resource is occupied by any lower priority task.

[RM_OSEK_020]	Tasks which might occupy the same resource as the running task do not enter the running state, due to their lower or equal priority than the running task.
[RM_OSEK_021]	If a resource occupied by a task is released, other task which might occupy the resource can enter the running state. For preemptable tasks this is a point of rescheduling.
[RM_OSEK_022]	Internal resources are resources which are not visible to the user and therefore cannot be addressed by the system functions GetResource and ReleaseResource.
[RM_OSEK_023]	the behaviour of internal resources is exactly the same as standard resources
[RM_OSEK_024]	At most one internal resource can be assigned to a task during system generation.
[RM_OSEK_025]	The resource is automatically taken when the task enters the running (not when it is activated) except when it has already taken the resource.
[RM_OSEK_026]	Nested resource occupation is only allowed if the inner critical sections are completely executed within the surrounding critical section
[RM_OSEK_027]	The internal resource of the task is released while the task is in the waiting state.
[RM_OSEK_028]	Each access to a resource should be encapsulated with calls to the services GetResource and ReleaseResource.
[RM_OSEK_029]	Resources have to be released in reversed order of their occupation

5.3 AUTOSAR Requirements

[SRS_Os_00097]	The SWFRT shall provide a "user" dependent API (function / macro) to convert ticks to time.	[SWS_Os_00001]	The Operating System module shall provide an API that is backward compatible with the OSEK OS API
[SRS_Os_11003]	The operating system shall be able to monitor stack usage and check for a stack overflow on a per executable object basis	[SWS_Os_00067]	The Operating System module shall provide a stack monitoring which detects possible stack faults of Task(s)/Category 2 ISR(s).
[SRS_Os_11003]	The operating system shall be able to monitor stack usage and check for a stack overflow on a per executable object basis	[SWS_Os_00068]	If a stack fault is detected by stack monitoring AND no ProtectionHook is configured, the Operating System module shall call the ShutdownOS service with the status E_OS_STACKFAULT.

[SRS_Os_11009]	The operating system shall prevent the corruption of the OS by any call of a system service	[SWS_Os_00051]	If an invalid address (address is not writable by this OSApplication) is passed as an out-parameter to an Operating System service, the Operating System module shall return the status code E_OS_ILLEGAL_ADDRESS.
[SRS_Os_11009]	The operating system shall prevent the corruption of the OS by any call of a system service	[SWS_Os_00052]	If a Task returns from its entry function without making a TerminateTask or ChainTask call, the Operating System module shall terminate the Task (and call the OsPostTaskHook if configured).
[SRS_Os_11009]	The operating system shall prevent the corruption of the OS by any call of a system service	[SWS_Os_00069]	If a Task returns from its entry function without making a TerminateTask or ChainTask call AND the error hook is configured, the Operating System module shall call the ErrorHook (this is done regardless of whether the Task causes other errors, e.g. E_OS_RESOURCE) with status E_OS_MISSINGEND before the Task leaves the RUNNING state.
[SRS_Os_11009]	The operating system shall prevent the corruption of the OS by any call of a system service	[SWS_Os_00070]	If a Task returns from the entry function without making a TerminateTask or ChainTask call and still holds OSEK Resources, the Operating System module shall release them.
[SRS_Os_11009]	The operating system shall prevent the corruption of the OS by any call of a system service	[SWS_Os_00088]	If an OS-Application makes a service call from the wrong context AND is currently not inside a Category 1 ISR the Operating System module shall not perform the requested action (the service call shall have no effect) and return E_OS_CALLEVEL or the "invalid value" of the service.
[SRS_Os_11009]	The operating system shall prevent the corruption of the OS	[SWS_Os_00092]	If EnableAllInterrupts / ResumeAllInterrupts / ResumeOSInterrupts are called and no

	by any call of a system service		corresponding DisableAllInterrupts / SuspendAllInterrupts / SuspendOSInterrupts was done before, the Operat- ing System module shall not perform this Operating System service.
[SRS_Os_11009]	The operating system shall prevent the corruption of the OS by any call of a system service	[SWS_Os_00093]	If interrupts are disabled/suspended by a Task/ISR/Hook and the Task/ISR/Hook calls any Operating System service (excluding the interrupt services) then the Operating System module shall ignore the service AND shall return E_OS_DISABLEDINT if the service returns a StatusType value.
[SRS_Os_11020]	The OS shall provide a standard interface to tick a software counter	[SWS_Os_00286]	If the input parameter of IncrementCounter is valid, IncrementCounter shall increment the Counter <CounterID> by one (if any alarm connected to this Counter expires, the given action, e.g. Task activation, and is done) and shall return E_OK.
[SRS_Os_11021]	The OS shall provide a mechanism to cascade multiple software counters from a single hardware counter.	[SWS_Os_00301]	The Operating System module shall provide the ability to increment a software Counter as an alternative action on alarm expiry.

5.3 Testing

This part explains the various activities performed as part of Testing of the project:-

5.3.1 Testing Techniques

The Agile methodology was chosen for testing in this project for several reasons.
First, the project was complex and involved multiple components that needed to be tested simultaneously. The Agile methodology allowed for continuous testing and feedback throughout the development process, which helped to identify and address issues early on.
Second, the agile methodology emphasizes collaboration and communication among team members. This was important for this project because it involved multiple stakeholders with

different requirements and expectations. The Agile methodology provided a framework for managing these relationships and ensuring that everyone was working towards the same goals. Third, the agile methodology is flexible and allows for changes and adjustments to be made throughout the development process. This was important for this project because the requirements and specifications were not fully defined at the beginning of the project. The Agile methodology allowed for changes to be made as the project progressed, which helped to ensure that the final product met the needs of the stakeholders.

5.3.2 Testing Types

For this project, several different types of testing were performed to ensure that the operating system was reliable and met the requirements of the stakeholders. These included:

- Unit Testing: Unit testing was performed on individual components of the operating system to ensure that they functioned as expected. This testing was performed using a variety of tools, including the OSEK and AUTOSAR APIs.
- Integration Testing: Integration testing was performed to ensure that the different components of the operating system worked together seamlessly. This testing was performed using both manual and automated testing methods.
- System Testing: System testing was performed to ensure that the operating system functioned correctly as a whole. This testing was performed using a variety of test scenarios and data sets.

5.3.3 Tools Used

A variety of tools were used to perform the different types of testing for this project. These included:

OSEK and AUTOSAR APIs: These APIs were used extensively throughout the project to develop and test the operating system.

TestPad: TestPad was used to hold the different scenarios that will be implemented during the development of the operating system code.

Eclipse IDE: The Eclipse IDE was used to develop and test the operating system code.

Trace32: trace32 was used for automated integration testing.

The testing performed on the operating system showed that it was reliable and met the requirements of the stakeholders. The unit testing and integration testing identified several issues, which were addressed and fixed before system testing was performed. The system testing and performance testing showed that the operating system could handle a variety of scenarios and load levels without issue.

5.3.4 Unit testing

Testpad tools used to handle the different scenarios that will be needed in implementing the test cases and by using those scenarios it ensures that either the test case failed or succeeded.
The scenarios that was extracted from the particularities that was in OSEK/AUTOSAR requirements about every API and how it should work properly.
The next image shows the OSEK specifications and particularities:

13.2.3 System services

13.2.3.1 ActivateTask

Syntax:	StatusType ActivateTask (TaskType <TaskID>)
Parameter (In):	
TaskID	Task reference
Parameter (Out):	none
Description:	The task <TaskID> is transferred from the *suspended* state into the *ready* state[14]. The operating system ensures that the task code is being executed from the first statement.
Particularities:	The service may be called from interrupt level and from task level (see Figure 12-1). Rescheduling after the call to *ActivateTask* depends on the place it is called from (ISR, non preemptable task, preemptable task). If E_OS_LIMIT is returned the activation is ignored. When an extended task is transferred from suspended state into ready state all its events are cleared.
Status:	
Standard:	• No error, E_OK • Too many task activations of <TaskID>, E_OS_LIMIT
Extended:	• Task <TaskID> is invalid, E_OS_ID
Conformance:	BCC1, BCC2, ECC1, ECC2

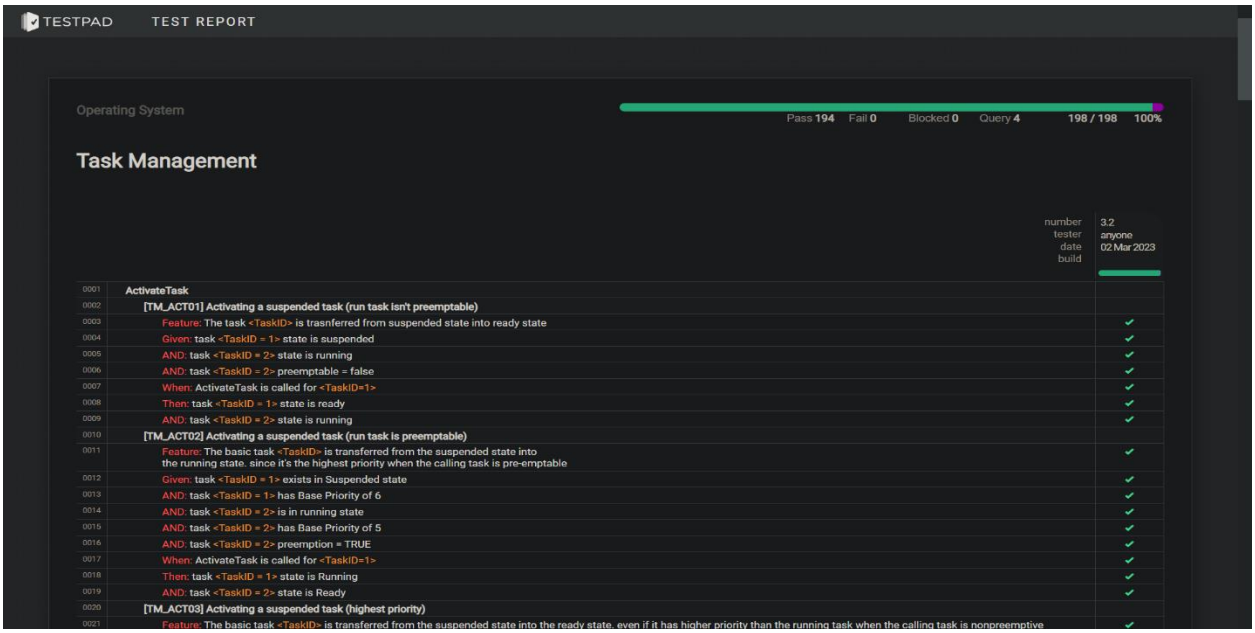
The OSEK provides the syntax, description and everything that is needed to help in implementing the code for this API so after the implementation it should work properly and by using the particularities and description then lots of test cases can be implemented to ensure that it works correctly.

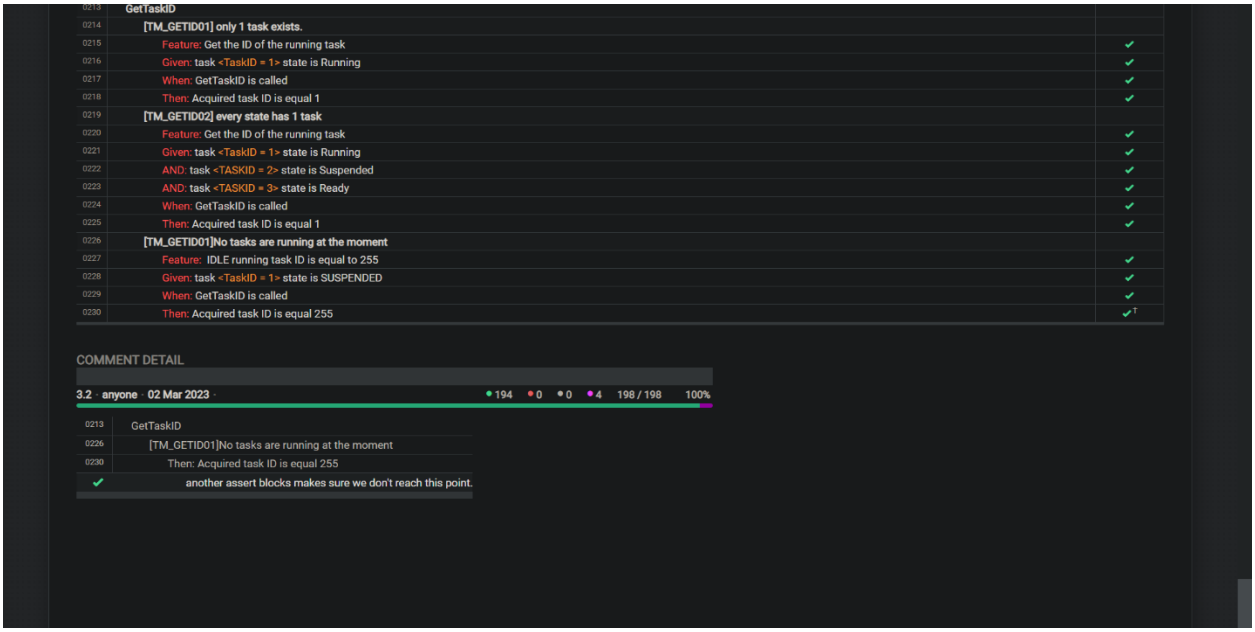
Then by implementing the scenario on Testpad it should look like that:

▼	0001	**ActivateTask**	
▼	0002	**[TM_ACT01] Activating a suspended task (run task isn't preemptable)**	ACTIVE01
	0003	Feature: The task <TaskID> is trasnferred from suspended state into ready state	
	0004	Given: task <TaskID = 1> state is suspended	
	0005	AND: task <TaskID = 2> state is running	
	0006	AND: task <TaskID = 2> preemptable = false	
	0007	When: ActivateTask is called for <TaskID=1>	PARAMETER
	0008	Then: task <TaskID = 1> state is ready	
	0009	AND: task <TaskID = 2> state is running	

During the unit test and after the extraction of the test cases and writing them then it should be tested on Eclipse IDE and by using a library that helps in Automation testing then the test cases will be tested.

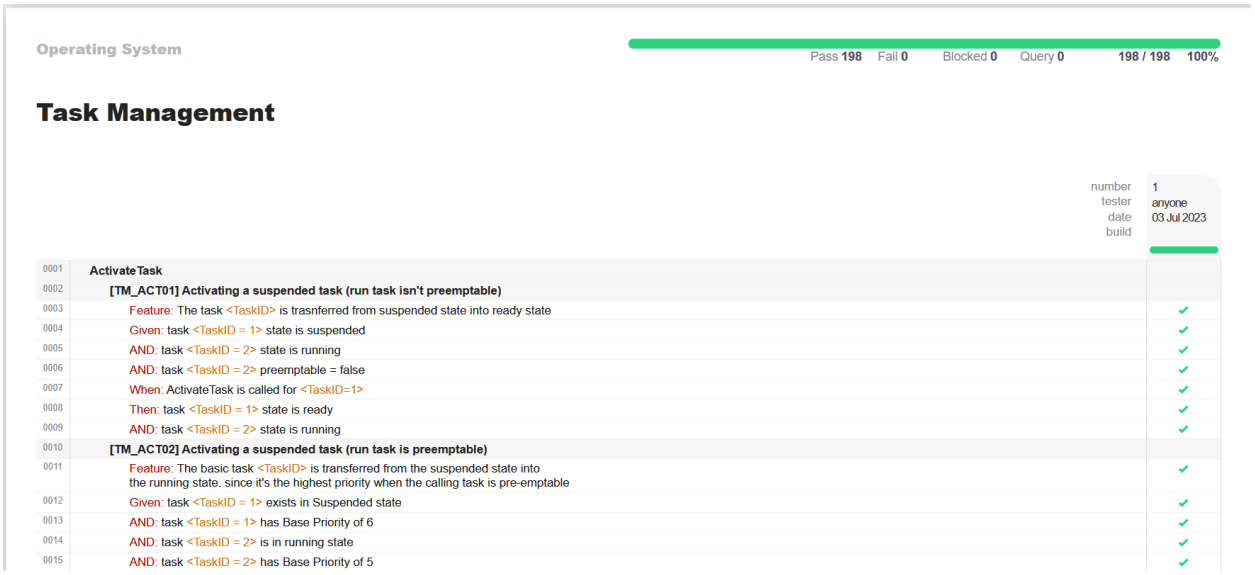
After testing all the API's then a test report which is provided from the Testpad tool





Any comments that were made during the testing is in the Comment Detail section after testing all the scenarios.

The whole test scenarios of Unit testing:



0017	When: ActivateTask is called for <TaskID=1>	✔
0018	Then: task <TaskID = 1> state is Running	✔
0019	AND: task <TaskID = 2> state is Ready	✔
0020	**[TM_ACT03] Activating a suspended task (highest priority)**	
0021	Feature: The basic task <TaskID> is transferred from the suspended state into the ready state. even if it has higher priority than the running task when the calling task is nonpreemptive	✔
0022	Given: task <TaskID = 1> exists in Suspended state	✔
0023	AND: task <TaskID = 1> has Base Priority of 6	✔
0024	AND: task <TaskID = 2> is in running state	✔
0025	AND: task <TaskID = 2> has Base Priority of 5	✔
0026	AND: task <TaskID = 2> is not pre-emptable	✔
0027	When: ActivateTask is called for <TaskID=1>	✔
0028	Then: task <TaskID = 1> state is Ready	✔
0029	AND: task <TaskID = 2> state is Running	✔
0030	**[TM_ACT04] Activating a suspended task (not highest priority)**	
0031	Feature: The basic task <TaskID> is transferred from the suspended state into the ready state since it's not the highest priority task when the calling task is preemptable.	✔
0032	Given: task <TaskID = 1> exists in Suspended state	✔
0033	AND: task <TaskID = 1> has Base Priority of 4	✔
0034	AND: task <TaskID = 2> has Base Priority of 5	✔
0035	AND: task <TaskID = 2> is in running state	✔
0036	AND: task <TaskID = 2> is pre-emptable	✔
0037	When: ActivateTask is called for <TaskID=1>	✔
0038	Then: task <TaskID = 1> state is ready	✔
0039	AND: task <TaskID = 2> state is Running	✔
0040	**[TM_ACT05] Reactivating a task(highest priority)**	
0041	Feature: a basic task can be activated multiple times.	✔
0042	Given: task <TaskID = 1> state is Running	✔
0043	AND: task <TaskID = 1> activation count = 0	✔
0044	AND: task <TaskID = 1> max activation count > 0	✔
0045	AND: task <TaskID = 1> base priority = 3	✔
0046	AND: task <TaskID = 2> state is Ready	✔
0047	AND: task <TaskID = 2> base priority = 2	✔
0048	When: ActivateTask is called for <TaskID=1>	✔
0049	Then: task <TaskID = 1> state is Running	✔
0050	AND: task <TaskID = 1> activation count = 1	✔
0051	AND: task <TaskID = 2> state is Ready	✔
0052	**[TM_ACT06] Reactivating a task(not highest priority)**	
0053	Feature: a basic task can be activated multiple times.	✔
0054	Given: task <TaskID = 1> state is READY	✔
0055	AND: task <TaskID = 1> activation count = 0	✔
0056	AND: task <TaskID = 1> max activation count >0	✔
0057	AND: task <TaskID = 1> base priority = 1	✔
0058	AND: task <TaskID = 2> state is RUNNING	✔
0059	AND: task <TaskID = 2> base priority = 2	✔
0060	When: ActivateTask is called for <TaskID=1>	✔
0061	Then: task <TaskID = 1> state is READY	✔
0062	AND: task <TaskID = 1> activation count = 1	✔
0063	AND: task <TaskID = 2> state is RUNNING	✔
0064	**Ensure Entry Point is reached**	
0065	Feature: Activating an invalid taskID	✔
0066	Given task <TaskID = 10> doesn't exist	✔
0067	When: ActivateTask is called for <TaskID=10>	✔
0068	Then: task <TaskID = 10> isn't found in any running or ready state	✔
0069	**TerminateTask**	
0070	**[TM_TERM01] Terminating a single activation task**	
0071	Feature: The running Task with <TaskID> is currently Running then TerminateTask is called then the Task will be in Suspended State	✔
0072	Given: task <TaskID = 1> is in Running State	✔
0073	AND: task <TaskID = 2> is in ready state	✔
0074	When: task <TaskID = 1> calls TerminateTask	✔
0075	Then: task <TaskID = 1> state is Suspended	✔
0076	AND: task <TaskID = 2 > state is Running	✔
0077		

0078	**[TM_TERM02] Terminating a multiple activation task**	
0079	Feature: The currently running task have multiple activation and TerminateTask is called so, the task will not be terminated but the multiple activation is decremented and then the task with <TaskID> will be in Ready state	✔
0080	Given: task <TaskID = 1> is in Running State	✔
0081	AND: task <TaskID = 1> Activation count = 1	✔
0082	When: task <TaskID = 1> calls TerminateTask	✔
0083	Then: task <TaskID = 1> will be in Running state	✔
0084	AND: task <TaskID = 1> activation_count = 0	✔
0085	**[TM_TERM03] Terminating a multiple activation task (Higher Priority)**	
0086	Feature: The currently running task have multiple activation and TerminateTask is called so, the task will not be terminated but the multiple activation is decremented and then the task with <TaskID> will be in Ready state with the higher priority the task will continue execution	✔
0087	Given: task <TaskID = 1> state is Running	✔
0088	AND: task <TaskID = 1> has Base Priority of 5	✔
0089	AND: task <TaskID = 1> Activation count = 1	✔
0090	AND: task <TaskID = 2> state is Ready	✔
0091	AND: task <TaskID = 2> has Base Priority of 3	✔
0092	When: task <TaskID = 1> calls TerminateTask	✔
0093	Then: task <TaskID = 1> state is Running	✔
0094	AND: task <TaskID = 1> activation_count = 0	✔
0095	AND: task <TaskID = 2> state is Ready	✔
0096	**[TM_TERM04] Terminating a multiple activation task (Lower Priority)**	
0097	Feature: The currently running task have multiple activation and TerminateTask is called so, the task will not be terminated but the multiple activation is decremented and then the task with <TaskID> will be in Ready state the running task is preemptable when another task with higher priority came then the task won't continue execution and move to ready state	✔
0098	Given: task <TaskID = 1> state is Running	✔
0099	AND: task <TaskID = 1> has Priority of 3	✔
0100	AND: task <TaskID = 1> pre-emption disabled	✔
0101	AND: task <TaskID = 2> state is Ready	✔
0102	AND: task <TaskID = 2> has Priority of 5	✔
0103	AND: task <TaskID = 1> Activation count = 1	✔
0104	When: task <TaskID = 1> calls TerminateTask	✔
0105	Then: task <TaskID = 1> state is ready	✔
0106	AND: task <TaskID = 1> activation_count = 0	✔
0107	AND: task <TaskID = 2> state is running	✔

0108	**ChainTask**	
0109	**[TM_CHAIN01] Chaining another task with Highest priority**	
0110	Feature: Calling task is terminated then a succeeding task <TaskID> is activated and is run as soon as possible	✔
0111	Given: task <TaskID = 1> exists in Suspended state	✔
0112	AND: task <TaskID = 1> has Base Priority = 4	✔
0113	AND: task <TaskID = 2> exists in Running state	✔
0114	AND: task <TaskID = 2> has Base Priority = 5	✔
0115	AND: task <TaskID = 2> has Activation count = 0	✔
0116	AND: task <TaskID = 3> exists in Ready state	✔
0117	AND: task <TaskID = 3> has Base Priority = 2	✔
0118	When: task <TaskID=2> calls ChainTask for <TaskID=1>	✔
0119	Then: task <TaskID = 1> state is running	✔
0120	AND: task <TaskID = 2> state is Suspended	✔
0121	AND: task <TaskID = 3> state is ready	✔
0122	**[TM_CHAIN02] Chaining another task with Lowest priority**	
0123	Feature: Calling task is terminated then a succeeding task <TaskID> is activated and is run as soon as possible	✔
0124	Given: task <TaskID = 1> exists in Suspended state	✔
0125	AND: task <TaskID = 1> has Base Priority = 1	✔
0126	AND: task <TaskID = 2> exists in Running state	✔
0127	AND: task <TaskID = 2> has Base Priority = 5	✔
0128	AND: task <TaskID = 2> has Activation count = 0	✔
0129	AND: task <TaskID = 3> exists in Ready state	✔
0130	AND: task <TaskID = 3> has Base Priority = 2	✔
0131	When: task <TaskID=2> calls ChainTask for <TaskID=1>	✔
0132	Then: task <TaskID = 1> state is ready	✔
0133	AND: task <TaskID = 2>state is Suspended	✔
0134	AND: task <TaskID = 3> state is running	✔

0135	**[TM_CHAIN03] Chaining the same task(high priority)**	
0136	Feature: If the succeeding task is identical with the current task, this does not result in multiple requests. The task is not transferred to the suspended state, but will immediately become running again	✔
0137	Given: task <TaskID = 1> in Running state	✔
0138	AND: task <TaskID = 1> has Base Priority = 5	✔
0139	AND: task <TaskID = 1> has Activation count = 0	✔
0140	AND: task <TaskID = 2> in Ready state	✔
0141	AND: task <TaskID = 2> has Base Priority = 2	✔
0142	When: task <TaskID=1> calls ChainTask for <TaskID=1>	✔
0143	Then: task <TaskID = 1>state is running	✔
0144	AND: task <TASKID = 1> activation count = 0	✔
0145	AND: task <TaskID = 2> state is ready	✔
0146	**[TM_CHAIN04] Chaining the same task(low priority)**	
0147	Feature: If the succeeding task is identical with the current task, this does not result in multiple requests. The task is not transferred to the suspended state, but will immediately become running again	✔
0148	Given: task <TaskID =1> in Running state	✔
0149	AND: task <TaskID = 1> has Base Priority = 1	✔
0150	AND: task <TaskID = 1> has Activation count = 0	✔
0151	AND: task <TaskID = 1> preemptive = FALSE	✔
0152	AND: task <TaskID = 2> in Ready state	✔
0153	AND: task <TaskID = 2> has Base Priority = 2	✔
0154	When: task <TaskID = 1> calls ChainTask for <TaskID=1>	✔
0155	Then: task <TaskID = 1>state is ready	✔
0156	AND task <TaskID = 1> activation count = 0	✔
0157	AND: task <TaskID = 2> state is running	✔
0158		
0159	**Schedule**	
0160	**[TM_SCHED01] Task scheduling with No pre-emption (Higher priority task state is Running)**	
0161	Feature: A task with higher priority is Running and another task with lower priority is in Ready state the current task is continued in Running state	✔
0162	Given: task <TaskID = 1> with priority = 15 (highest priority)	✔
0163	AND: task <TaskID = 2> with prioritty = 2	✔
0164	AND: task <TaskID = 1> Pre-emption = false	✔
0165	AND: task <TaskID = 1> state is Running	✔
0166	AND: task <TaskID = 2> state is Ready	✔
0167	AND: task<TaskID = 2> Pre-emption = false	✔
0168	When: Schedule API is	✔
0169	Then: Nothing changed	✔
0170	**[TM_SCHED02] Task scheduling with No pre-emption (Lower priority task state is Running)**	
0171	Feature: If a higher-priority task is ready, the lower priority task is put into the ready state, and the higher-priority task is executed.	✔
0172	Given: task <TaskID = 1> has base priority = 15 (highest priority)	✔
0173	AND: task <TaskID = 1> state is Ready	✔
0174	AND: task <TaskID = 1> Pre-emption = false	✔
0175	AND: task <TaskID = 2> has base priority = 2	✔
0176	AND: task <TaskID = 2> state is Running	✔
0177	AND: task <TaskID = 2> Pre-emption = false	✔
0178	When: Schedule API is called.	✔
0179	Then: task <TaskID = 1> is in Running state	✔
0180	AND: task <TaskID = 2> is in Ready state	✔
0181	**[TM_SCHED03] Task Scheduling with a changed in priority task**	
0182	Feature: There is a changed in priority task while running when Schedule is called then the task is returned into the initial priority	✔
0183	Given: <TaskID = 1> with initial priority = 6	✔
0184	AND: <TaskID = 1> is in Running state	✔
0185	AND: <TaskID = 1> pre-emption = false	✔
0186	AND:<TaskID = 1> Execution priority = 12	✔
0187	AND: <TaskID = 2> with initial priority = 11	✔
0188	AND: <TaskID = 2> is in Ready State	✔
0189	AND: <TaskID = 2> pre-emption = false	✔
0190	When: Schedule API is called	✔
0191	Then:<TaskID = 1> priority = initial priority = 6	✔
0192	AND:<TaskID = 1> state is Ready	✔
0193	AND:<TaskID =2> State is Running	✔

0194	**GetTaskState**	
0195	**[TM_GETSTATE01] get the state of a running task**	
0196	Feature: Get the state of the task that is assigned by an ID	✔
0197	Given: task <TaskID = 1> state is Running	✔
0198	When: GetTaskState is called for <TaskID = 1>	✔
0199	Then: Acquired task state is equal to Running	✔
0200	**[TM_GETSTATE02] get the state of a Suspended task**	
0201	Feature: Get the state of the task that is assigned by an ID	✔
0202	Given: task <TaskID = 1> state is Running	✔
0203	AND: task <TaskID = 2> state is Suspended	✔
0204	When: GetTaskState is called for <TaskID = 2>	✔
0205	Then: Acquired task state is equal to Suspended	✔
0206	**[TM_GETSTATE03] get the state of a Ready task**	
0207	Feature: Get the state of the task that is assigned by ID	✔
0208	Given: task <TaskID = 1> state is Running	✔
0209	AND: task <TaskID = 1> pre-emption = False	✔
0210	AND: task <TaskID = 2> state is Ready	✔
0211	When: GetTaskState is called for <TaskID = 2>	✔
0212	Then: Acquired task state is equal to Ready	✔
0213	**GetTaskID**	
0214	**[TM_GETID01] only 1 task exists.**	
0215	Feature: Get the ID of the running task	✔
0216	Given: task <TaskID = 1> state is Running	✔
0217	When: GetTaskID is called	✔
0218	Then: Acquired task ID is equal 1	✔
0219	**[TM_GETID02] every state has 1 task**	
0220	Feature: Get the ID of the running task	✔
0221	Given: task <TaskID = 1> state is Running	✔
0222	AND: task <TASKID = 2> state is Suspended	✔
0223	AND: task <TASKID = 3> state is Ready	✔
0224	When: GetTaskID is called	✔
0225	Then: Acquired task ID is equal 1	✔
0226	**[TM_GETID01]No tasks are running at the moment**	
0227	Feature: IDLE running task ID is equal to 255	✔
0228	Given: task <TaskID = 1> state is SUSPENDED	✔
0229	When: GetTaskID is called	✔
0230	Then: Acquired task ID is equal 255	✔

COMMENT DETAIL

1 · anyone · 03 Jul 2023 · ● 198 ● 0 ● 0 ● 0 198 / 198 100%

(no test comments, issues or result attachments)

TESTPAD TEST REPORT

Operating System

Pass 108 Fail 0 Blocked 0 Query 0 108 / 108 100%

Event Handling

		number 1 tester anyone date 24 Feb 2023 build
0001	**SetEvent**	
0002	**[EH_T0] Task is Ready and an Event mask is being set (Happy scenario)**	
0003	Feature: When a task is in Ready state and by calling set event to a mask then the number of events will be set to this task and the wait event of the task will be cleared	✔
0004	Given: <Taskid =8> is an Extended task	✔
0005	AND: <Mask = 11> of <Taskid =8> in Binary = 1011	✔
0006	When: SetEvent API is called for <ID=8, mask = Shuffle_Cards>	✔
0007	Then: Current event of <Taskid =8> equals 11	✔
0008	AND: wait event = EVTMASK_NONE	✔
0009	AND: events 3, 5 -> 8 remain unchanged	✔
0010	**[EH_SET_EVT02] Setting event to a Suspended Task**	
0011	Feature: When a task is in suspended state then there will be no event will be set as the task with task id isn't in Ready or Running state Then nothing is triggered or set	✔
0012	Given: Task <Taskid = 9> is an Extended task	✔
0013	AND: <Taskid = 9> state is Suspended	✔
0014	AND: <Mask = 7> of <Taskid =9> which in binary = 111	✔
0015	When: SetEvent is called	✔

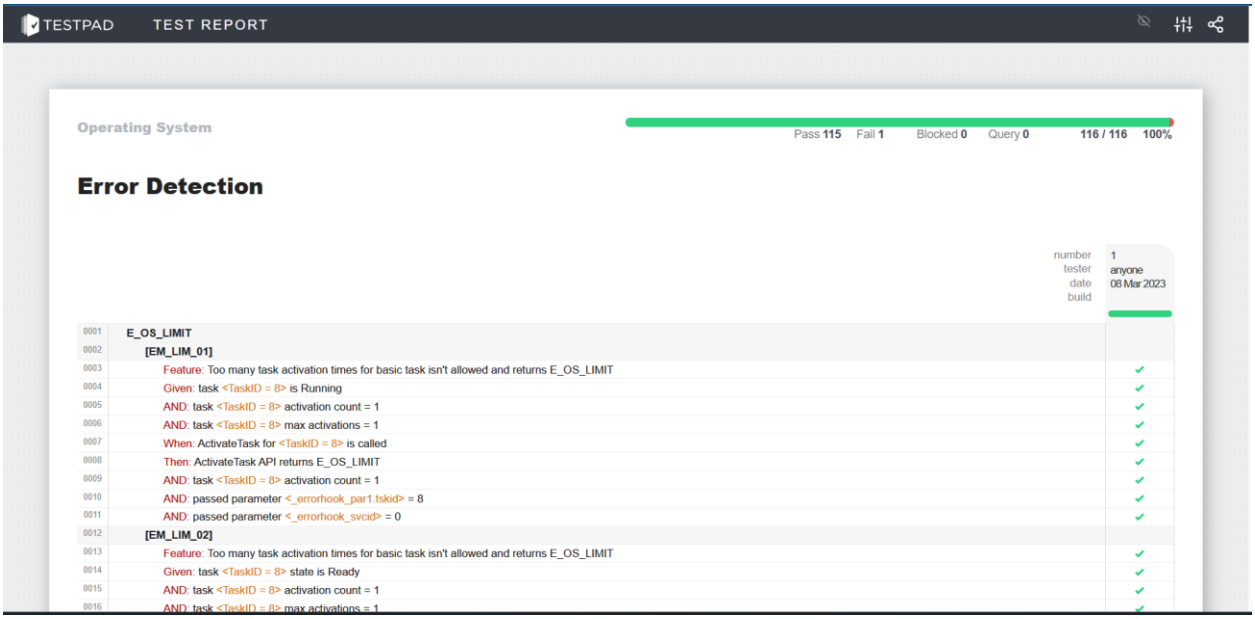
```
0015        When: SetEvent is called                                                                                                  ✔
0016        Then: Nothing changed                                                                                                     ✔
0017        AND: Current event of <Taskid = 9>  isn't triggered                                                                       ✔
0018    [EH_SET_EVT03] Transition of a waiting task to Ready when an event is set
0019        Feature: Task is in Ready state and another task is waiting for the Event to be set then when task is set the state of the other event will be transmitted from waiting to Ready state  ✔
0020        Given: <Taskid =8> is an Extended task                                                                                    ✔
0021        AND: <Taskid =8> state is waiting                                                                                         ✔
0022        AND: <Taskid =8> priority = 9                                                                                             ✔
0023        AND: <Taskid =9> is an Extended task                                                                                      ✔
0024        AND: <Taskid =9> state is Running                                                                                         ✔
0025        AND: <Taskid =9> Priority = 10                                                                                            ✔
0026        AND: <Mask = 2> for <Taskid =9>                                                                                           ✔
0027        When: setEvent is called for <Taskid =9> and <Mask = 2>                                                                   ✔
0028        Then: <Taskid =8> state is Ready                                                                                          ✔
0029        And: <Taskid =9> state is Running                                                                                         ✔
0030        AND: Current event of <Taskid = 9> equals 2                                                                               ✔
0031        AND: wait event = EVTMASK_NONE                                                                                            ✔
0032    [EH_SET_EVT04] Transition of a waiting task to Ready when an event is set
0033        Feature: Task is in Ready state and another task is waiting for the Event to be set then when task is set the state of the other event will be transmitted from waiting to Ready state  ✔
0034        Given: <Taskid =8> is an Extended task                                                                                    ✔
0035        AND: <Taskid =8> state is waiting                                                                                         ✔
0036        AND: <Taskid =8> priority = 10                                                                                            ✔
0037        AND: <Taskid =8> pre-emption = TRUE                                                                                       ✔
0038        AND: <Taskid =9> is an Extended task                                                                                      ✔
0039        AND: <Taskid =9> state is Running                                                                                         ✔
0040        AND: <Taskid =9> Priority = 9                                                                                             ✔
0041        AND: <Taskid =9> Pre-emption = TRUE                                                                                       ✔
0042        AND: <Mask = 2> for <Taskid =9>                                                                                           ✔
0043        When: setEvent is called for <Taskid =9> and <Mask = 2>                                                                   ✔
0044        Then: <Taskid =8> state is Running                                                                                        ✔
0045        AND: <Taskid =9> state is Ready                                                                                           ✔
0046        AND: Current event of <Taskid = 9> equals 2                                                                               ✔
0047        AND: wait event = EVTMASK_NONE                                                                                            ✔
0048
0049    ClearEvent
0050        [EH_CLR_EVT01] Clearing a specific mask of event (Happy Scenario)
0051        Feature: There is an event is being set and then when calling clear the event then the event is being cleared             ✔
0052        Given: <Taskid =11> is an extended task                                                                                   ✔
0053        AND: <Taskid =11> state is Running                                                                                        ✔
0054        AND: <Taskid =11> have <Mask = 7> in binary = 111                                                                         ✔
0055        AND: setEvent for <Taskid =11> and <Mask = 7> is set                                                                      ✔
0056        When: ClearEvent for <Mask = 1> is called                                                                                 ✔
0057        Then: current event of the <Taskid =11>  equals 6                                                                         ✔
0058    [EH_CLR_EVT02] Clearing a mask of a Ready Task
0059        Feature: Clear event is only called if the task is in running state only so when making clear event of a ready task then nothing is cleared despite the event is being set by the task id  ✔
0060        Given: <Taskid =8> is an Extended task                                                                                    ✔
0061        AND: <Taskid =8> state is Ready                                                                                           ✔
0062        AND: <Taskid =8>  it's mask is <Mask = 5>                                                                                 ✔
0063        AND: SetEvent for <Taskid =8>  and <Mask = 5> is called                                                                   ✔
0064        When: ClearEvent for <Mask = 1> is called                                                                                 ✔
0065        Then: current event of <Taskid =8>  equals 5                                                                              ✔
0066
0067    GetEvent
0068        [EH_GET_EVT01] Getting the event mask of a given task (Happy scenario)
0069        Feature: When calling the function it will return the mask of the needed task                                             ✔
0070        Given: <Taskid =10> is an Extended Task                                                                                   ✔
0071        AND: Define EventMaskType eventMask;                                                                                      ✔
0072        AND: <Taskid =10> has <Mask = 2>                                                                                          ✔
0073        AND: setEvent is called for <Taskid =10> and <Mask = 2>                                                                   ✔
0074        When: GetEvent is called for <Taskid =10> and (EventMaskRefType)&eventMask                                               ✔
0075        Then: the current event mask of <Taskid =10> will be returned which equals 2                                             ✔
0076    WaitEvent
0077        [EH_WAIT_EVT01] Moving a task to wait state (Happy Scenario)
0078        Feature:A task is in Running state and when wait event is called then the state of the task will be waiting               ✔
```

0079	Given: <Taskid =10> is an Extended task	✔
0080	AND: <Taskid =10> state is Running	✔
0081	AND: <Taskid =8> is an Extended task	✔
0082	AND: ClearEvent is set for <Taskid =8> and <Mask = 1>	✔
0083	When: WaitEvent is called for <Mask = 1>	✔
0084	Then: <Taskid =10> state is waiting	✔
0085	**[EH_WAIT_EVT02] Task is in Running state and being preempt by another task**	
0086	Feature: we have 2 tasks where one of them is in running and other in ready state the one running when an event is cleared then wait event is called then the state of the running will be waiting and the other =task will be in running state	✔
0087	Given: <Taskid =10> is an Extended task	✔
0088	AND: <Taskid =10> preempt = true	✔
0089	AND: <Taskid =10> has initial priority = 7	✔
0090	AND: <Taskid =10> state is Running	✔
0091	AND: <Taskid =14> is an Extended task	✔
0092	AND: <Taskid =14> state is Ready	✔
0093	AND: <Taskid =10> has <Mask = 7>	✔
0094	AND: <Taskid =14> preempt = true	✔
0095	AND: <Taskid =14> has initial priority = 6	✔
0096	AND: ClearEvent is called for <Taskid =14> and <Mask = 7>	✔
0097	When: WaitEvent is called for <Mask = 7>	✔
0098	Then: <Taskid =10> state is Waiting	✔
0099	AND: <Taskid =14> state is Running	✔
0100		
0101	**[EH_WAIT_EVT03] Task in waiting state and it's execution priority is set to the initial one**	
0102	Feature: there are two tasks one with higher priority than another when wait event is called then the priority of the running task will be back to the initial priority and then will enter waiting state then the other task will be running	✔
0103	Given: <Taskid =10> is an Extended task	✔
0104	AND: <Taskid =10> preempt = True	✔
0105	AND: <Taskid =10> has initial priority = 7	✔
0106	AND:<Taskid =10> has execution priority = 10	✔
0107	AND: <Taskid =10> state is Running	✔
0108	AND: <Taskid =14> is an Extended task	✔
0109	AND: <Taskid =14> state is Ready	✔
0110	AND: <Taskid =10> has <Mask = 7>	✔
0111	AND: <Taskid =14> preempt = True	✔
0112	AND: <Taskid =14> has initial priority = 9	✔
0113	AND: ClearEvent is called for <Taskid =14> and <Mask = 7>	✔
0114	When: WaitEvent is called for <Mask = 7>	✔
0115	Then: <Taskid =10> state is Waiting	✔
0116	AND:<Taskid =10> priority = initial priority	✔
0117	AND: <Taskid =14> state is Running	✔
0118	**[EH_WAIT_EVT04] The event mask of a task isn't cleared or equal to None**	
0119	Feature: when there is a running task and the event mask of another task isn't cleared and still in set then the running task will still be in running state and nothing happened	✔
0120	Given: <Taskid =10> is an Extended task	✔
0121	AND: <Taskid =10> state is Running	✔
0122	AND: <Taskid =8> is an extended task	✔
0123	AND:<Taskid =8> has <Mask = 4>	✔
0124	AND: setEvent is called for <Taskid =8> and <Mask = 4>	✔
0125	When: waitEvent is called for <Mask=4>	✔
0126	Then: <Taskid =10> state is Running	✔

COMMENT DETAIL

1 · **anyone** · **24 Feb 2023** · ● 108 ● 0 ● 0 ● 0 108 / 108 100%

(no test comments, issues or result attachments)



TESTPAD TEST REPORT

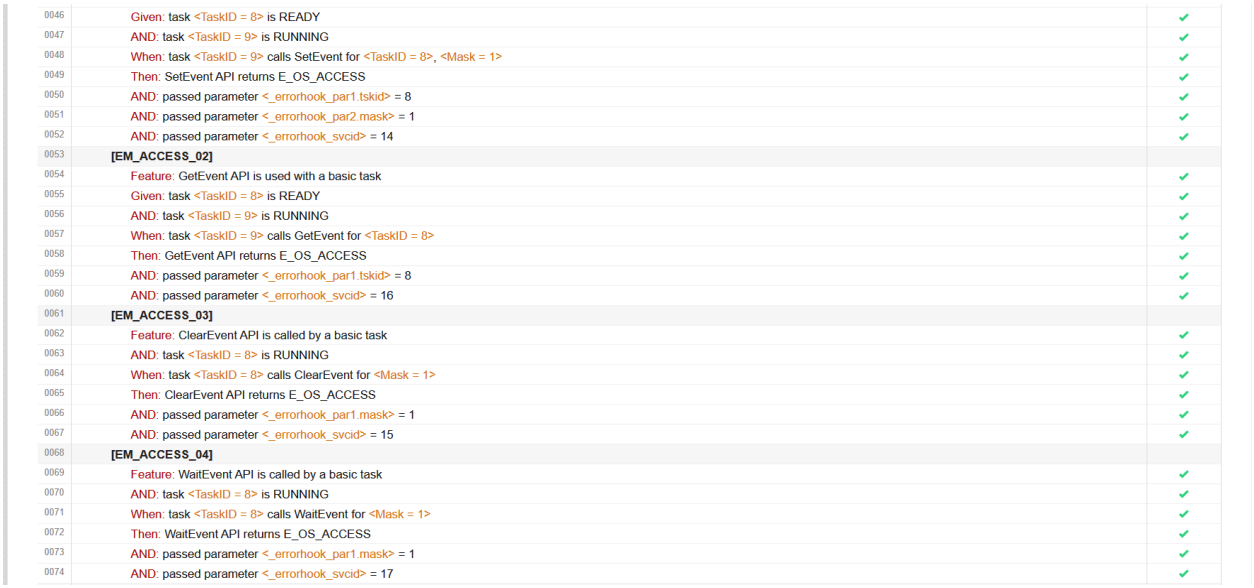
Operating System

Pass 115 Fail 1 Blocked 0 Query 0 116 / 116 100%

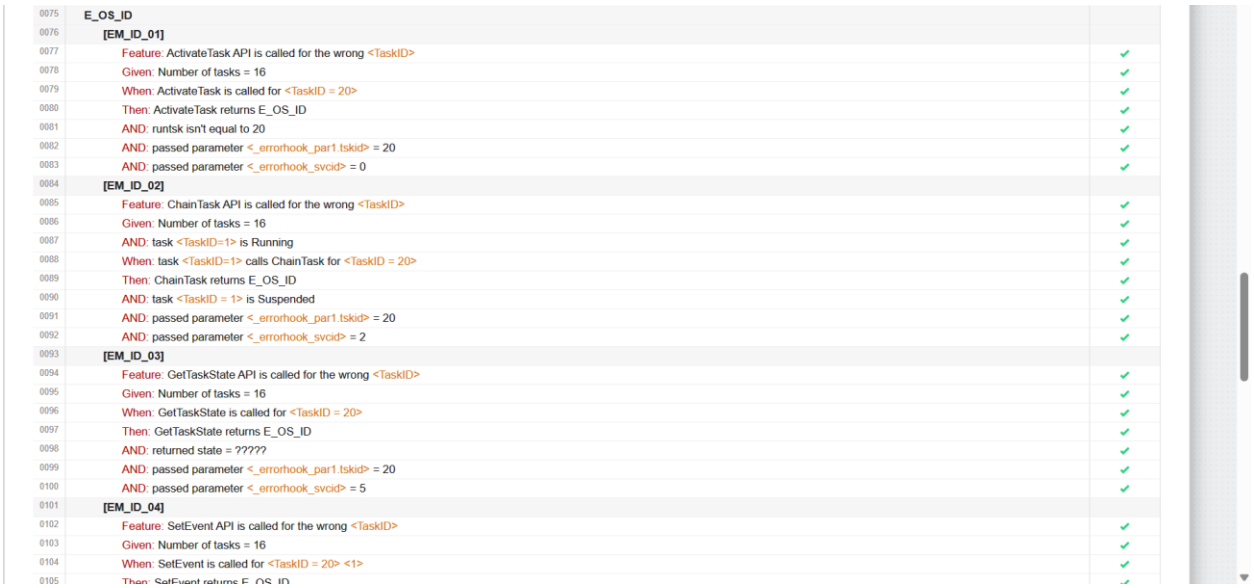
Error Detection

number 1
tester anyone
date 08 Mar 2023
build

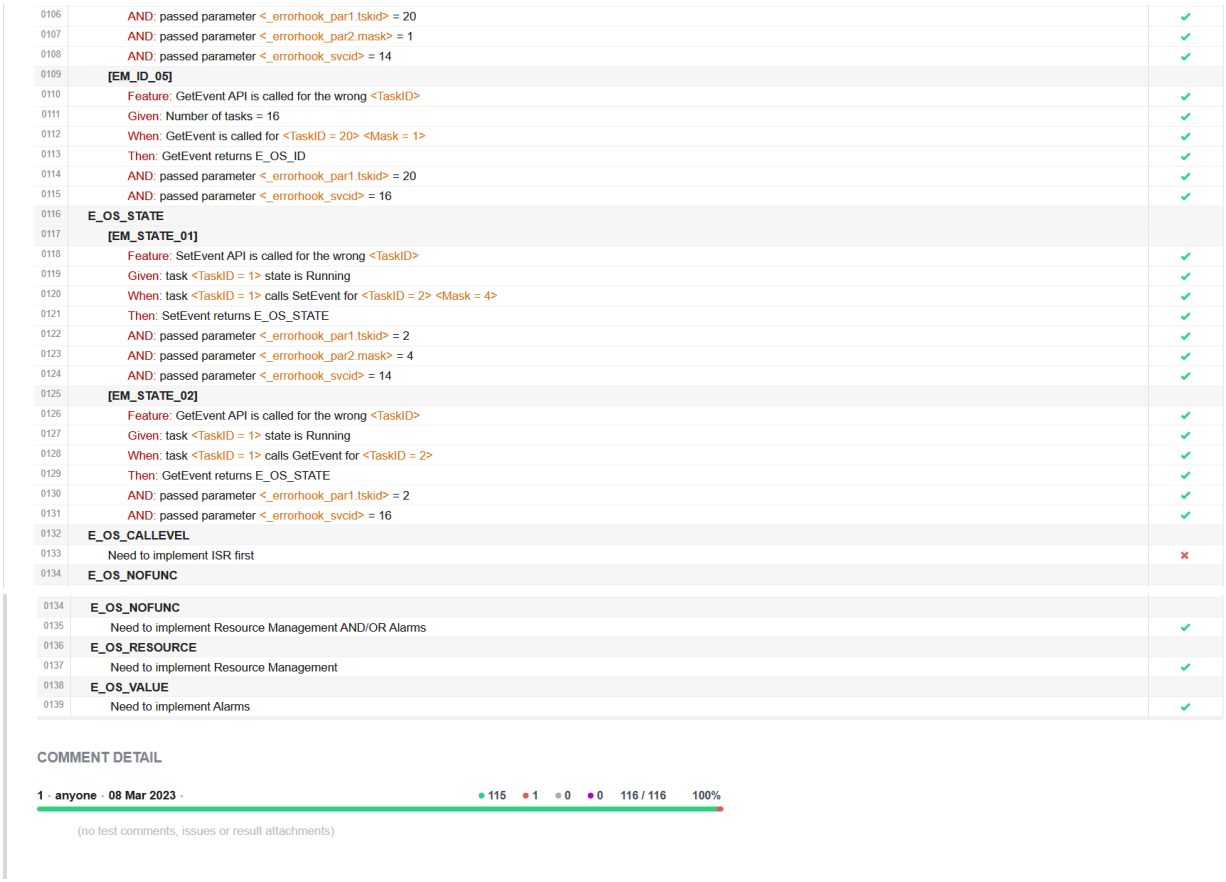
0001	**E_OS_LIMIT**	
0002	**[EM_LIM_01]**	
0003	Feature: Too many task activation times for basic task isn't allowed and returns E_OS_LIMIT	✔
0004	Given: task <TaskID = 8> is Running	✔
0005	AND: task <TaskID = 8> activation count = 1	✔
0006	AND: task <TaskID = 8> max activations = 1	✔
0007	When: ActivateTask for <TaskID = 8> is called	✔
0008	Then: ActivateTask API returns E_OS_LIMIT	✔
0009	AND: task <TaskID = 8> activation count = 1	✔
0010	AND: passed parameter <_errorhook_par1.tskid> = 8	✔
0011	AND: passed parameter <_errorhook_svcid> = 0	✔
0012	**[EM_LIM_02]**	
0013	Feature: Too many task activation times for basic task isn't allowed and returns E_OS_LIMIT	✔
0014	Given: task <TaskID = 8> state is Ready	✔
0015	AND: task <TaskID = 8> activation count = 1	✔
0016	AND: task <TaskID = 8> max activations = 1	✔
0017	AND: task <TaskID = 9> state is Running	✔
0018	AND: task <TaskID = 9> activation count = 0	✔
0019	AND: task <TaskID = 9> preemption = FALSE	✔
0020	When: <TaskID = 9> calls ChainTask for <TaskID = 8>	✔
0021	Then: ChainTask API returns E_OS_LIMIT	✔
0022	AND: task <TaskID = 8> activation count = 1	✔
0023	AND: task <TaskID = 9> state is Suspended	✔
0024	AND: passed parameter <_errorhook_par1.tskid> = 8	✔
0025	AND: passed parameter <_errorhook_svcid> = 2	✔
0026	**[EM_LIM_03]**	
0027	Feature: Reactivation of active Extended Tasks isn't allowed and returns E_OS_LIMIT	✔
0028	Given: task <TaskID = 1> is Running	✔
0029	When: ActivateTask for <TaskID = 1> is called	✔
0030	Then: ActivateTask API returns E_OS_LIMIT	✔
0031	AND: passed parameter <_errorhook_par1.tskid> = 1	✔
0032	AND: passed parameter <_errorhook_svcid> = 0	✔
0033	**[EM_LIM_04]**	
0034	Feature: Reactivation of active Extended Tasks isn't allowed and returns E_OS_LIMIT	✔
0035	Given: task <TaskID = 1> state is Ready	✔
0036	AND: task <TaskID = 2> state is Running	✔
0037	AND: task <TaskID = 2> preemption = FALSE	✔
0038	When: <TaskID = 2> calls ChainTask for <TaskID = 1>	✔
0039	Then: ChainTask API returns E_OS_LIMIT	✔
0040	AND: task <TaskID = 2> state is Suspended	✔
0041	AND: passed parameter <_errorhook_par1.tskid> = 1	✔
0042	AND: passed parameter <_errorhook_svcid> = 2	✔
0043	**E_OS_ACCESS**	
0044	**[EM_ACCESS_01]**	
0045	Feature: SetEvent API is used with a basic task	✔



```
0046        Given: task <TaskID = 8> is READY                                                    ✔
0047        AND: task <TaskID = 9> is RUNNING                                                    ✔
0048        When: task <TaskID = 9> calls SetEvent for <TaskID = 8>, <Mask = 1>                  ✔
0049        Then: SetEvent API returns E_OS_ACCESS                                               ✔
0050        AND: passed parameter <_errorhook_par1.tskid> = 8                                    ✔
0051        AND: passed parameter <_errorhook_par2.mask> = 1                                     ✔
0052        AND: passed parameter <_errorhook_svcid> = 14                                        ✔
0053    [EM_ACCESS_02]
0054        Feature: GetEvent API is used with a basic task                                      ✔
0055        Given: task <TaskID = 8> is READY                                                    ✔
0056        AND: task <TaskID = 9> is RUNNING                                                    ✔
0057        When: task <TaskID = 9> calls GetEvent for <TaskID = 8>                              ✔
0058        Then: GetEvent API returns E_OS_ACCESS                                               ✔
0059        AND: passed parameter <_errorhook_par1.tskid> = 8                                    ✔
0060        AND: passed parameter <_errorhook_svcid> = 16                                        ✔
0061    [EM_ACCESS_03]
0062        Feature: ClearEvent API is called by a basic task                                    ✔
0063        AND: task <TaskID = 8> is RUNNING                                                    ✔
0064        When: task <TaskID = 8> calls ClearEvent for <Mask = 1>                              ✔
0065        Then: ClearEvent API returns E_OS_ACCESS                                             ✔
0066        AND: passed parameter <_errorhook_par1.mask> = 1                                     ✔
0067        AND: passed parameter <_errorhook_svcid> = 15                                        ✔
0068    [EM_ACCESS_04]
0069        Feature: WaitEvent API is called by a basic task                                     ✔
0070        AND: task <TaskID = 8> is RUNNING                                                    ✔
0071        When: task <TaskID = 8> calls WaitEvent for <Mask = 1>                               ✔
0072        Then: WaitEvent API returns E_OS_ACCESS                                              ✔
0073        AND: passed parameter <_errorhook_par1.mask> = 1                                     ✔
0074        AND: passed parameter <_errorhook_svcid> = 17                                        ✔
```



```
0075 E_OS_ID
0076     [EM_ID_01]
0077         Feature: ActivateTask API is called for the wrong <TaskID>                           ✔
0078         Given: Number of tasks = 16                                                          ✔
0079         When: ActivateTask is called for <TaskID = 20>                                       ✔
0080         Then: ActivateTask returns E_OS_ID                                                   ✔
0081         AND: runtsk isn't equal to 20                                                        ✔
0082         AND: passed parameter <_errorhook_par1.tskid> = 20                                   ✔
0083         AND: passed parameter <_errorhook_svcid> = 0                                         ✔
0084     [EM_ID_02]
0085         Feature: ChainTask API is called for the wrong <TaskID>                              ✔
0086         Given: Number of tasks = 16                                                          ✔
0087         AND: task <TaskID=1> is Running                                                      ✔
0088         When: task <TaskID=1> calls ChainTask for <TaskID = 20>                              ✔
0089         Then: ChainTask returns E_OS_ID                                                      ✔
0090         AND: task <TaskID = 1> is Suspended                                                  ✔
0091         AND: passed parameter <_errorhook_par1.tskid> = 20                                   ✔
0092         AND: passed parameter <_errorhook_svcid> = 2                                         ✔
0093     [EM_ID_03]
0094         Feature: GetTaskState API is called for the wrong <TaskID>                           ✔
0095         Given: Number of tasks = 16                                                          ✔
0096         When: GetTaskState is called for <TaskID = 20>                                       ✔
0097         Then: GetTaskState returns E_OS_ID                                                   ✔
0098         AND: returned state = ?????                                                          ✔
0099         AND: passed parameter <_errorhook_par1.tskid> = 20                                   ✔
0100         AND: passed parameter <_errorhook_svcid> = 5                                         ✔
0101     [EM_ID_04]
0102         Feature: SetEvent API is called for the wrong <TaskID>                               ✔
0103         Given: Number of tasks = 16                                                          ✔
0104         When: SetEvent is called for <TaskID = 20> <1>                                       ✔
0105         Then: SetEvent returns E_OS_ID                                                       ✔
```



0106	AND: passed parameter <_errorhook_par1.tskid> = 20	✔
0107	AND: passed parameter <_errorhook_par2.mask> = 1	✔
0108	AND: passed parameter <_errorhook_svcid> = 14	✔
0109	**[EM_ID_05]**	
0110	Feature: GetEvent API is called for the wrong <TaskID>	✔
0111	Given: Number of tasks = 16	✔
0112	When: GetEvent is called for <TaskID = 20> <Mask = 1>	✔
0113	Then: GetEvent returns E_OS_ID	✔
0114	AND: passed parameter <_errorhook_par1.tskid> = 20	✔
0115	AND: passed parameter <_errorhook_svcid> = 16	✔
0116	**E_OS_STATE**	
0117	**[EM_STATE_01]**	
0118	Feature: SetEvent API is called for the wrong <TaskID>	✔
0119	Given: task <TaskID = 1> state is Running	✔
0120	When: task <TaskID = 1> calls SetEvent for <TaskID = 2> <Mask = 4>	✔
0121	Then: SetEvent returns E_OS_STATE	✔
0122	AND: passed parameter <_errorhook_par1.tskid> = 2	✔
0123	AND: passed parameter <_errorhook_par2.mask> = 4	✔
0124	AND: passed parameter <_errorhook_svcid> = 14	✔
0125	**[EM_STATE_02]**	
0126	Feature: GetEvent API is called for the wrong <TaskID>	✔
0127	Given: task <TaskID = 1> state is Running	✔
0128	When: task <TaskID = 1> calls GetEvent for <TaskID = 2>	✔
0129	Then: GetEvent returns E_OS_STATE	✔
0130	AND: passed parameter <_errorhook_par1.tskid> = 2	✔
0131	AND: passed parameter <_errorhook_svcid> = 16	✔
0132	**E_OS_CALLEVEL**	
0133	Need to implement ISR first	✖
0134	**E_OS_NOFUNC**	

0134	**E_OS_NOFUNC**	
0135	Need to implement Resource Management AND/OR Alarms	✔
0136	**E_OS_RESOURCE**	
0137	Need to implement Resource Management	✔
0138	**E_OS_VALUE**	
0139	Need to implement Alarms	✔

COMMENT DETAIL

1 · anyone · 08 Mar 2023 · 115 1 0 0 116 / 116 100%

(no test comments, issues or result attachments)

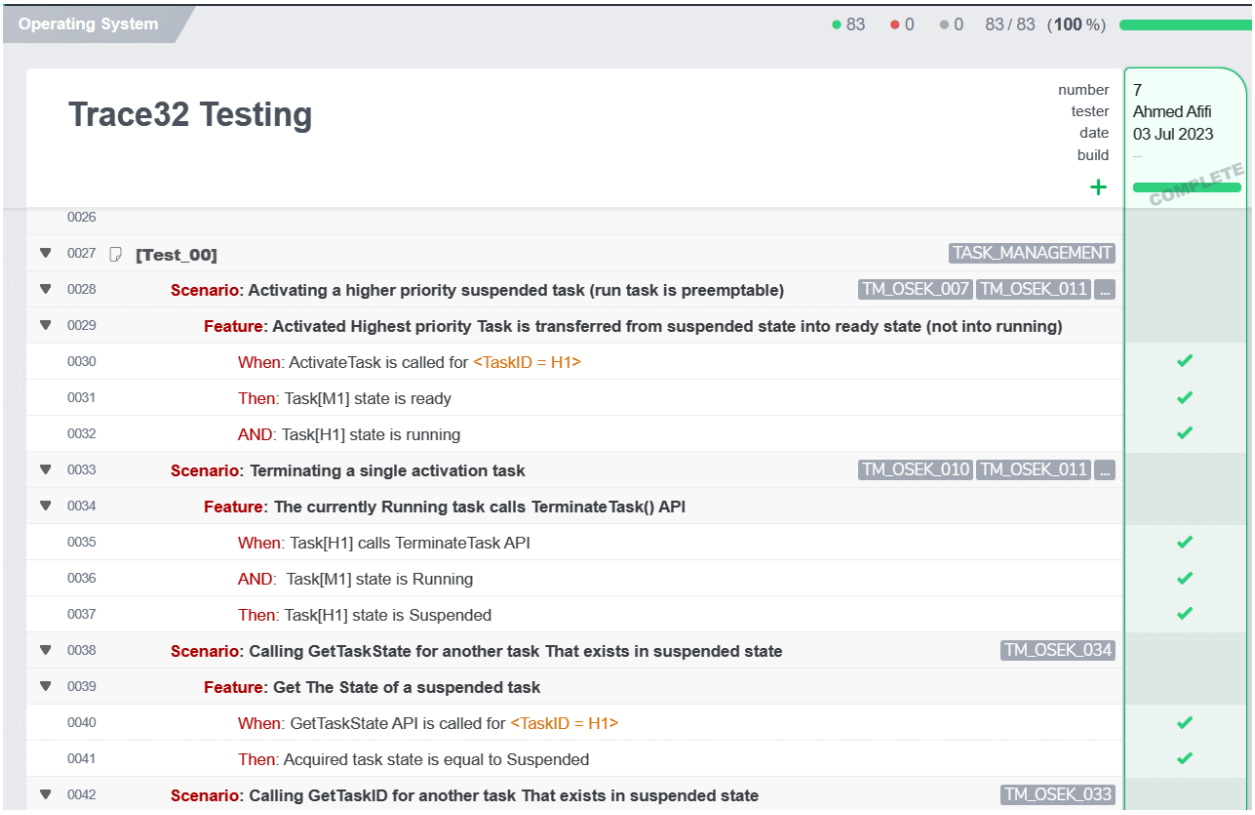
5.3.5 Integration testing

Integration testing is an essential part of the software development process that involves testing the integration of different components of a software system to ensure that they work together seamlessly.

Several challenges were encountered during the integration testing process for this project. These included:

- Integration with Third-Party Components: The project involved integrating the operating system with third-party components, which posed challenges in terms of compatibility and interoperability.
- Integration of Real-Time and Non-Real-Time Components: The operating system included both real-time and non-real-time components, which posed challenges in terms of timing and synchronization.
- Integration of Complex Components: The task management, event handling, error handling, resource management, alarms, and counters components of the operating system were complex and required careful integration testing to ensure that they worked together seamlessly.

To overcome these challenges, several solutions were implemented during the integration testing process. These included:
Compatibility Testing: Compatibility testing was performed to ensure that the operating system was compatible with third-party components and that they worked together seamlessly.
Timing and Synchronization Testing: Timing and synchronization testing were performed to ensure that the real-time and non-real-time components of the operating system worked together seamlessly.
Component-Level Integration Testing: Component-level integration testing was performed to ensure that the task management, event handling, error handling, resource management, alarms, and counters components of the operating system worked together seamlessly.
Automated Testing: Automated testing was used to streamline the integration testing process and ensure that all components of the operating system were tested thoroughly.
The Scenarios that were added was to use all the API's together then test the whole operating system together then apply the failed and successful cases.



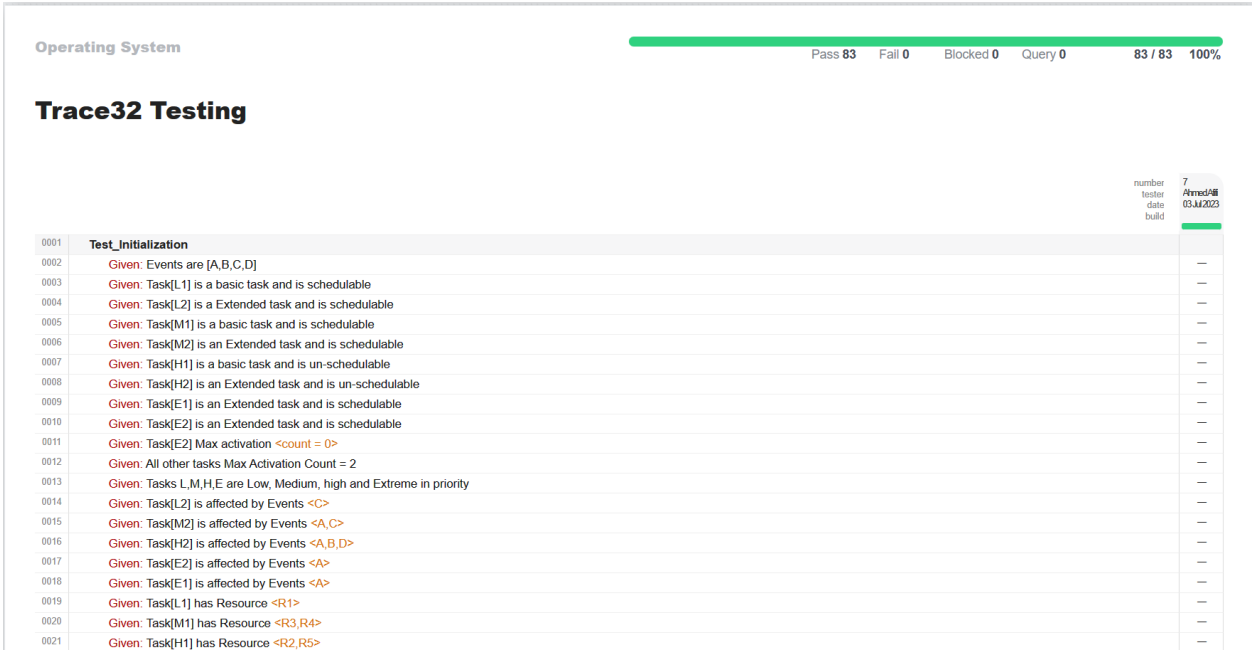
The [Test_00] file includes all the scenarios that depend on each and the tags were for the Specifications that should be in the operating system as without them then it doesn't meet the requirements that the OSEK or AUTOSAR that were put to implement the and achieve the goal for using the OSEK/AUTOSAR.
Tools that used to implement the test cases and ensure the goal of the test cases was:

Trace32: TRACE32 is a software tool for debugging and testing embedded systems that are based on various microprocessors and microcontrollers.

TRACE32 has a wide range of features that make it a powerful tool for debugging and testing embedded systems. Some of the key features of TRACE32 include:

- Real-time debugging: TRACE32 provides real-time debugging capabilities that allow a developer to debug and test an embedded system in real-time.
- Multi-core debugging: TRACE32 supports debugging of multi-core systems, allowing developers to debug and test multiple cores simultaneously.
- On-chip debugging: TRACE32 supports on-chip debugging, which allows developers to debug and test an embedded system without the need for any external hardware.
- Trace analysis: TRACE32 can analyze the trace data generated by an embedded system, allowing developers to identify and diagnose issues in the system.
- Code coverage analysis: TRACE32 can perform code coverage analysis, which allows developers to determine how much of the code has been executed during testing.
- Scripting: TRACE32 has a scripting language that allows developers to automate testing and debugging tasks.

The whole test scenarios of integration test:



0022	Given: Task[L2] has Resource <R2>	—
0023	Given: Task[M2] has Resource <R3>	—
0024	Given: Task[H2] has Resource <R1>	—
0025	Given: Task[M1] is Auto Activated	—
0026		
0027	**[Test_00]**	
0028	**Scenario: Activating a higher priority suspended task (run task is preemptable)**	
0029	**Feature: Activated Highest priority Task is transferred from suspended state into ready state (not into running)**	
0030	When: ActivateTask is called for <TaskID = H1>	✔
0031	Then: Task[M1] state is ready	✔
0032	AND: Task[H1] state is running	✔
0033	**Scenario: Terminating a single activation task**	
0034	**Feature: The currently Running task calls TerminateTask() API**	
0035	When: Task[H1] calls TerminateTask API	✔
0036	AND: Task[M1] state is Running	✔
0037	Then: Task[H1] state is Suspended	✔
0038	**Scenario: Calling GetTaskState for another task That exists in suspended state**	
0039	**Feature: Get The State of a suspended task**	
0040	When: GetTaskState API is called for <TaskID = H1>	✔
0041	Then: Acquired task state is equal to Suspended	✔
0042	**Scenario: Calling GetTaskID for another task That exists in suspended state**	
0043	**Feature: Get The ID of the running task**	
0044	When: GetTaskID API is called	✔
0045	Then: Acquired <TaskID = M1>	✔
0046	**Scenario: Activating a suspended task (not highest priority)**	
0047	**Feature: Activating a lower priority task moves it from Suspended to Ready state without affecting currently running task**	
0048	When: ActivateTask is called for <TaskID = L1>	✔
0049	AND: Task[M1] state is Running	✔
0050	Then: Task[L1] state is ready	✔
0051	**Scenario: Reactivating a task(highest priority)**	
0052	**Feature: a basic task can reactivate itself before termination**	
0053	When: ActivateTask() is called for <TaskID = M1>	✔
0054	AND: TerminateTask() is called	✔

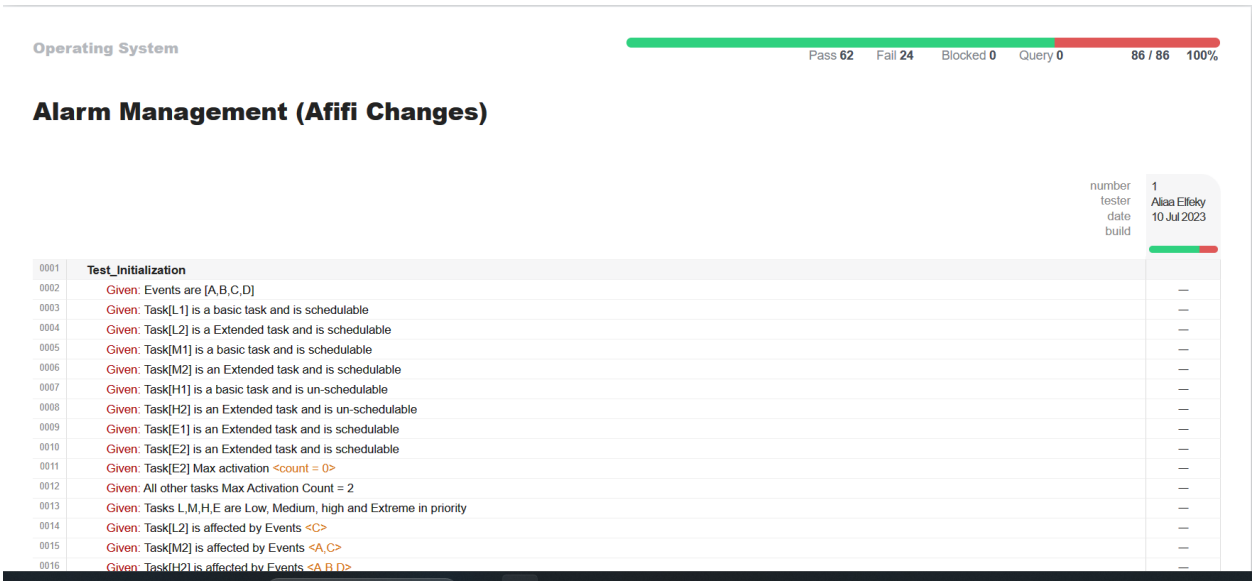
0058	**[Test_01]**	
0059	**Scenario: A Scheduable Task Activates another Task of the same priority Then Activates a task that's higher in priority**	
0060	**Feature: When a task is pre-empted, it should be at the Head of the FIFO Queue of the Schedular meaning it should execute before the other task of the same priority is executed**	
0061	When: ActivateTask is called for <ID = M2>	✔
0062	AND : ActivateTask is called for <ID = H1>	✔
0063	Then: task <TaskID = H1> state is Running	✔
0064	AND: task <TaskID = M1> state is Ready	✔
0065	AND: task <TaskID = M2> state is Ready	✔
0066	AND: M1 is at the head of the FIFO queue <tcb_next[M1] = M2>	✔
0067	**Scenario: A non-Scheduable Task Activates another Task of higher priority should keep in running state**	
0068	**Feature: a non-Scheduable task execution Priority is set to Maximum Priority setting Current priority of a running task to <TPRI_MAX> Denying any other task from taking the processor**	
0069	When: ActivateTask is called for <TaskID = E1>	✔
0070	Then: task <TaskID = E1> state is Ready	✔
0071	AND: task <TaskID = H1> state is Running	✔
0072	**Scenario: A non-Scheduable Task calls Schedule API when there's a task of higher priority than it.**	
0073	**Feature: Schedule API returns Task Priority back to it's initial priority allowing other tasks of higher priority than the running task's initial and of lower priority than the running task's Execution to be allowed to run instead.**	
0074	When: Schedule API is called	✔
0075	Then: task <TaskID = H1> state is READY	✔
0076	AND: task <TaskID = E1> state is RUNNING	✔
0077	**Scenario: A Task reaches its end without the call of Terminate or ChainTask APIs**	
0078	**Feature: OS terminates any task that reaches its end without terminating**	
0079	When: task[E1] reaches its end	✔
0080	Then: task[E1] is Suspended	✔
0081	AND: task[H1] is running	✔
0082	**Scenario: A Task calling ChainTask for another Task would cause Rescheduling to take place**	
0083	**Feature: ChainTask activates the given task and uses schedular to find the highest priority task to be dispatched**	
0084	When: task[H1] Calls ChainTask for <TaskID = L1>	✔
0085	Then: task[H1] is Suspended	✔
0086	**Scenario: the latest Task That has been preempted is executed before any other tasks with the same priority**	
0087	**Feature: Preempted task higher FIFO Priority in its queue**	
0088	When: task[H1] is terminated	✔

0089	Then: task[M1] is running until terminated	✔
0090	Then: task[M2] is running until terminated	✔
0091	Then: task[L1] is running	✔
0092	**Scenario: A Task calling ChainTask for itself shall not leave the running state**	
0093	**Feature: ChainTask API checks if the task to be activated is the task to be terminated.**	
0094	When: task[L1] Calls ChainTask for itself	✔
0095	Then: task[L1] Doesn't go to suspended state	✔
0096	AND: task[L1] is Running	✔
0097	**[Test_02]**	
0098	**Scenario: When activating an extended task, these events are cleared by the OS.**	
0099	When: Task[M1] calls ChainTask for <ID = M2>	✔
0100	AND: Task[M2] calls ActivateTask <ID = L1>	✔
0101	AND: Task[M2] calls ActivateTask <ID = L2>	✔
0102	AND: Task[M2] calls GetEvent <ID = L2>	✔
0103	Then: Returned Event Mask is equal to zero	✔
0104	AND: Task[L1] in Ready State	✔
0105		
0106	**Scenario: A task tries to wait for an Event that it's not supposed to be affected by.**	
0107	When: Task[M2] calls ActivateTask <ID = H2>	✔
0108	AND: Task[H2] calls WaitEvent <mask = C>	✔
0109	Then: Task[H2] remains in Running State	✔
0110		
0111	**Scenario: Only the owner extended task is able to clear its events and to wait for the reception of its events, GetEvent is used to return current state of all event bits, any events not Set in the event mask remain unchanged**	
0112	When: Task[H2] calls SetEvent <TaskID = H2, mask = A|B|D>	✔
0113	AND: Task[H2] calls ClearEvent <mask A|B>	✔
0114	AND: Task[H2] calls GetEvent API	✔
0115	Then: returned <mask = D>	✔
0116		
0117	**Scenario: Calling WaitEvent causes state of the calling task is set to waiting, unless at least one of the events has already been set.**	
0118	When: Task[H2] calls WaitEvent <mask = A|B>	✔
0119	Then: Task[H2] Enters Waiting state	✔
0120		
0121	**Scenario: Calling SetEvent causes the task to be transferred to the ready state, if it was waiting for at least one of the events specified**	
0122	When: Task[L1] calls SetEvent API for <ID = H2, mask = A>	✔
0123	Then: Task[H2] Changes into running state	✔
0124		
0125	**Scenario: Calling SetEvent using basic task as task ID**	
0126	When: Task[H2] calls SetEvent API for <ID = L1, mask = A>	✔
0127	Then: SetEvent returns E_OS_ACCESS	✔
0128		
0129	**Scenario: All tasks terminate and IDLE mechanism runs the processor**	
0130	When: Task[L2] Terminates	✔
0131	Then: <runtsk = 255>	✔
0132	**[Test_03]**	
0133	**Scenario: A task occupies a resource then tries to re-occupy it before releasing it**	
0134	When: Task[M1] tries to re-occupy the same resource	✔
0135	Then: No effect happens	✔
0136	**Scenario: A task Occupying in an order then releases them in a non-LIFO Manner (should cause an error)**	
0137	When: Task[M1] occupies another Resource	✔
0138	Then: Task[M1] releases the firstly occupied resource first	✔
0139	AND: Resource Shall not be released	✔
0140	**Scenario: On termination without releasing that resource The OS shall release the resource first before terminating**	
0141	When: Task[M1] calls ChainTask for <TaskID = L1>	✔
0142	Then: Resources shall be automatically released.	✔
0143	**Scenario: Three Tasks of different priorities, Lowest priority task runs first and reserves a resource then activates two higher priority tasks**	
0144	When: Task[L1] calls GetResource API for <res = R1>	✔
0145	AND: Task[L1] activates task[H2]	✔
0146	AND: Task[L1] activates task[M2]	✔
0147	Then: Task[L1] priority shall be equal to Task[H]	✔
0147	Then: Task[L1] priority shall be equal to Task[H]	✔
0148	Then: Task[L1] remains in Running state	✔
0149	**Scenario: Highest priority task that is blocked until the resource occupied by a lower priority task is released, medium priority Task is also blocked until Highest priority task terminates**	
0150	When: Task[L1] calls ReleaseResource API for <res = R1>	✔
0151	Then: Task[L1] priority shall return to its initial priority	✔
0152	AND: Task[L1] enters Ready state	✔
0153	AND: Task[H2] enters Running state	✔
0154	**Scenario: An extended task occupies a resource then calls WaitEvent or Schedule shall not change its state.**	
0155	When: Task[H2] calls ActivateTask(E1)	✔
0156	AND: Task[H2] calls GetResource <res= R1>	✔
0157	AND: Task[H2] Calls WaitEvent API<mask = A>	✔
0158	AND: Task[H2] Calls Schedule API	✔
0159	Then: Task [H2] remains in Running State	✔

COMMENT DETAIL

7 · **Ahmed Afifi** · **03 Jul 2023** · ● 83 ● 0 ● 0 ● 0 83 / 83 100%

(no test comments, issues or result attachments)



Operating System

Pass 62 Fail 24 Blocked 0 Query 0 86 / 86 100%

Alarm Management (Afifi Changes)

number 1
tester Aliaa Elfeky
date 10 Jul 2023
build

0001	**Test_Initialization**
0002	Given: Events are [A,B,C,D]
0003	Given: Task[L1] is a basic task and is schedulable
0004	Given: Task[L2] is a Extended task and is schedulable
0005	Given: Task[M1] is a basic task and is schedulable
0006	Given: Task[M2] is an Extended task and is schedulable
0007	Given: Task[H1] is a basic task and is un-schedulable
0008	Given: Task[H2] is an Extended task and is un-schedulable
0009	Given: Task[E1] is an Extended task and is schedulable
0010	Given: Task[E2] is an Extended task and is schedulable
0011	Given: Task[E2] Max activation <count = 0>
0012	Given: All other tasks Max Activation Count = 2
0013	Given: Tasks L,M,H,E are Low, Medium, high and Extreme in priority
0014	Given: Task[L2] is affected by Events <C>
0015	Given: Task[M2] is affected by Events <A,C>
0016	Given: Task[H2] is affected by Events <A,B,D>

0016	Given: Task[H2] is affected by Events <A,B,D>	—
0017	Given: Task[E2] is affected by Events <A>	—
0018	Given: Task[E1] is affected by Events <A>	—
0019	Given: Task[L1] has Resource <R1>	—
0020	Given: Task[M1] has Resource <R3,R4>	—
0021	Given: Task[H1] has Resource <R2,R5>	—
0022	Given: Task[L2] has Resource <R2>	—
0023	Given: Task[M2] has Resource <R3,R4>	—
0024	Given: Task[H2] has Resource <R1>	—
0025	Given: HW_C0 is a Hardware Counter Belong to Timer0	—
0026	Given: HW_C1 is a Hardware Counter Belong to Timer0	—
0027	Given: SW_C2 is a Software counter	—
0028	Given: SW_C3 is a Software counter	—
0029	Given: SW_C2 <maximum Counter Value = 5>	—
0030	Given: SW_C3 <maximum Counter Value = 6>	—
0031	Given: Alarm [A1] belongs to <CounterID = SW_C2>	—
0032	Given: Alarm [A2] belongs to <CounterID = SW_C3>	—
0033	Given: Alarm [A3] belongs to <CounterID = SW_C3>	—
0034	Given: Alarm [A4] belongs to <CounterID = SW_C3>	—
0035	Given: Alarm [A5] belongs to <CounterID = SW_C4>	—
0036	Given: Alarm [A6] belongs to <CounterID = SW_C4>	—
0037	Given: Alarm [A7] belongs to <CounterID = HW_C0>	—
0038	Given: Alarm [A8] belongs to <CounterID = HW_C1>	—
0039	Given: Alarm[A1] Action is Activating Task[L1]	—
0040	Given: Alarm[A2] Action is calling <AlarmCallBack = CB2>	—
0041	Given: Alarm[A3] Action is Activating Task[H2]	—
0042	Given: Alarm[A4] Action is Setting an Event for (TaskID = H2, eventMask = A)	—
0043	Given: Alarm[A5] Action is Increment <SW_C3>	—
0044	Given: Alarm[A6] Action iscalling <AlarmCallBack = CB3>	—
0045	Given: Alarm[A7] Action is Activating Task[L1]	—
0046	Given: Alarm[A8] Action is Setting an Event for (TaskID = H2, eventMask = A)	—
0047	Given: Task[M1] is Auto Activated	—
0048	Given: ALARMS_AUTOSTART_COUNT equal 2	—
0049	**[Test_04]**	
0050	**Scenario: Calling SetAbsAlarm API Enables an Alarm and sets it AlarmTime.**	
0051	WHEN: Task[M1] calls SetAbsAlarm API is called for <alarmID = A1, start = 3, cycle=0 >	✔
0052	AND: GetAlarm() API is called for <AlarmID = A1>	✔
0053	Then: Alarm[A1] is Enabled	✔
0054	AND: returned <Alarm_time = 3>	✔
0055	**Scenario: calling IncrementCounter API increments the Count_time by one and decrements the alarm count by one.**	
0056	WHEN: IncrementCounter API is called for <countID = SW_C2>	✔
0057	AND: GetAlarm() API is called for <AlarmID = A1>	✔
0058	Then: returned <Alarm_time = 2>	✔
0059	**Scenario : GetCounterValue returns the current counter tick value.**	
0060	When: GetCounterValue API is called for <countID = SW_C2, Value>	✔
0061	Then: returned <Counter_Value = 1>	✔
0062	**Scenario: An alarm will expire when a predefined counter value is reached. (SetAbsAlarm)**	
0063	WHEN: IncrementCounter API is called for <countID = SW_C2> twice	✔
0064	Then: Alarm <alarmID = A1> will expire	✔
0065	AND: Alarm[A1] state is disabled	✔
0066	AND: Task[L1] is at ready state	✔
0067	**Scenario: An alarm will expire when a predefined counter value is reached. (SetAbsAlarm)**	
0068	Note: <countID = SW_C2> count equals 3	✔
0069	When: SetAbsAlarm API is called for <alarmID = A1, start = 1, cycle=0 >	✔
0070	AND : IncrementCounter API is called for <CounterID = SW_C2> three times	✔
0071	Then: Alarm[A1] will expire	✔
0072	AND: Alarm[A1] will be disabled	✔
0073	AND: Task[L1] will be reactivated (hint: check reactivation count)	✔
0074	**Scenario: A Cyclic alarm will expire when a predefined counter value is reached and will be reactivated (SetAbsAlarm)**	
0075	When: IncrementCounter is called for <CounterID = SW_C3>	✔
0076	Note: <countID = SW_C3> count equals 1	✔

0077	AND: SetAbsAlarm API is called for <alarmID = A2, start = 2, cycle = 1 >	✔
0078	And: IncrementCounter is called for <CounterID = SW_C3>	✔
0079	Then: Alarm[A2] will Expire	✔
0080	AND: Alarm[A2] will remain in enabled state	✔
0081	AND: CallBackFunction[CB2] will be running	✔
0082	**Scenario: using SetRelAlarm where it's value is very small then the alarm shall expire, Also the Alarm Expiry Activates a higher priority task causing Rescheduling to take place**	
0083	When: SetRelAlarm API is called for <alarmID = A3, increment = 0, cycle=0 >	✔
0084	Then: Alarm[A3] will be expired immediately	✔
0085	AND: ALARM[A3] state is Disabled	✔
0086	AND: Task[H2] is in Running State	✔
0087	AND: Task[M1] is in Ready State	✔
0088	**Scenario: using Cancel alarm to cancel the alarm and nothing should happen when a counter reaches the value of alarm expiry**	
0089	When: CancelAlarm API is called for <alarmID = A2>	✔
0090	When: IncrementCounter API is called for <CounterID = SW_C3>	✔
0091	When: H2 calls WaitEvent API for <Msk = A>	✔
0092	Then: Alarm[A2] state will be Disabled	✔
0093	AND: Alarm[A2] Will not expire	✔
0094	And: Task[H2] state will be Waiting	✔
0095	AND: Task[M1] is in Running State	✔
0096	**Scenario: using SetRelAlarm for a cyclic alarm that on expiry activates a higher priority Task**	
0097	When: SetRelAlarm API is called for <alarmID = A4, increment = 2, cycle=1 >	✔
0098	And: IncrementCounter API is called for <CounterID = SW_C3> twice	✔
0099	Then: Alarm[A4] will Expiry (hint : sets event of H2)	✔
0100	AND: Task[M1] is Ready State	✔
0101	AND: Task[H2] is Running State	✔
0102		
0103		
0104	**[Test_04] Scenario: Autostart of an alarm**	
0105	Given: Alarm [A5] belongs to <CounterID = SW_C4> (same counter of A6)	✔
0106	When: <alarmID = A5> cycle = 1, start = 2	✔
0107	AND: <alarmID = A6> cycle = 0, start =4	✔
0108	then: Alarm <alarmID = A5 > is enabled and start counting	✔
0109	AND: <alarmID = A5> will expire after 2 counts	✔
0110	AND: Alarm <alarmID = A6> is enabled and start counting	✔
0111	AND: <alarmID = A6> will expire after 4 counts	✔
0112	**Scenario: testing GetElapsedValue with any value**	
0113	Given: define a TickRefType for variable named User_val equals 3	✔
0114	AND:define a TickRefType for variable named Elapsed_val	✔
0115	When: IncrementCounter is called for <CounterID = SW_C4> where counter value is 0	✔
0116	AND: GetElapsedValue API is called for <countID = SW_C4, Value, ElapsedValue >	✔
0117	Then: the value of the User_val equals 0	✔
0118	AND: Elapsed_val it's value equals 3	✔
0119	**Scenario: GetCounterValue of the current count time of a software count then call the GetElapsedValue**	
0120	When: IncrementCounter is called for <CounterID = SW_C4> where current count = 1	✔
0121	AND: GetCounterValue API is called for <countID = SW_C4, Value>	✔
0122	AND: IncrementCounter is called for <CounterID = SW_C4>	✔
0123	AND: GetElapsedValue API is called for <countID = SW_C4, Value, ElapsedValue >	✔
0124	Then: the value of the User_val equals 2	✔
0125	AND: Elapsed_val it's value equals 1	✔
0126	**HW_counter**	
0127	**Scenario: Hardware counter is used for Alarms**	
0128	When: Calling Timer0_init <clock_source =0, prescaler_enable =0,prescaler = 0>	✘
0129	AND: SetAbsAlarm API is called for <alarmID = A7, start = 10, cycle = 100>	✘
0130	AND:Timer overflows for frequency = 50Mhz	✘
0131	Then: counter <CounterID = HW_C0> will increment	✘
0132	AND: counter <CounterID = HW_C1> will increment	✘
0133	AND: Alarm [A7] will expire after 2 increments	✘
0134	AND: Alarm[A7] will remain in enabled state	✘
0135	AND Task[L2] is activated	✘
0136	AND: Alarm [A7] will expire after 1 increment again	✘
0137	AND Task[L2] is reactivated	✘

0140	**Scenario: Hardware counter is used for Alarms**	
0141	When: Calling Timer0_init <clock_source =0, prescaler_enable =1,prescaler = 4>	✖
0142	AND: SetAbsAlarm API is called for <alarmID = A7, start = 2, cycle = 1 >	✖
0143	AND: SetAbsAlarm API is called for <alarmID = A8, start = 1, cycle = 0 >	✖
0144	AND:Timer overflows for a frequency 50Mhz/4 = 12.5Mhz	✖
0145	Then: counter <CounterID = HW_C0> will increment	✖
0146	AND: counter <CounterID = HW_C1> will increment	✖
0147	AND: Alarm [A8] will expire after 1 increment	✖
0148	Then: Alarm[A8] will Expiry (hint : sets event of H2)	✖
0149	AND: Task[M1] is Ready State	✖
0150	AND: Alarm [A7] will expire after 2 increments	✖
0151	AND: Alarm[A7] will remain in enabled state	✖
0152	AND Task[L2] is activated	✖
0153	AND: Alarm [A7] will expire after 1 increment again	✖
0154	AND Task[L2] is reactivated	✖
0155		
0156		

COMMENT DETAIL

1 · **Aliaa Elfeky** · **10 Jul 2023** · 62 24 0 0 86 / 86 100%

(no test comments, issues or result attachments)

TESTPAD TEST REPORT

Operating System

Pass **20** Fail **0** Blocked **0** Query **0** **20 / 20** **100%**

Resource Management

number	2
tester	Omar Kadry
date	15 Jul 2023
build	

0001	**Test_Initialization**	
0002	Given: Events are [A,B,C,D]	—
0003	Given: Task[L1] is a basic task and is schedulable	—
0004	Given: Task[L2] is a Extended task and is schedulable	—
0005	Given: Task[M1] is a basic task and is schedulable	—
0006	Given: Task[M2] is an Extended task and is schedulable	—
0007	Given: Task[H1] is a basic task and is un-schedulable	—
0008	Given: Task[H2] is an Extended task and is un-schedulable	—
0009	Given: Task[E1] is an Extended task and is schedulable	—
0010	Given: Task[E2] is an Extended task and is schedulable	—
0011	Given: Tasks L,M,H,E are Low, Medium, high and Extreme in priority	—
0012	Given: Task[L2] is affected by Events <C>	—
0013	Given: Task[M2] is affected by Events <A,C>	—
0014	Given: Task[H2] is affected by Events <A,B,D>	—
0015	Given: Task[E1] is affected by Events <A>	—

0015	Given: Task[E1] is affected by Events <A>	—
0016	Given: Task[E2] is affected by Events <A>	—
0017	Given: Task[L1] has Resource <R1>	—
0018	Given: Task[L2] has Resource <R2>	—
0019	Given: Task[M1] has Resource <R3,R4>	—
0020	Given: Task[M2] has Resource <R3>	—
0021	Given: Task[H1] has Resource <R1,R5>	—
0022	Given: Task[H2] has Resource <R2>	—
0023	Given: Task[M1] is Auto Activated	—
0024		
0025	**Test_03**	
0026	**Scenario: A task occupies a resource then tries to re-occupy it before releasing it**	
0027	When: Task[M1] tries to re-occupy the same resource	✔
0028	Then: No effect happens	✔
0029	**Scenario: A task Occupying in an order then releases them in a non-LIFO Manner (should cause an error)**	
0030	When: Task[M1] occupies another Resource	✔
0031	Then: Task[M1] releases the firstly occupied resource first	✔
0032	AND: Resource Shall not be released	✔
0033	**Scenario: On termination without releasing that resource The OS shall release the resource first before terminating**	
0034	When: Task[M1] calls ChainTask for <TaskID = L1>	✔
0035	Then: Resources shall be automatically released.	✔
0036	**Scenario: Three Tasks of different priorities, Lowest priority task runs first and reserves a resource then activates two higher priority tasks**	
0037	When: Task[L1] calls GetResource API for <res = R1>	✔
0038	AND: Task[L1] activates task[H1]	✔
0039	AND: Task[L1] activates task[M2]	✔
0040	Then: Task[L1] priority shall be equal to Task[H]	✔
0041	Then: Task[L1] remains in Running state	✔
0042	**Scenario: Highest priority task that is blocked until the resource occupied by a lower priority task is released, medium priority Task is also blocked until Highest priority task terminates**	
0043	When: Task[L1] calls ReleaseResource API for <res = R1>	✔
0044	Then: Task[L1] priority shall return to its initial priority	✔
0045	AND: Task[L1] enters Ready state	✔
0046	AND: Task[H1] enters Running state	✔
0047	**Scenario: An extended task occupies a resource then calls WaitEvent or Schedule shall not change its state.**	
0048	When: Task[H2] calls ActivateTask(E1)	✔
0049	AND: Task[H2] calls GetResource <res= R1>	✔
0050	AND: Task[H2] Calls WaitEvent and Schedule API	✔
0051	Then: Task [H2] remains in Running State	✔

COMMENT DETAIL

2 · **Omar Kadry** · **15 Jul 2023** · 20 0 0 0 20 / 20 100%

(no test comments, issues or result attachments)

6. Tools and technologies

6.1 GitHub

GitHub is a web-based platform that provides hosting for software development projects using the GitHub version control system. It offers a range of features and tools that support collaborative software development, including code management, issue tracking, project management, and team communication. Developers can use GitHub to store and share their code repositories, collaborate with other developers on a project, and contribute to open source projects. It provides a range of collaboration features, such as pull requests, issues, and code reviews, that allow developers to review and approve code changes before they are merged into the main codebase. GitHub also provides a range of tools for project management, including project boards, milestones, and task lists, that allow teams to organize their work and track progress. It also integrates with a range of other tools and services, including continuous integration and deployment (CI/CD) tools like Jenkins and Travis CI, and communication tools like Slack and Microsoft Teams.

6.2 ARM Cortex-M3

The ARM Cortex-M3 is a 32-bit microcontroller core designed for embedded systems. It is designed to be low power and highly efficient, making it suitable for a wide range of applications, from low-power devices to high-performance embedded systems.
The Cortex-M3 core is based on a Harvard architecture, which separates the instruction and data memory to improve performance and reduce power consumption. It supports a range of memory types, including flash, SRAM, and EEPROM, and provides a range of peripherals, such as timers, serial interfaces, and analog-to-digital converters. The Cortex-M3 core is widely used in a variety of industries, including automotive, industrial automation, and consumer electronics. It is also popular for use in hobbyist and educational projects, due to its ease of use and availability of low-cost development boards.

6.3 HSM Core

"HSM core" is not a well-defined term in the context of hardware security modules (HSMs). However, HSMs typically have a core set of features for managing and safeguarding digital keys and performing encryption/decryption operations. These core features may include:
Tamper-resistant design: HSMs are built to resist physical tampering, such as attempts to open or modify the device.

- Secure key storage: HSMs provide a secure environment for storing cryptographic keys, protecting them from unauthorized access.

- Cryptographic operations: HSMs can perform a range of cryptographic operations, including encryption, decryption, hashing, and digital signature generation and verification.
- Key management: HSMs provide tools for generating, importing, exporting, and revoking cryptographic keys.
- Access control: HSMs provide mechanisms for controlling access to cryptographic keys and operations, such as role-based access control and multi-factor authentication.
- Compliance and certification: HSMs are often certified to meet industry standards for security, such as FIPS 140-2 for government applications and PCI-DSS for payment card processing.

Overall, the core features of an HSM are designed to provide a high level of security for digital keys and cryptographic operations, protecting sensitive data and assets from unauthorized access and manipulation.

6.4 Tasking Compiler

Tasking is a brand of software development tools used for developing embedded systems software. It is developed and marketed by Altium Limited, a global provider of software and hardware design solutions for electronic products. Tasking provides a range of development tools for embedded systems software development, including compilers, debuggers, and integrated development environments (IDEs). These tools support a wide range of microprocessors, microcontrollers, and other embedded systems architectures.
Tasking is designed to support a range of programming languages, including C, C++, and Assembler, and provides advanced optimization techniques to generate efficient and optimized code. It also provides a range of debugging and profiling tools to help developers identify and fix issues in their code.
Tasking is widely used in industries such as automotive, aerospace, and industrial automation, where the development of high-quality and reliable embedded systems software is critical. It is also used in academic and research settings for teaching and research purposes. Tasking Compiler is a software development tool used for developing embedded systems software. It is developed and marketed by Altium Limited, a global provider of software and hardware design solutions for electronic products.

6.5 HighTec Compiler

HighTec Compiler is a software development tool used for developing embedded systems software. It is developed and marketed by HighTec EDV-Systeme GmbH, a German software company that specializes in developing software tools for embedded systems.The HighTec Compiler provides a range of features for developing embedded systems software, including compilers, debuggers, and integrated development environments (IDEs). It supports a wide range of microprocessors, microcontrollers, and other embedded systems architectures

6.6 Trace32 Debugger

TRACE32 is a powerful software development tool used for debugging, testing and analyzing embedded systems. It is developed and marketed by Lauterbach GmbH, a German company that specializes in providing debugging tools and support for various microprocessor architectures. The TRACE32 toolset includes hardware debuggers, software debuggers, and software trace tools that support a wide range of microprocessors and microcontrollers from different vendors. TRACE32 can be used for debugging and testing on various operating systems, including Windows, Linux, and UNIX. Some features of TRACE32 include real-time tracing and visualization of system behavior, advanced debugging capabilities like instruction and data trace, and support for various programming languages such as C, C++, and Assembler. TRACE32 also provides support for various communication interfaces, including Ethernet, USB, and JTAG.

6.7 Eclipse IDE

Eclipse is an open-source integrated development environment (IDE) that is primarily used for developing software applications in Java, although it supports a wide range of programming languages through plugins. It is one of the most popular IDEs used by developers worldwide and is available for free download under the Eclipse Public License. Eclipse provides a range of features and tools that help developers to write, test, and debug code more efficiently. It includes a code editor, a debugger, a build automation tool, and a visual user interface builder. Eclipse also supports version control systems like Git and Subversion and offers integrated support for unit testing frameworks like JUnit.

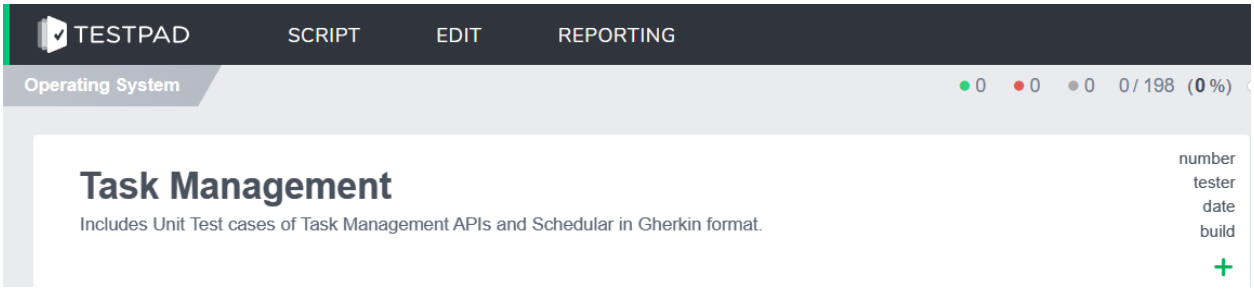
One of the most significant advantages of Eclipse is its extensibility. Developers can customize the IDE by installing plugins that provide additional functionality, such as support for other programming languages, frameworks, and tools. Eclipse's plugin ecosystem includes thousands of plugins developed by a large community of developers.

6.8 TestPad Tool

TestPad is an execution notepad feature that houses all the test-run details and results. Using TestPad you can view test case details, insert and view attachments also it adds all notes in one locatio0n while you are executing tests.
TestPad is divided into:

- TestPad Header
- Execute Steps
- Test Run
- Test Case Details
- Attachments
- Notes
- Defects

6.8.1 TestPad Header



The TestPad Header provides information about the test run you are about to execute.

- Test RunID
- Test RunName

6.8.2 Execute Steps Tab

The Execute Steps Tab is where the information for the Test Case, and then record the results, statues, and defects for Test Steps. A rich-text editor is available in the Test Step fields to provide formatting options such as bold, underline, and text color.

Set Status: After marking the execution status for the Test Steps, then set the status of the overall Test Log in the Set Status field.

- Submit Defect: In addition to being able to submit Defects for a Test Step, also submitting a Defect for the overall Test Log.
- Test Case Attachment (paperclip icon): allows to add an attachment for the Test Case associated with the Test Step. When an attachment is available for the Test Case, the paperclip icon and attachment count will display in the column in line with the appropriate Test Step number.
- Description: description of what the Test Step is accomplishing
- Expected Result: what should happen when the Test Step executes.

- Actual Result: what does happen when the Test Step executes.
- Test Log Attachment (Clip icon/Log): allow to add an attachment for the individual Test Step Log. When an attachment is available for the Test StepLog, the paperclip icon and attachment count will display in the column in line with the appropriate Test Step number.

6.8.3 Test Run Tab

The Test Run tab can allow to view and modify Test Run properties while executing a Test Run in the TestPad. The changes will be saved to the latest Test Run properties and the executed Test Log.

- Test Case Details Tab
- The Test Case Details tab is read-only and provides information entered in Test Design.

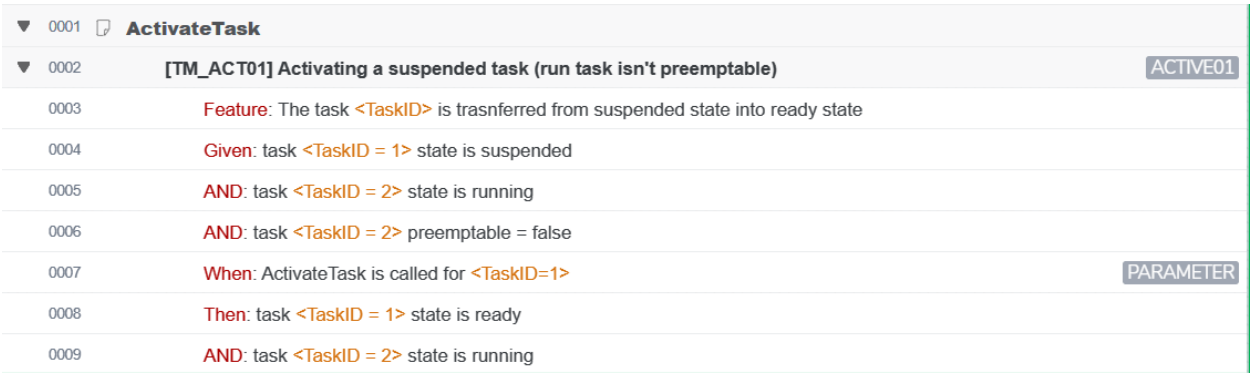
6.8.4 Attachments Tab

The Attachments tab allow to add new attachments for the Test Case or the Test Log. The Attachments Table and Attachment Preview provide additional information for the attachment.

In our test cases we used Gherkin format where:

Gherkin is a plain-text language with a simple structure. It is designed to be easy to learn for non-programmers yet structured enough to allow concise description of test scenarios and examples to illustrate business rules in most real-world domains.

Here is a sample Gherkin test case:



Here this test case form a sequence of steps in implementing the test case where we test a scenario about activating a task that wasn't suspended then creating a steps about how to implement these test cases and then after the implementation test the should either fail or succeed then go to the scenario and put that this case succeeded or failed and then re-implementing the wrong syntax and re-test again.

An example for implementation the previous case:

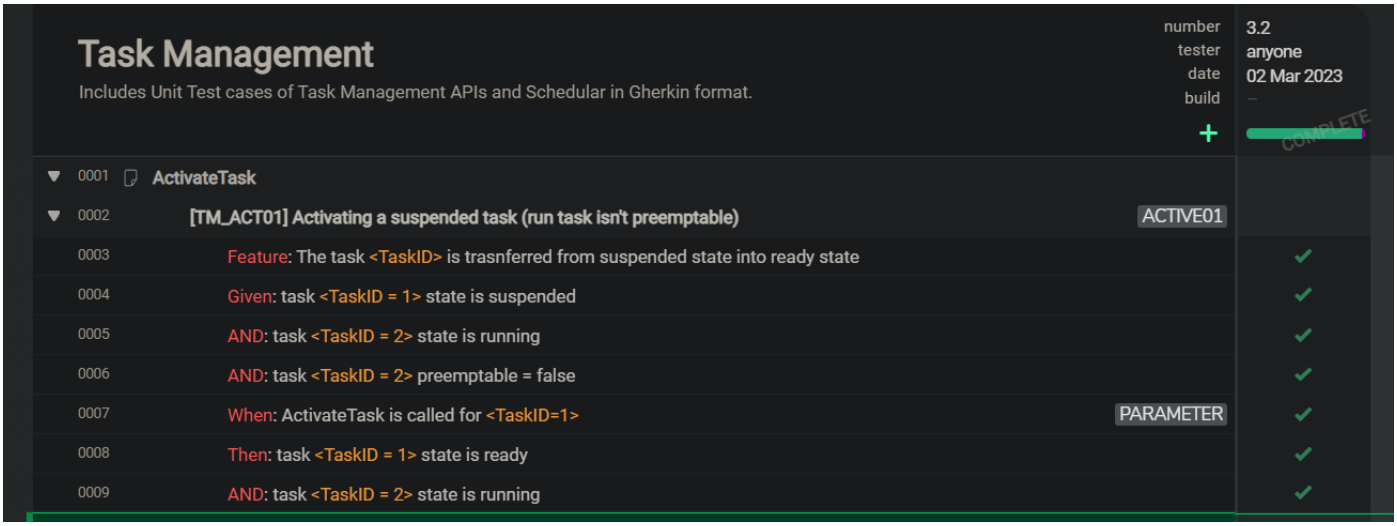
```
/*****************************
 * CODE: [TM_ACT01]
 * Summary: Activating a suspended task (run task isn't pre-empt
able)
 * Feature: The task <TaskID> is transferred from suspended stat
e into ready state
 * Given: task <TaskID = 8> state is suspended
 * AND: task <TaskID = 9> state is running
 * AND: task <TaskID = 9> pre-emptable = false
 * When: ActivateTask is called for <TaskID=8>
 * Then: task <TaskID = 8> state is ready
 * AND: task <TaskID = 9> state is running
 *****************************/
void Active01(){
    resetTasks();

    /**Given**/
    task[8].state = SUSPENDED;
    task[9].state = RUNNING;
    task[9].preemption = FALSE;
    createTasks(task, 2);

    /**When**/
    assert(tcb_tstat[9] == RUNNING);
    ActivateTask(8);

    /**THEN**/
    assert(tcb_tstat[8] == READY);
    assert(tcb_tstat[9] == RUNNING);
}
```

After completion of the test case and being succeeded then check pass for this test case as following:



6.9 Configuration Tool

6.9.1 Introduction

The Configuration Tool is a user-friendly graphical user interface (GUI) developed using Microsoft Excel, coupled with a Python script, that allows users to easily define their operating system (OS) objects without requiring in-depth knowledge of the underlying OS concepts. This documentation aims to provide a comprehensive guide on how to effectively utilize the Configuration Tool for generating ".c" files for compilation with our OS.

6.9.2 Tool Overview

The Configuration Tool consists of three main components:

- Excel Interface: The Excel workbook serves as the front-end GUI for users to input their desired OS object configurations.
- Python Script: The Python script acts as the backend engine that processes the user input from the Excel workbook and modifies a Jinja template file.
- Jinja Template: The template file is used as a blueprint to generate the .c files based on the user-defined configurations.

6.9.3 Usage Guide

Follow the steps below to effectively use the Configuration Tool:

Step 1: Launch the Configuration Tool

- Open the Excel workbook provided in the tool package.

Step 2: Input Configuration Data

- Fill in the appropriate fields in the Excel workbook according to the provided instructions or guidelines.

Step 3: Run the Python Script

- Double click on OZEKAA.exe this will run the Python script.

Step 4: Generate .c Files

- The Python script will process the input data and modify the Jinja template file accordingly.
- The modified template will be used to generate the desired .c files.

Step 5: Compilation

Once the .c files are generated, you can now compile this configuration file with OS Kernel.

6.9.4 Benefits

The Configuration Tool offers several benefits for both users and developers:

1. User-Friendly Interface: The tool utilizes a familiar and user-friendly Excel interface, making it easy for users to input their desired configurations without requiring extensive knowledge of the underlying OS concepts. This lowers the barrier to entry and enables a broader range of users to define their OS objects effectively.
2. Simplified Configuration Process: With the Configuration Tool, users no longer need to manually modify complex configuration files or understand intricate syntax. The tool abstracts away the technical details and provides a structured, guided approach to defining OS objects, reducing the chances of errors or inconsistencies.
3. Increased Efficiency: By automating the generation of .c files based on user-defined configurations, the Configuration Tool significantly improves efficiency. Users can quickly iterate and experiment with different configurations, making it easier to fine-tune and customize the OS objects without having to manually modify files or understand the internals of the system.
4. Error Reduction: The Configuration Tool helps mitigate errors that may arise from manual configuration by providing input validation mechanisms. Users are guided to provide correct and valid data through predefined constraints and data type checks within the Excel interface. This reduces the likelihood of mistakes, resulting in more reliable and accurate configurations.
5. Reusability and Maintainability: The separation of the Excel interface, Python script, and Jinja template file promotes code modularity and reusability. The tool can be easily extended or modified to accommodate future changes in the OS or additional features. This modular design simplifies maintenance efforts and allows for better scalability as the project evolves.

6.9.5 Conclusion

The Configuration Tool provides a simplified way for users to define their OS objects without needing to possess in-depth knowledge of the underlying OS concepts. By leveraging the Excel GUI and the Python script, users can easily generate .c files for compilation with our OS.

Input

	Name	Priority	Preemption	Max_Activation	Auto_Start	Stack_Size	Events	Resources
2	HashingTask	3	1	3	0	512	HashingComplete	HashingModule
3	EncryptionAESTask	8	0	2	0	512		EncryptionModule & PKCModule
4	DecryptionAESTask	6	1	2	0	512		DecryptionModule & PKCModule
5	AuthenticationTask	15	0	1	1	1024	AuthenticationSuccess &AuthenticationFailure	AuthenticationResource
6	KeyGenerationTask	10	1	5	1	1024		PKCModule & TRNGModule
7	EncryptionRSATask	7	0	2	0	512	EncryptionComplete	EncryptionModule
8	DecryptionRSATask	5	1	2	0	512	DecryptionComplete	DecryptionModule

Task | Alarm | Counter

	Name	Action	ActionInfo	AutoStart	AlarmTime	CycleTime
2	EncryptionTimeOut	INCREMENT	SW_Count1	0	0	0
3	DecryptionTimeOut	ACTIVATETASK	DecryptionRSATask	0	0	0
4	KeyExpiryAlarm	CaLLBACK	Key_CallBack	1	50	500
5	AuthenticationAlarm	ACTIVATETASK	AuthenticationTask	0	0	0
6	PerodicHashing	SETEVENT	HashingComplete & HashingTask	1	30	200

Task | Alarm | Counter

	A	B	C	D	E
1	Name	CounterType	MinCycle	Maximum	Alarms
2	HW_C0	HW	0	500	PerodicHashing
3	HW_C1	HW	0	400	KeyExpiryAlarm
4	SW_Count1	SW	0	300	AuthenticationAlarm
5	SW_Count2	SW	0	200	EncryptionTimeOut & DecryptionTimeOut & PerodicHashing
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					
21					
22					
23					

Task | Alarm | Counter

Output

```
extern TASK(AuthenticationTask);
#define STACK_SIZE_T0        1024

extern TASK(DecryptionRSATask);
#define STACK_SIZE_T1        512

extern TASK(EncryptionRSATask);
#define STACK_SIZE_T2        512

extern TASK(HashingTask);
#define STACK_SIZE_T3        512

extern TASK(EncryptionAESTask);
#define STACK_SIZE_T4        512

extern TASK(DecryptionAESTask);
#define STACK_SIZE_T5        512

extern TASK(KeyGenerationTask);
#define STACK_SIZE_T6        1024
```

```
#define EXTENDED_STATUS
#define TASK_COUNT              7
#define EXT_TASK_COUNT          4
#define RES_COUNT               6

#define COUNTER_HW_COUNT        2
#define ALARMS_COUNT            5
#define COUNTER_COUNT           4
#define ALARMS_AUTOSTART_COUNT  2
```

```
const Priority          tinib_inipri[TASK_COUNT] =  { 6, 1, 3, 0, 4, 2, 5 };

const Priority          tinib_exepri[TASK_COUNT] = { 15, 1, 15, 0, 15, 2, 5 };

const UINT8             tinib_maxact[TASK_COUNT] = { 1, 2, 2, 3, 2, 2, 5 };

const BOOL              tinib_autoact[TASK_COUNT] = { 1, 0, 0, 0, 0, 0, 1 };

const EventMaskType     tinib_mskevt[EXT_TASK_COUNT] = { 3, 4, 8, 16 };

const Priority          resinib_ceilpri[RES_COUNT] = { 6, 2, 4, 0, 5, 5 };

const ResourceType      resinib_mskres[TASK_COUNT] = { 1, 2, 4, 8, 64, 32, 512 };
```

```
1  const CounterType alrmini_cnt[ALARMS_COUNT]={ SW_Count2, HW_C1, SW_Count1, SW_Count2, SW_Count2 };
2  const ActionType alrmini_act[ALARMS_COUNT]= { 3, 2, 2, 1 };
3  const ActionInfoType alrmini_actInfo[ALARMS_COUNT]={ {0,0,0,SW_Count1},{ DecryptionRSATask},{ AuthenticationTask},{ HashingComplete , HashingTask} };
4
5  const UINT8 cntini_AlrmNum[COUNTER_COUNT]={ 1, 1, 1, 3 };
6  const UINT32 cntini_AlrmRef[COUNTER_COUNT]={ 16, 4, 8, 19 };
7  const TickType cntini_maxVal[COUNTER_COUNT]= { 500, 400, 300, 200 };
8  const TickType cntini_minCyc[COUNTER_COUNT]= { 0, 0, 0, 0 };
9  const TickType cntini_ticksPBase[COUNTER_COUNT]= { 1,1,1,1 };
10
11 const AutoStartAlarmType alrm_autostrt[ALARMS_AUTOSTART_COUNT] = { {KeyExpiryAlarm,50,500}, {PerodicHashing,30,200} };
```

7. References

1. OSEK/VDX Operating System Specification 2.2.3
https://www.irisa.fr/alf/downloads/puaut/TPNXT/images/os223.pdf

2. *"Free OSEK real-time kernel for dsPIC | Microchip"*. *www.microchip.com. Retrieved 2021-10-05*

3. "nxtOSEK/JSP: ANSI C/C++ with OSEK/μITRON RTOS for LEGO MINDSTORMS NXT". *lejos-osek.sourceforge.net*. Retrieved 2021-10-05.

4. *"TOPPERS Project／ATK2"*. *www.toppers.jp. Retrieved 2021-10-07.*

5. *"MICROSAR | Vector"*. *www.vector.com. Retrieved 2021-10-07*

6. Evertsson, Pontus (2004). *"Investigation of Real-Time Operating Systems: OSEK/VDX and Rubus"*. MSC Theses. *ISSN 0280-5316*.

7. Feiler, Peter H. (2018). "Real-Time Application Development with OSEK: A Review of the OSEK Standards"