Zephyr™ Project

Developer Summit

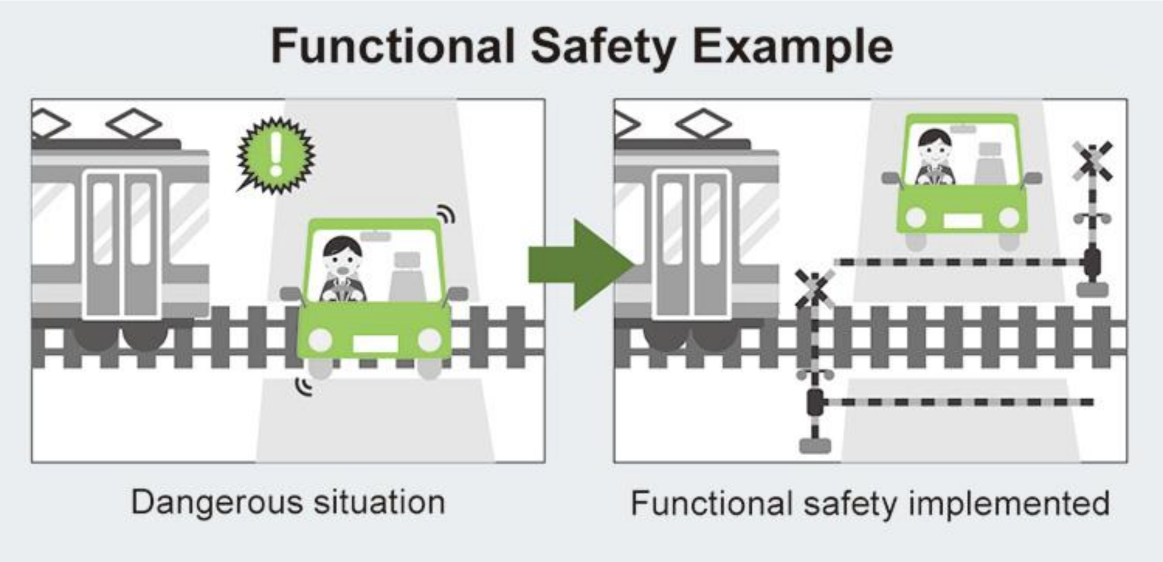
June 8-10, 2021 ▪ @ZephyrIoT

Functional Safety Verification & Validation Test Case Development and the Challenges

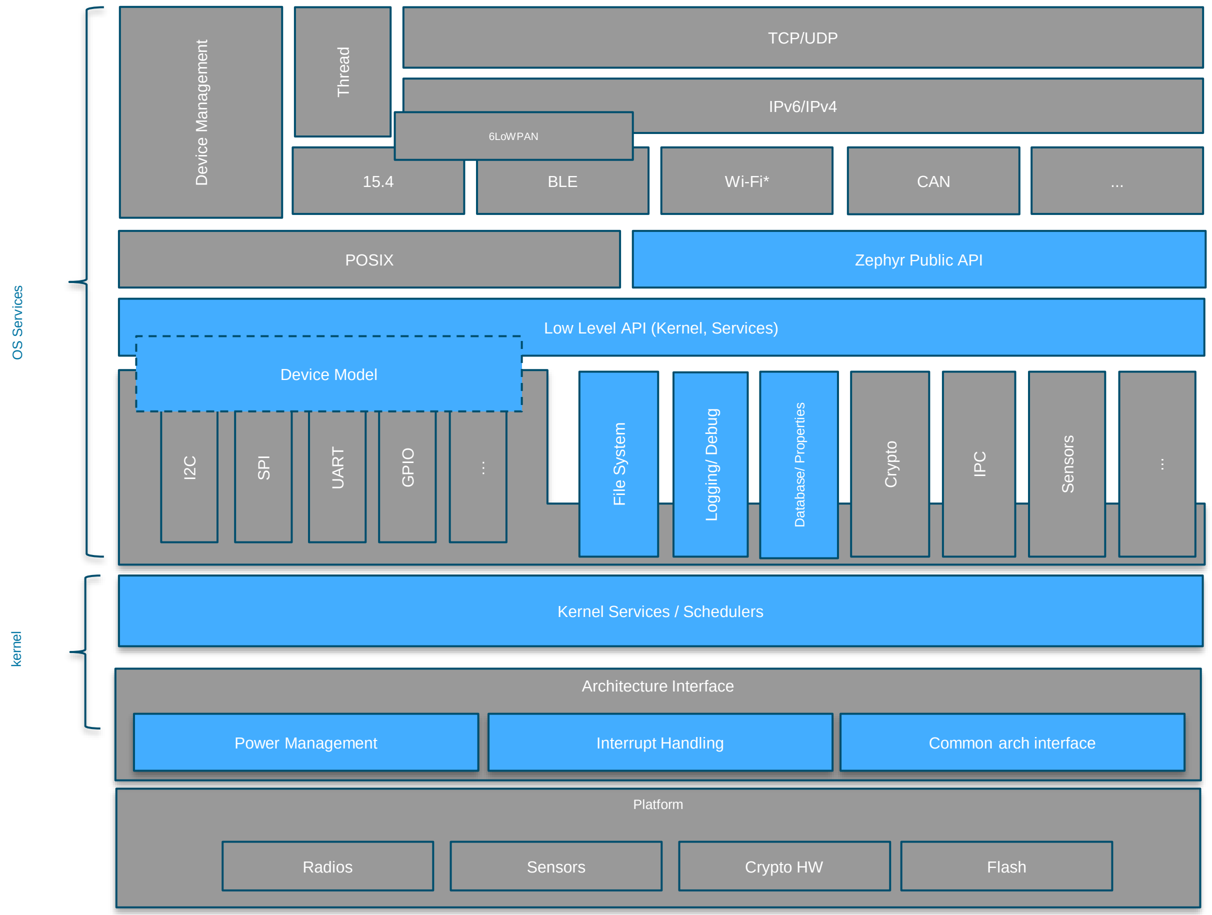
STEVEN WANG, INTEL

Functional Safety (FuSa)

- The objective of functional safety is freedom from unacceptable risk of physical injury or of damage to the health of people either directly or indirectly (through damage to property or to the environment) by the proper implementation of one or more automatic protection functions. (IEC61508)



Zephyr FuSa Certificate Scope



- In zephyr, it includes not only kernel, but also other services, which are needed for application development.
- Currently, only the components in blue are in Zephyr FuSa certificate.

Verification & Validation V-Mode

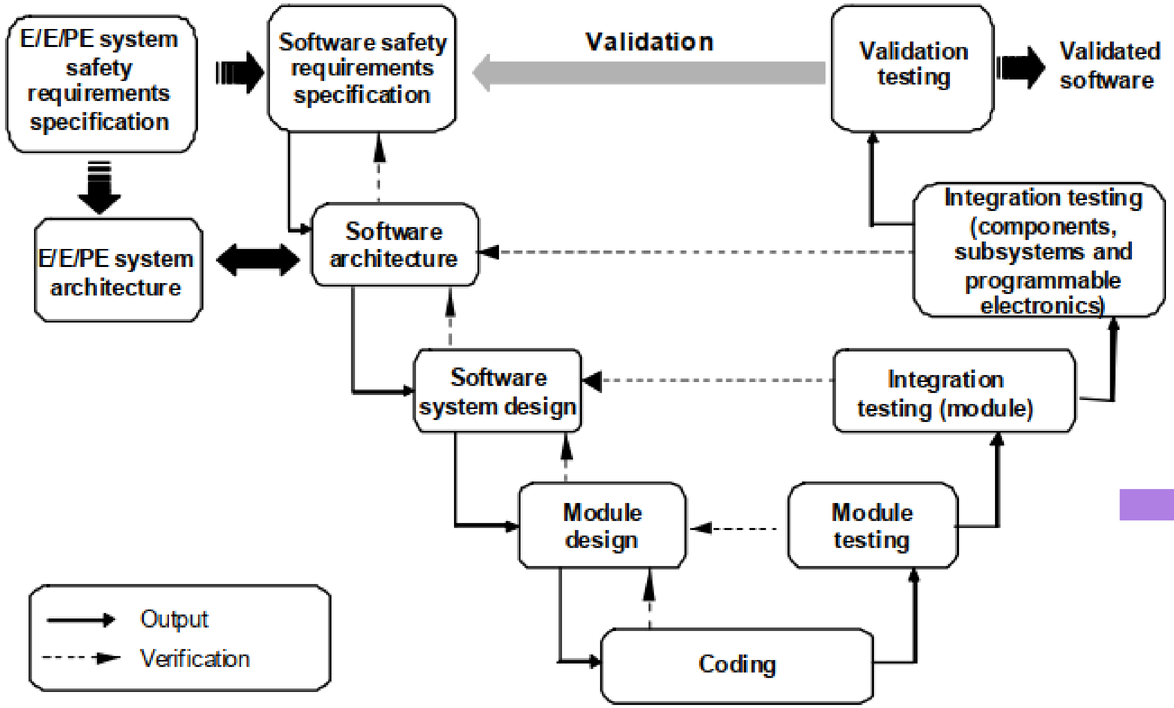
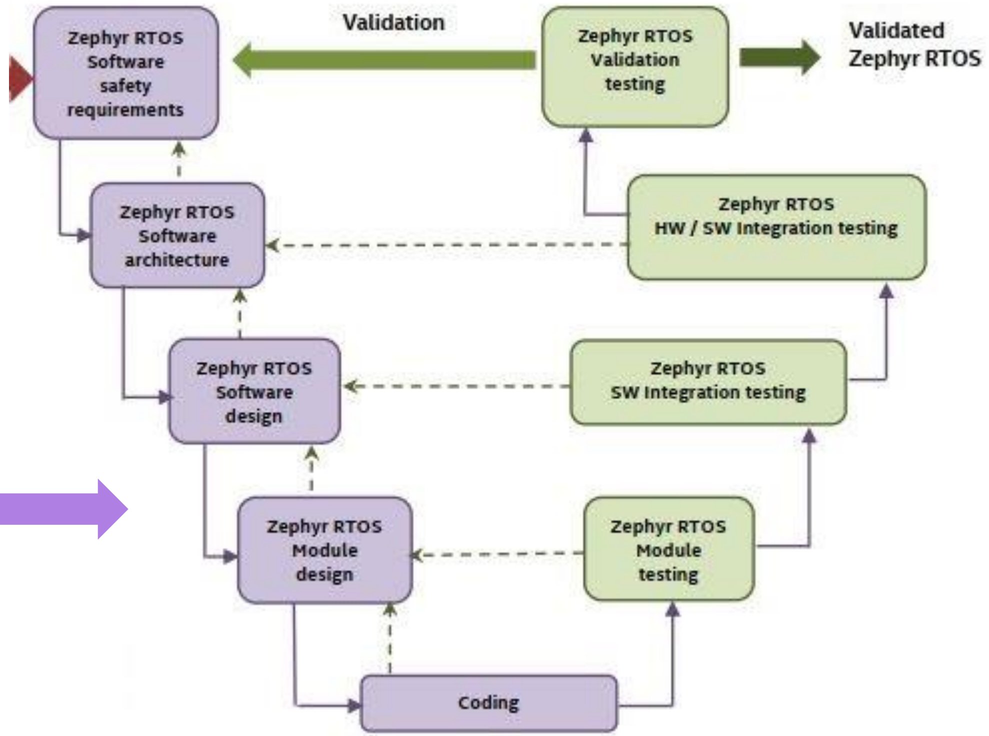


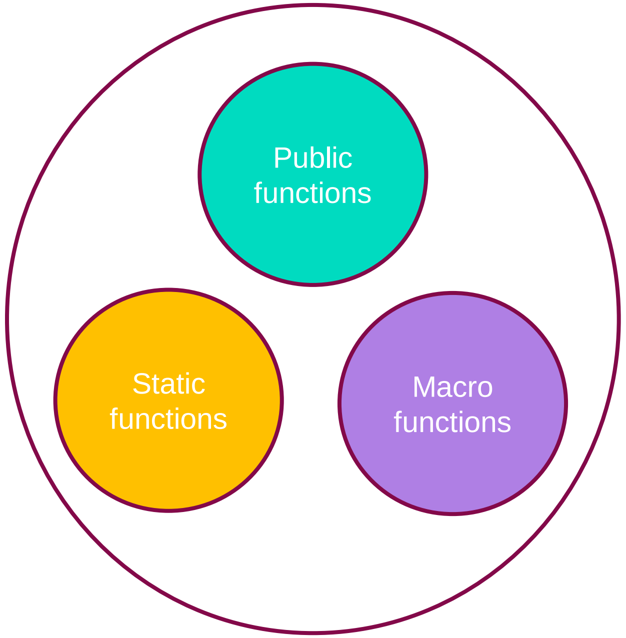
Figure 6 – Software systematic capability and the development lifecycle (the V-model)



Zephyr Verification & Validation V-mode

Module Testing

- Objectives
  - Verify all functions of a module to ensure that they follow the module designs, and no unintended code exists. It intends to verify all both public and internal functions.
  - Line, function and branch coverage reaches 100%.

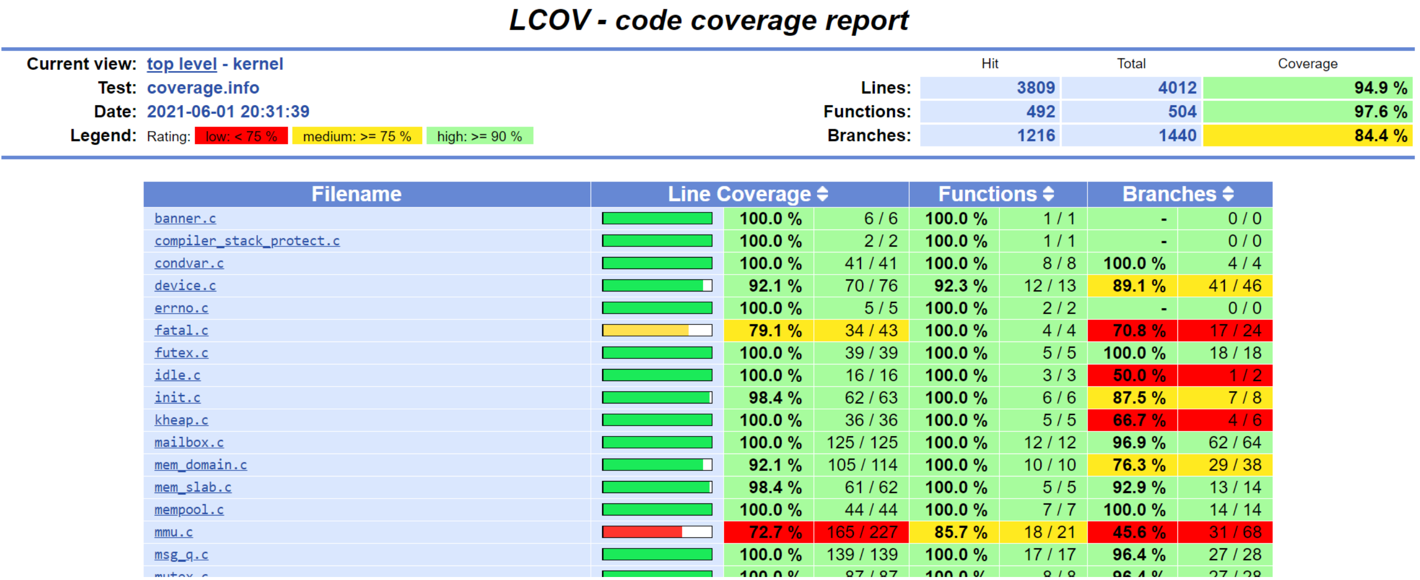


Module A

Module Test Case Design

- Test cases based on boundary value analysis.
  - For example, there is function “void foo(unsigned short a)”, the maximum value 65535 of “a” should be tested.
- Test cases based on Equivalence classes and input partition.
  - Equivalence classes derived from the specification may be either input orientated, for example the values selected are treated in the same way, or output orientated, for example the set of values lead to the same functional result.
  - Equivalence classes derived from the internal structure of the program – the equivalence class results are determined from static analysis of the program, for example the set of values leading to the same path being executed.

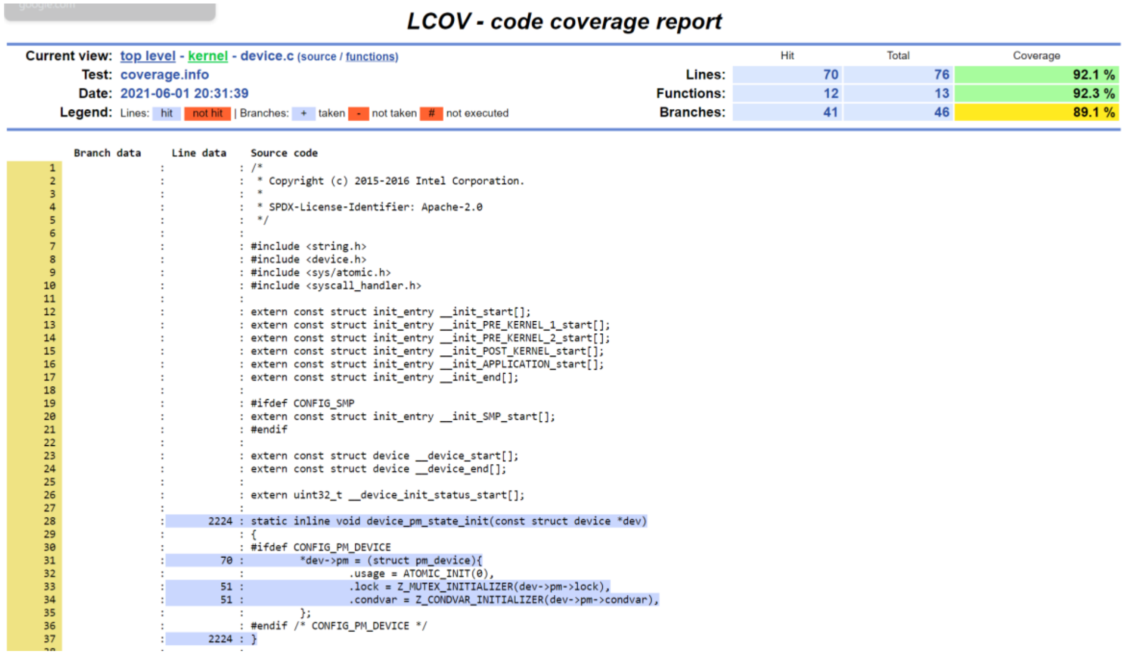
Code Coverage



LCOV - code coverage report

			Hit	Total	Coverage
Current view:	top level - kernel	Lines:	3809	4012	94.9 %
Test:	coverage.info	Functions:	492	504	97.6 %
Date:	2021-06-01 20:31:39	Branches:	1216	1440	84.4 %
Legend:	Rating: low: < 75 % medium: >= 75 % high: >= 90 %				

Filename	Line Coverage		Functions		Branches	
banner.c	100.0 %	6 / 6	100.0 %	1 / 1	-	0 / 0
compiler_stack_protect.c	100.0 %	2 / 2	100.0 %	1 / 1	-	0 / 0
condvar.c	100.0 %	41 / 41	100.0 %	8 / 8	100.0 %	4 / 4
device.c	92.1 %	70 / 76	92.3 %	12 / 13	89.1 %	41 / 46
errno.c	100.0 %	5 / 5	100.0 %	2 / 2	-	0 / 0
fatal.c	79.1 %	34 / 43	100.0 %	4 / 4	70.8 %	17 / 24
futex.c	100.0 %	39 / 39	100.0 %	5 / 5	100.0 %	18 / 18
idle.c	100.0 %	16 / 16	100.0 %	3 / 3	50.0 %	1 / 2
init.c	98.4 %	62 / 63	100.0 %	6 / 6	87.5 %	7 / 8
kheap.c	100.0 %	36 / 36	100.0 %	5 / 5	66.7 %	4 / 6
mailbox.c	100.0 %	125 / 125	100.0 %	12 / 12	96.9 %	62 / 64
mem_domain.c	92.1 %	105 / 114	100.0 %	10 / 10	76.3 %	29 / 38
mem_slab.c	98.4 %	61 / 62	100.0 %	5 / 5	92.9 %	13 / 14
mempool.c	100.0 %	44 / 44	100.0 %	7 / 7	100.0 %	14 / 14
mmu.c	72.7 %	165 / 227	85.7 %	18 / 21	45.6 %	31 / 68
msg_q.c	100.0 %	139 / 139	100.0 %	17 / 17	96.4 %	27 / 28



LCOV - code coverage report

			Hit	Total	Coverage
Current view:	top level - kernel - device.c (source / functions)	Lines:	70	76	92.1 %
Test:	coverage.info	Functions:	12	13	92.3 %
Date:	2021-06-01 20:31:39	Branches:	41	46	89.1 %
Legend:	Lines: hit not hit | Branches: + taken - not taken # not executed				

```
Branch data     Line data    Source code
            :            : /*
            :            :  * Copyright (c) 2015-2016 Intel Corporation.
            :            :  *
            :            :  * SPDX-License-Identifier: Apache-2.0
            :            :  */
            :            :
            :            : #include <string.h>
            :            : #include <device.h>
            :            : #include <sys/atomic.h>
            :            : #include <syscall_handler.h>
            :            :
            :            : extern const struct init_entry __init_start[];
            :            : extern const struct init_entry __init_PRE_KERNEL_1_start[];
            :            : extern const struct init_entry __init_PRE_KERNEL_2_start[];
            :            : extern const struct init_entry __init_POST_KERNEL_start[];
            :            : extern const struct init_entry __init_APPLICATION_start[];
            :            : extern const struct init_entry __init_end[];
            :            :
            :            : #ifdef CONFIG_SMP
            :            : extern const struct init_entry __init_SMP_start[];
            :            : #endif
            :            :
            :            : extern const struct device __device_start[];
            :            : extern const struct device __device_end[];
            :            :
            :            : extern uint32_t __device_init_status_start[];
            :            :
            :       2224 : static inline void device_pm_state_init(const struct device *dev)
            :            : {
            :            : #ifdef CONFIG_PM_DEVICE
            :         70 :         *dev->pm = (struct pm_device){
            :            :                 .usage = ATOMIC_INIT(0),
            :         51 :                 .lock = Z_MUTEX_INITIALIZER(dev->pm->lock),
            :         51 :                 .condvar = Z_CONDVAR_INITIALIZER(dev->pm->condvar),
            :            :         };
            :            : #endif /* CONFIG_PM_DEVICE */
            :       2224 : }
```

- In zephyr project, Lcov and Gcov tools are used for the code coverage report. They have been integrated with the twister.
  - twister -p qemu_x86 -C -T tests/xxxxx

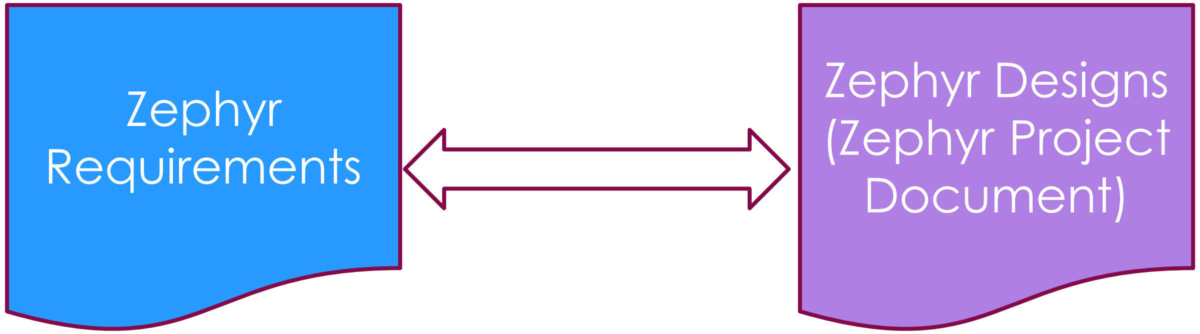
Integration Test Case Design

- Integration test cases are designed based on Zephyr project document. All the features of Zephyr shall be covered.
- Design Approaches
  - Functional and black box testing
  - Performance testing, such as thread switch performance, memory footprint, etc.
  - Fault injection testing. It is to inject some known faults to see if they can be handled by the error or exception handling mechanisms.

Validation Test Case Design

- In Zephyr FuSa certificate, Zephyr is a pre-existing project. The Zephyr requirements will be derived from the existing features. They shall be validated by the corresponding test cases.
- Design Approaches
  - Functional and black box testing

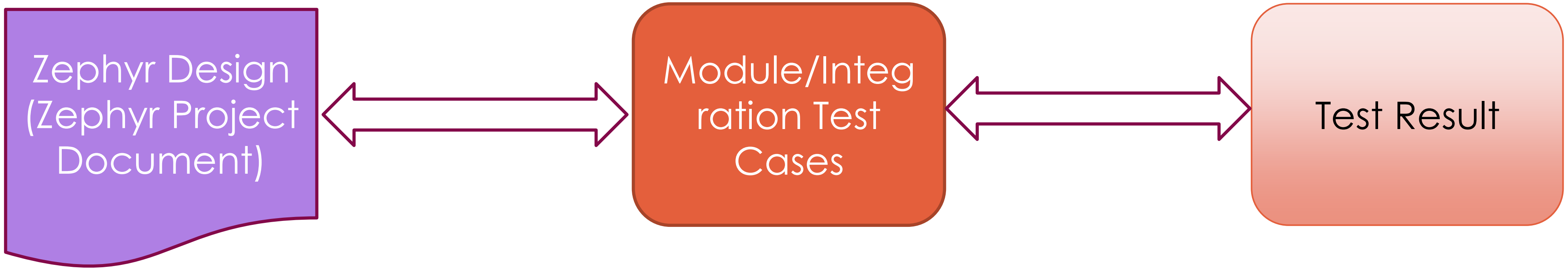
Traceability



- It shows that Zephyr requirements are fulfilled by Zephyr architecture and module designs, which are in Zephyr project document.

- It shows that Zephyr requirements are validated by validation test cases, and if test cases are passed.

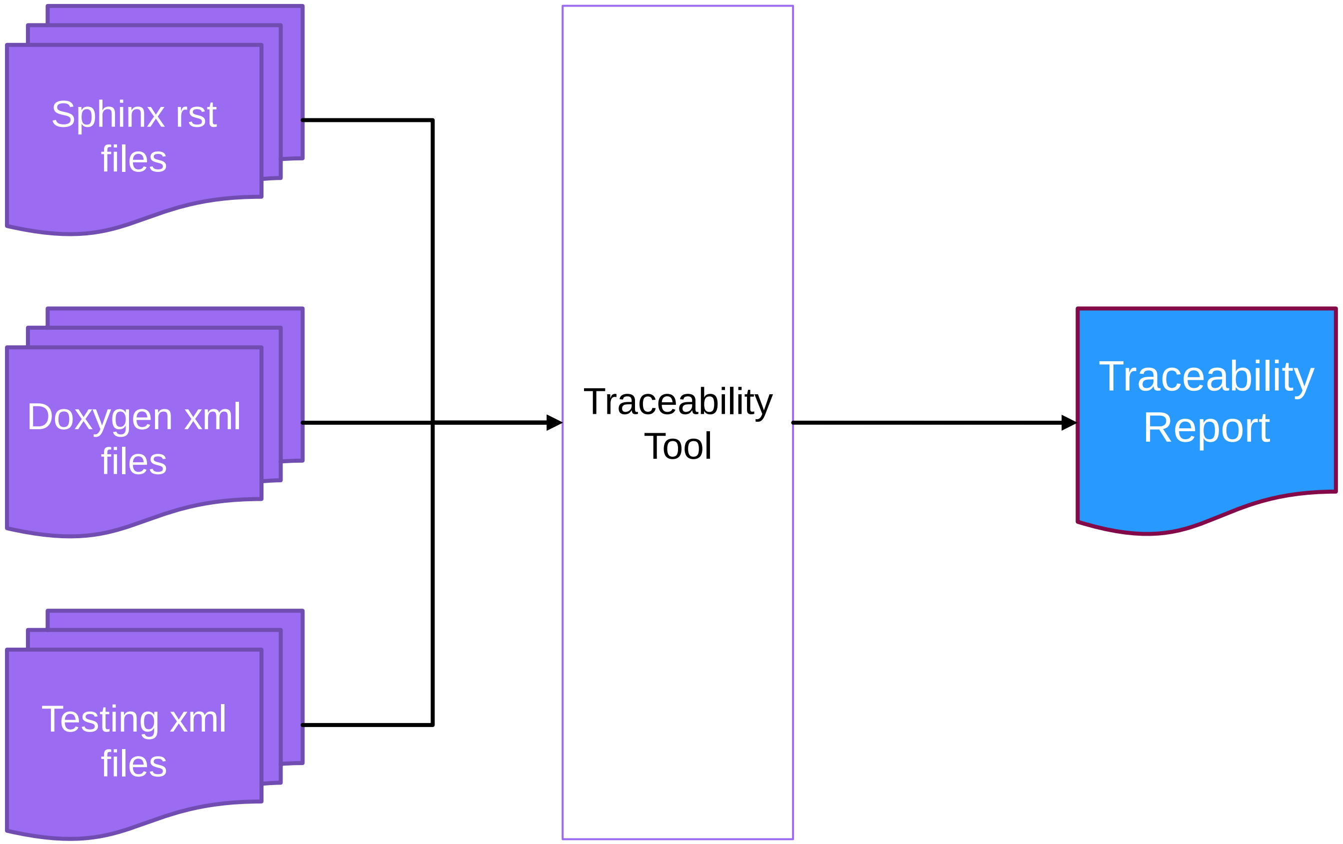
Traceability



- It shows that all features and functions of Zephyr have been verified by module and integration test cases, and if all the test cases are passed.

Traceability Tool

- The tool is designed to link sphinx rst, doxygen xml and testing xml files for traceability.



- Zephyr Project Document is generated with Sphinx rst and Doxygen xml files.
- The testing documents such as the testing plan and specification are based on Sphinx rst.
- Zephyr testing result is a xml file.

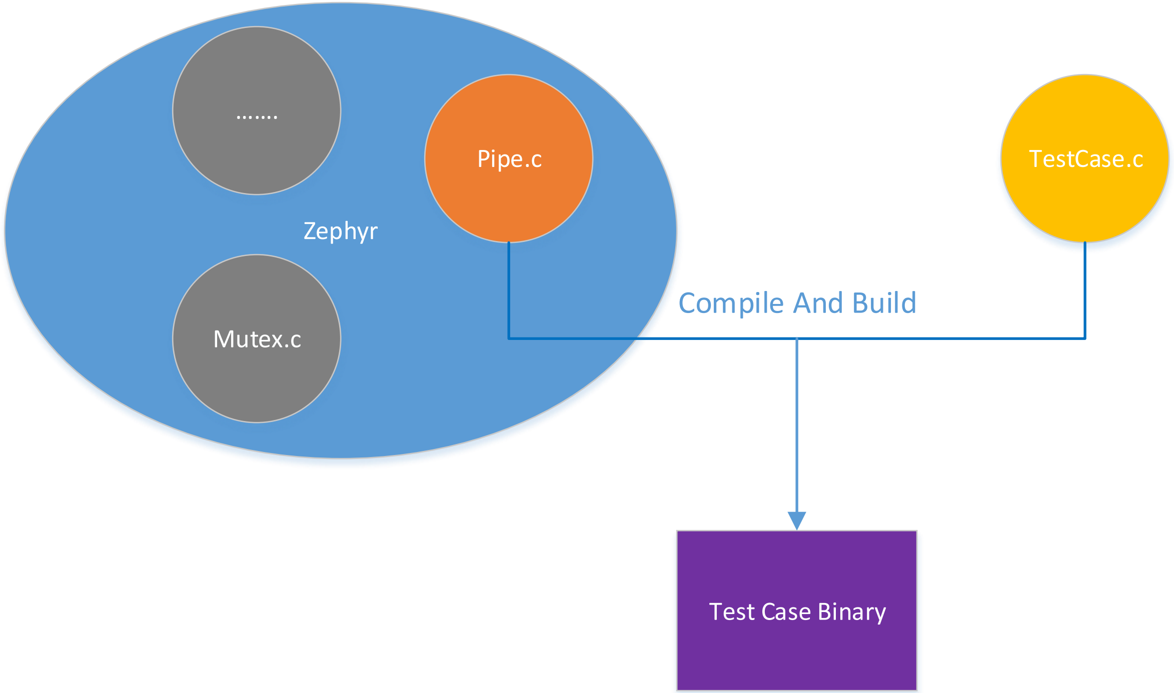
The Challenges

- It is not easy to test the internal functions, such as “static c functions in a .c file. Currently, we test them indirectly via the public functions.

```c
static int pipe_return_code(size_t min_xfer, size_t bytes_remaining,
                             size_t bytes_requested)
{
        if (bytes_requested - bytes_remaining >= min_xfer) {
                /*
                 * At least the minimum number of requested
                 * bytes have been transferred.
                 */
                return 0;
        }

        return -EAGAIN;
}
```

- For module testing, most of Zephyr kernel code can not be tested in an isolation way.



FuSa And Zephyr

Zephyr™ Project

Developer Summit

June 8-10, 2021 ▪ @ZephyrIoT