Zephyr® Project
Developer Summit 2022

Building the traceability for design and testing in Zephyr

Enjia Mai GitHub: @enjiamai

Intel Corporation : Software Developer

Introduction

- What is traceability
- Why we need traceability
- How we propose to do that
- Conclusion

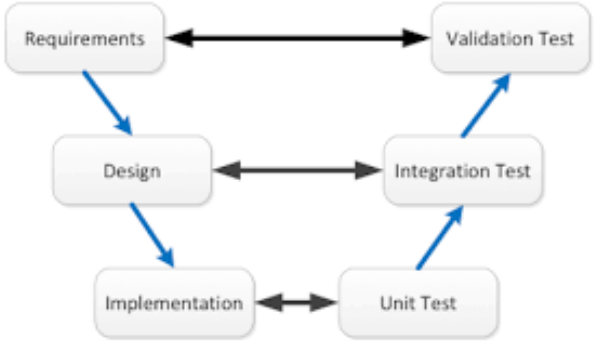
Enjia Mai
enjiamai

Intel contributor to the Zephyr Project
Member of Testing WG
Zephyr OS developer

What is traceability

- Requirement ⇔ Design ⇔ Tests
  - Requirement : RTOS features
  - Design: Zephyr Docs
  - Tests: testcases in Zephyr code

- Forward / backward traceability



V model

Why we need traceability

- User perspective
  - What features we can use in Zephyr RTOS?
  - Which platforms/boards support this feature?
  - How are their status/software quality?
    - Test Coverage and Test Results
- Developer perspective
  - How to use it: Build documentations, example samples or test cases.
  - Does it work well? Test results/report.

Open-Source Best Practices

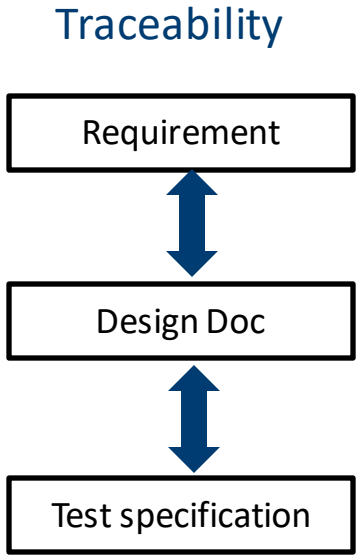
USE OPEN-SOURCE TOOLS FOR MANAGING EVERY ASPECT OF THE PROJECT

KEEP THINGS CLOSE TOGETHER

INTEGRATE WITH EXISTING TOOLS

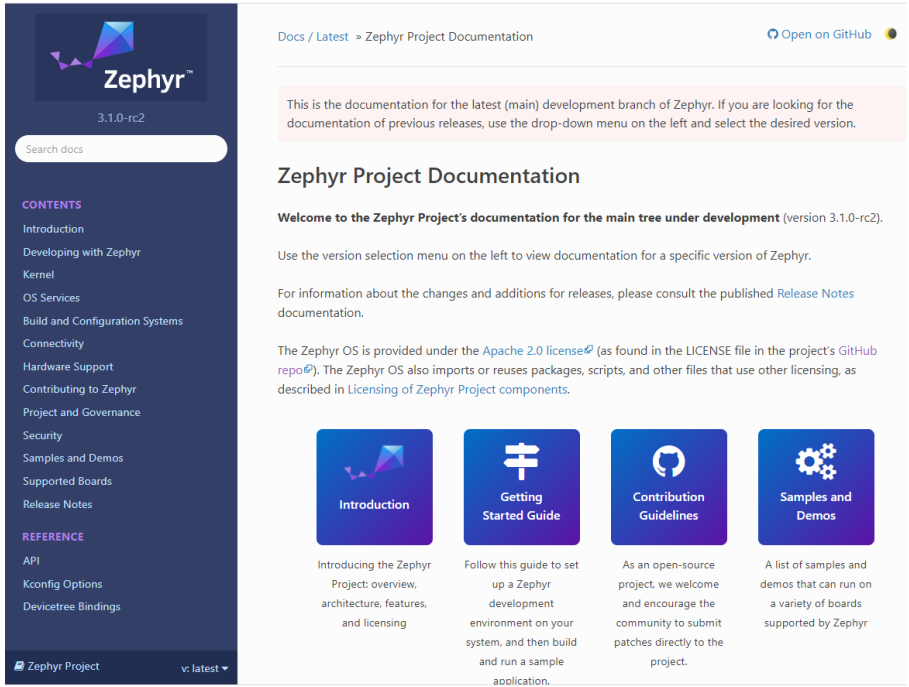
How we propose to do that

- Integrate into current documentations system
- The method of building traceability
  - Build the requirement docs
  - Refer to the existing design docs
  - Build the test specifications
  - Build the connections



Integrate into Current Documentations System

- Zephyr documentations system is composed of :
  - Doxygen
  - Sphinx
- Put a section into Zephyr doc navigation pane, it is optional and includes subsections:
  - Software Requirements
  - Test specification
  - Traceability Matrix

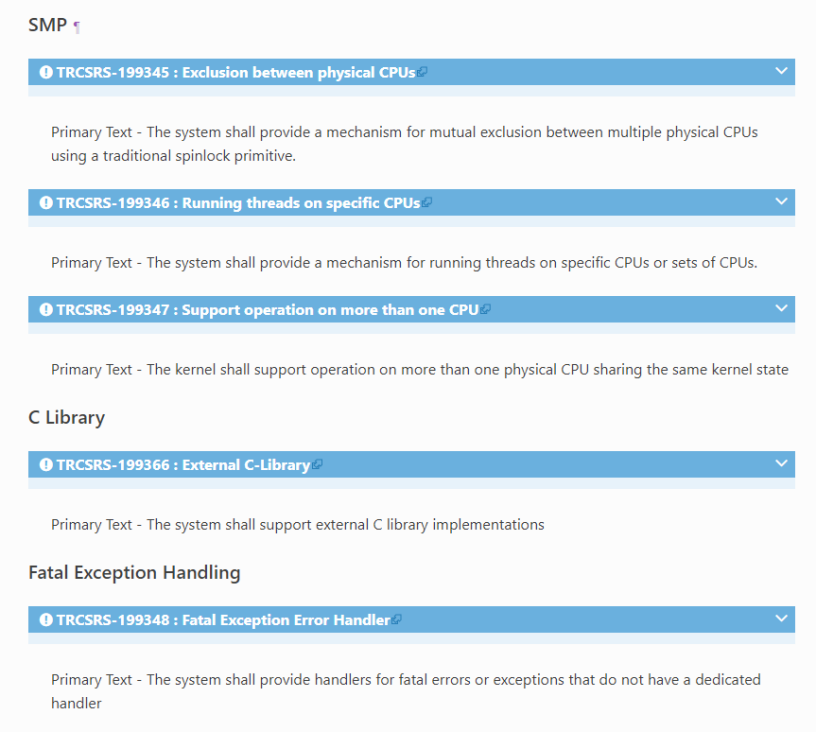


Build the Requirement Docs

- The Requirement docs need to be built.
- The requirements are usually the features of Zephyr RTOS.
- Ex. The spinlock of SMP support, the requirement might be:

"The system shall provide a mechanism for mutual exclusion between multiple physical CPUs using a traditional spinlock primitive."

Requirement Docs



Build the Test specification

- Test specification generate from the existing Doxygen descriptions of the testcase code.

- Need to add more descriptions of testcase code, Ex.

The doxygen tags
@brief
@details
- Precondition
- Expected output
- Pass and fail criteria
- Test case ID

Example:
Requirement ⇔ Design ⇔ Tests

Requirements

SMP

TRCSRS-199345 : Exclusion between physical CPUs

Fulfilled by

DESIGN_ARCH_spinlock_identical_interface : spinlocks works in uniprocessor and multiprocessor

DESIGN_ARCH_spinlock_irqlock : Spinlocks are non-recursive

DESIGN_ARCH_spinlock_support : basic function of spinlock

Validated by

TEST_VAL_test_spinlock_basic

TEST_VAL_test_spinlock_bounce_once

TEST_VAL_test_spinlock_mutual_exclusion

Primary Text - The system shall provide a mechanism for mutual exclusion between multiple physical CPUs using a traditional spinlock primitive.

Design Specification

resource. But that means that spinlocks must not be used recursively. Code that holds a specific lock must not try to re-acquire it or it will deadlock (it is perfectly legal to nest **distinct** spinlocks, however). A validation layer is available to detect and report bugs like this.

DESIGN_ARCH_spinlock_identical_interface : spinlocks works in uniprocessor and multiprocessor

Fulfills

TRCSRS-199345 : Exclusion between physical CPUs

When used on a uniprocessor system, the data component of the spinlock (the atomic lock variable) is unnecessary and elided. Except for the recursive semantics above, spinlocks in single-CPU contexts produce identical code to legacy IRQ locks. In fact the entirety of the Zephyr core kernel has now been ported to use spinlocks exclusively.

Legacy irq_lock() emulation

DESIGN_MODULE_IRQ_LOCK_APIS

DESIGN_ARCH_smp_global_lock : Provide compatible, global, recursive IRQ lock abstraction

Test Specification

TEST_VAL_test_spinlock_basic

```
void test_spinlock_basic(void)
```

Brief:

Test basic spinlock

Details:

Invoke k_spin_lock() and k_spin_unlock() API to lock/unlock and check if the locked status work correctly.

Expected output:

The locked status before locking shall be false. The locked status durning locking shall be true. The locked status after unlocking shall be false again.

Pass and fail criteria:

Pass: All the checkpoints are all passed. The locked status before/during/after locking operations work as expected.

Fail: Any one of the checkpoint is failed.

Testcase ID:

270205

Traceability Matrix and Test Coverage

Traceability Matrix

Software Requirements <-> Architecture Design Specification

Requirements to Architecture Specification traceability

Statistics: 61 out of 62 covered: 98%

Software Requirements	Architecture Specification
TRCSRS-199262	DESIGN_ARCH_heap_definition
TRCSRS-199263	DESIGN_ARCH_slab_definition
TRCSRS-199322	DESIGN_ARCH_disabling_individual_interrupts DESIGN_ARCH_mask_unmask_interrupt
TRCSRS-199323	DESIGN_ARCH_IDT_vector_table DESIGN_ARCH_configurable_direct_interrupts DESIGN_ARCH_configuring_interrupts DESIGN_ARCH_configuring_interrupts_buildtime DESIGN_ARCH_configuring_interrupts_runtime DESIGN_ARCH_raising_exception_nohandler DESIGN_ARCH_synchronously_run_after_interrupt
TRCSRS-199324	DESIGN_ARCH_dedicated_interrupt_stack DESIGN_ARCH_interrupt_nesting
TRCSRS-199325	DESIGN_ARCH_32bit_hardware_clock DESIGN_ARCH_64bit_uptime_counter DESIGN_ARCH_convert_time_unit DESIGN_ARCH_specify_milliseconds DESIGN_ARCH_tickless

Software Requirements <-> Validation Test Specification

Requirements to Test Specification traceability

Statistics: 61 out of 62 covered: 98%

Software Requirements	Validation Test Specification
TRCSRS-199262	TEST_VAL_test_k_heap_alloc TEST_VAL_test_mheap_malloc_align4 TEST_VAL_test_mheap_malloc_free
TRCSRS-199263	TEST_VAL_test_mslab_alloc_align TEST_VAL_test_mslab_kdefine TEST_VAL_test_mslab_kinit
TRCSRS-199323	TEST_VAL_test_build_time_direct_interrupt TEST_VAL_test_build_time_interrupt TEST_VAL_test_direct_interrupt TEST_VAL_test_isr_dynamic TEST_VAL_test_isr_regular TEST_VAL_test_prevent_interruption TEST_VAL_test_run_time_interrupt
TRCSRS-199324	TEST_VAL_test_nested_isr
TRCSRS-199325	TEST_VAL_test_clock_cycle TEST_VAL_test_clock_uptime TEST_VAL_test_ms_time_duration TEST_VAL_test_tickless_slice TEST_VAL_test_tickless_sysclock TEST_VAL_test_time_conversions

Build the connections

- Achieve by adding extra python modules:
  - Breathe
  - mlx.traceabiliy
- By adding “item” label to build the connections between Requirement docs, Design specification and Test specification.
- In Test Specification, the content of testcase details should refer to the Doxygen description in source code.

Requirements

```
SMP
---

.. item:: TRCSRS-199345 Exclusion between physical CPUs

   Primary Text - The system shall provide a mechanism for mutual exclusion between
   multiple physical CPUs using a traditional spinlock primitive.

.. item:: TRCSRS-199346 Running threads on specific CPUs

   Primary Text - The system shall provide a mechanism for running threads on specific
   CPUs or sets of CPUs.

.. item:: TRCSRS-199347 Support operation on more than one CPU

   Primary Text - The kernel shall support operation on more than one physical CPU
   sharing the same kernel state
```

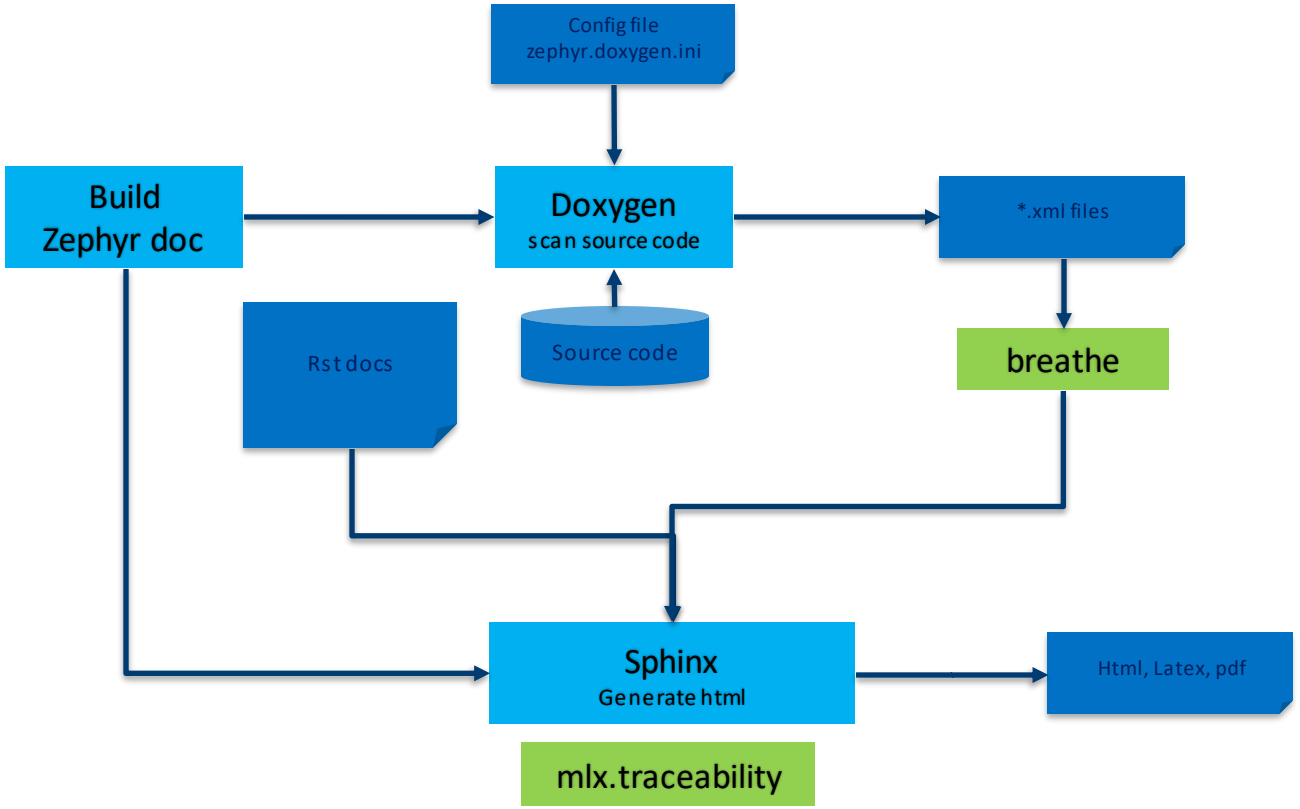
Design specification (Zephyr docs)

```
Spinlocks
=========

.. item:: DESIGN_MODULE_SPINLOCKS_APIS

.. item:: DESIGN_ARCH_spinlock_support basic function of spinlock
      :fulfills: TRCSRS-199345

SMP systems provide a more constrained :c:func:`k_spin_lock` primitive
that not only masks interrupts locally, as done by :c:func:`irq_lock`, but
also atomically validates that a shared lock variable has been
modified before returning to the caller, "spinning" on the check if
needed to wait for the other CPU to exit the lock.  The default Zephyr
implementation of :c:func:`k_spin_lock` and :c:func:`k_spin_unlock` is built
on top of the pre-existing :c:struct:`atomic_` layer (itself usually
implemented using compiler intrinsics), though facilities exist for
architectures to define their own for performance reasons.
```

Test specification

```
SMP
***

.. item:: TEST_VAL_test_spinlock_mutual_exclusion
     :validates: TRCSRS-199345
     :result: RESULT_test_spinlock_mutual_exclusion

.. code-block:: c

       void test_spinlock_mutual_exclusion(void)

**Brief:**
       Test basic mutual exclusion using interrupt masking

**Details:**
       Validate the spinlocks can be initialized at run-time.
       Validate the spinlocks in uniprocessor context should achieve mutual exclusion using
       interrupt masking.
```

Traceability generate flow



Test Result

- Use twister to run test automation and get the XML testing report.
  - The pass/fail/skip tests, pass rate, execution time, log, etc.
- Transform the XML test report to a Reports matrix:
  - Better to show the test result between testcases and the boards.

Reports Matrix

Failed	461
Passed	17152
Skipped	1684

portability.cmsis_rtos_v2	acrn	ehl_crb	frdm_k64f	intel_adsp_cavs15	it8xxx2_evb	mec15xxevb_assy6853	mec172xevb_assy6906	nsim_em	tiger_lake	up_squared	whiskey_lake	
- .delay	ok	ok	ok	ok	s	ok	ok	ok	ok	ok	Partial Pass
- .event_flags_isr	ok	ok	ok	ok	s	ok	ok	ok	ok	ok	Partial Pass
- .event_flags_no_wait_timeout	ok	ok	ok	ok	s	ok	ok	ok	ok	ok	Partial Pass
- .event_flags_signalled	ok	ok	ok	ok	s	ok	ok	ok	ok	ok	Partial Pass
- .kernel_apis	ok	ok	ok	ok	s	ok	ok	ok	ok	ok	Partial Pass
- .mempool	ok	ok	ok	ok	s	ok	ok	ok	ok	ok	Partial Pass
- .mempool_dynamic	ok	ok	ok	ok	s	ok	ok	ok	ok	ok	Partial Pass
- .messageq	ok	ok	ok	ok	s	ok	ok	ok	ok	ok	Partial Pass
- .mutex	ok	ok	ok	ok	s	ok	ok	ok	ok	ok	Partial Pass
- .mutex_lock_timeout	ok	ok	ok	ok	s	ok	ok	ok	ok	ok	Partial Pass
- .semaphore	ok	ok	ok	ok	s	ok	ok	ok	ok	ok	Partial Pass
- .thread_apis	ok	ok	ok	ok	s	ok	ok	ok	ok	ok	Partial Pass
- .thread_apis_dynamic	ok	ok	ok	ok	s	ok	ok	ok	ok	ok	Partial Pass
- .thread_detached	ok	ok	ok	ok	s	ok	ok	ok	ok	ok	Partial Pass
- .thread_flags_isr	ok	ok	ok	ok	s	ok	ok	ok	ok	ok	Partial Pass
- .thread_flags_no_wait_timeout	ok	ok	ok	ok	s	ok	ok	ok	ok	ok	Partial Pass
- .thread_flags_signalled	ok	ok	ok	ok	s	ok	ok	ok	ok	ok	Partial Pass
- .thread_join	ok	ok	ok	ok	s	ok	ok	ok	ok	ok	Partial Pass
- .thread_joinable_detach	ok	ok	ok	ok	s	ok	ok	ok	ok	ok	Partial Pass
- .thread_joinable_terminate	ok	ok	ok	ok	s	ok	ok	ok	ok	ok	Partial Pass
- .thread_prio	ok	ok	ok	ok	s	ok	ok	ok	ok	ok	Partial Pass
- .thread_prio_dynamic	ok	ok	ok	ok	s	ok	ok	ok	ok	ok	Partial Pass
- .timer	ok	ok	ok	ok	s	ok	ok	ok	ok	ok	Partial Pass

Conclusion

- Have traceability provides us:
  - The link between requirements, design and tests.
  - The traceability matrix and test coverage.
  - Spend less effort to build a higher quality SW system.

- What's the next?
  - Solve some opens of test specification generating.

Q & A

Thank you for listening!