Software Integration test specification

Release 0.1

intel

Jun 19, 2020

ZEPHYR INTERGRATION TEST SPEC

CHAPTER

ONE

OVERRALL

Zephyr Functional Safety Certification

This is a software architecture design test specification for zephyr RTOS prject. This document provides a generic verification specification suitable for use in IEC 61508 and ISO 26262 compliant developments.

Acronyms and Terminology:

Term	Description
RTOS	real time operate system

Revision History:

Data	Revision	Auther	Description
2020/06/19	1.0	Zhu Youhua	Initial draft version.

Applicable Standards:

Document	Document Number / Location
IEC 61508-2:2010, Functional safety of electrical/electronic/ programmable electronic safety-related-systems – Part 2: Requirements for electrical/ electronic/programmable electronic safety-related systems	www.iec.ch
IEC 61508-3:2010,Functional safety of electrical/electronic/ programmable electronic safety-related-systems – Part 3: Software requirements	www.iec.ch
ISO 26262-8:2011, Road vehicles — Functional safety — Part 8: Supporting processes	www.iso.org

Reference Documents: https://zephyrproject.org/

```
No license (express or implied, by estoppel or otherwise) to any intellectual␣
↪property rights is granted by this document.
All information provided here is subject to change without notice. Contact your Intel␣
↪representative to obtain the latest Intel product specifications and roadmaps.
The products described may contain design defects or errors known as errata which may␣
↪cause the product to deviate from published specifications. Current characterized␣
↪errata are available on request.
Copies of documents which have an order number and are referenced in this document␣
↪may be obtained by calling 1-800-548-4725 or by visiting:  http://www.intel.com/
↪design/literature.htm
```

(continues on next page)

(continued from previous page)

```
Intel technologies' features and benefits depend on system configuration and may␣
↪require enabled hardware, software or service activation. Performance varies␣
↪depending on system configuration. No computer system can be absolutely secure.␣
↪Check with your system manufacturer or retailer or learn more at [intel.com].
This material may relate to the creation of end products used in safety-critical␣
↪applications designed to comply with functional safety standards or requirements␣
↪(“Safety-Critical Applications”).  You agree and represent that you have all the␣
↪necessary expertise to design, manage and ensure effective system-level safeguards␣
↪to anticipate, monitor and control system failures in safety-critical applications.␣
↪It is your sole responsibility to design, manage and assure system-level safeguards␣
↪to anticipate, monitor and control system failures, and you agree that you are␣
↪solely responsible for all applicable regulatory standards and safety-related␣
↪requirements concerning your use of any material related to Safety Critical␣
↪Applications.  You agree to indemnify and hold Intel and its representatives␣
↪harmless against any damages, costs, and expenses arising in any way out of your␣
↪use of the material related to Safety-Critical Applications.  You further agree␣
↪that some of the material maybe be pre-production in nature and that all material␣
↪is provided "as is" without any express or implied warranty of any kind including␣
↪warranties of merchantability, noninfringement, or fitness for a particular purpose.
↪ intel does not warrant or assume responsibility for the accuracy or completeness␣
↪of any material provided.
Intel, Intel brands, and the Intel logo are trademarks of Intel Corporation in the U.
↪S. and/or other countries.
*Other names and brands may be claimed as the property of others.
Copyright © 2020, Intel Corporation. All rights reserved.
```

CHAPTER

TWO

ENVIRONMENT

Description of intergration testing environment, including software, hardware environment.

1. In Scope:

This document is intended to address the following functional safety requirements.

- **IEC 61508-2:2010 (systems/hardware development)** 7.9.2.3, 7.9.2.4
- **IEC 61508-3:2010 (software development)** 7.9.2.6, 7.9.2.7
- **ISO 26262-8:2011 (verification supporting process)** 9.4.2.1, 9.4.2.2, 9.4.2.3

2. Out of Scope:

Information specific to the verification plan and verification report are not addressed in this document.

3. Setup environment:

We use a script named sanitycheck to run intergration testcases. This script scans for the set of unit test applications in the git repository and attempts to execute them. By default, it tries to build each test case on boards marked as default in the board definition file.

The default options will build the majority of the tests on a defined set of boards and will run in an emulated environment if available for the architecture or configuration being tested.

In normal use, sanitycheck runs a limited set of kernel tests (inside an emulator). Because of its limited test execution coverage, sanitycheck cannot guarantee local changes will succeed in the full build environment, but it does sufficient testing by building samples and tests for different boards and different configurations to help keep the complete code tree buildable.

When using (at least) one `-v` option, sanitycheck's console output shows for every test how the test is run (qemu, native_posix, etc.) or whether the binary was just built.

To run the script in the local tree, follow the steps below:

```
$ source zephyr-env.sh
$ ./scripts/sanitycheck
```

If you have a system with a large number of cores, you can build and run all possible tests using the following options:

```
$ ./scripts/sanitycheck --all --enable-slow
```

This will build for all available boards and run all applicable tests in a simulated (for example QEMU) environment.

The list of command line options supported by sanitycheck can be viewed using:

```
$ ./scripts/sanitycheck --help
```

CHAPTER

THREE

INTERGRATION TEST CASES

3.1 Mutex

group **kernel_mutex_tests**
Test for the Mutex kernel object.

To test the all the function and mechnism for mutex, it contains below several test cases.

Functions

K_MUTEX_DEFINE (kmutex)
Test mutex initialization in build time.

To verify a mutex can be defined and initialized at compile time, this is the TESTPOINT of init via K_MUTEX_DEFINE.

Return should be PASS.

void **test_mutex_reent_lock_forever** (void)
Test mutex initialization and locking forever case.

- To verify a mutex can be defined and initialized at compile time or run time.
- To verify that getting a lock of a mutex already locked, waiters will be blocked forever.

Return should be PASS.

void **test_mutex_reent_lock_no_wait** (void)
Test mutex initialization and locking for no wait case.

- To verify a mutex can be defined and initialized at compile time or run time.
- To verify that getting a lock of a mutex already locked, waiters will not wait for it.

Return should be PASS.

void **test_mutex_reent_lock_timeout_fail** (void)
Test mutex initialization and locking for timeout failed case.

- To verify a mutex can be defined and initialized at compile time or run time.
- To verify that getting a lock of a mutex already locked, waiters will wait for a period of time and fail to get it.

Return should be PASS.

void **test_mutex_reent_lock_timeout_pass**(void)
Test mutex initialization and locking for timeout successful case.

- To verify a mutex can be defined and initialized at compile time or run time.
- To verify that getting a lock of a mutex already locked will succeed and waiters will be unblocked only when the number of locks reaches zero.

Return should be PASS.

void **test_mutex_lock_unlock**(void)
Test mutex set amount of time the thread will wait.

To verify the mutex lock operation shall accept a timeout parameter indicating the maximum amount of time the thread will wait.

Return should be PASS.

void **test_mutex_recursive**(void)
Test: recursive mutex

Describe:

- To verify that getting a lock of a mutex already locked will succeed and waiters will be unblocked only when the number of locks reaches zero.

Return should be PASS.

void **test_mutex_priority_inheritance**(void)
Test mutex's priority inheritance mechanism.

To verify mutex provide priority inheritance to prevent prority inversion, and there are 3 cases need to run. The thread T1 hold the mutex first and cases list as below:

- case 1. When prority T2 > T1, priority inheritance happened.
- case 2. When prority T1 > T2, priority inheritance won't happened.
- case 3. When prority T2 > T3 > T1, priority inheritance happened but T2 wait for timeout and T3 got the mutex.

Return should be PASS.

3.2 Semaphore

Testing of semaphore.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

K

T