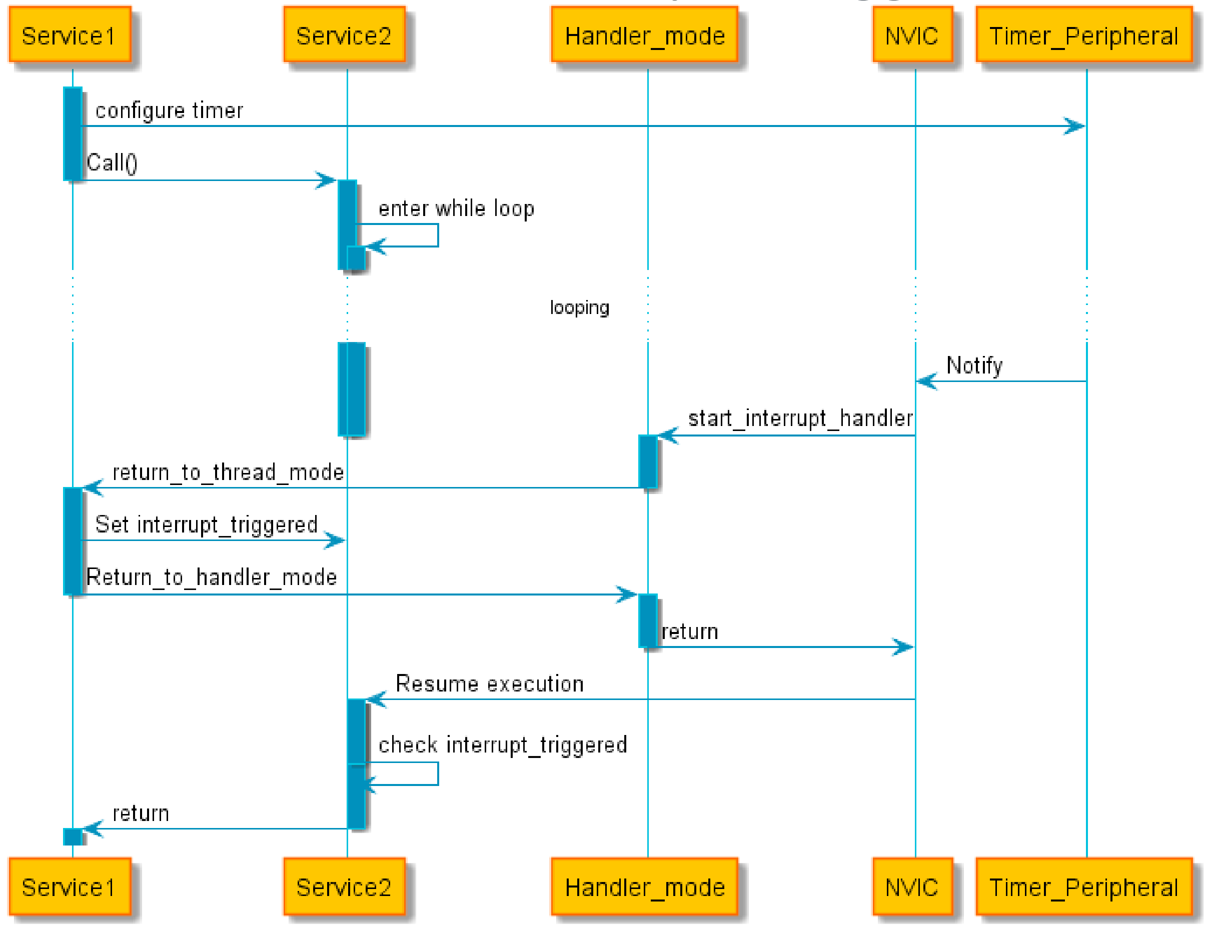
Automated testing of interrupt handling in TF-M

Proposal for a tool

Mate Toth-Pal
2020.03.19

Current way of testing interrupt handling

- Generate interrupts using timers
- Use while loops to make sure that the interrupt is triggered when the execution is at the desired location



Limitations of this approach

- On the currently supported platforms there are 2 timers
- One is configured as Non-Secure, one is configured as Secure. (Cannot test Secure interrupts interrupting a Secure interrupt handler)
- Each location that is to be interrupted, a while loop is needed.
- Cannot really solve negative tests (trigger a lower priority interrupt and verify that the handler is not executed immediately)
- Interrupt handlers set a global variable to signal that the interrupt happened. The location of this variable need to be passed to all the interrupted locations, and need to be made readable.

Limitations of this approach (continued)

- An issue was found by a partner that we couldn't catch with the current tests. It was found by code review
- https://lists.trustedfirmware.org/pipermail/tf-m/2019-December/000593.html
- To reproduce this issue, a secure interrupt handler must be interrupted by another, higher priority secure interrupt.

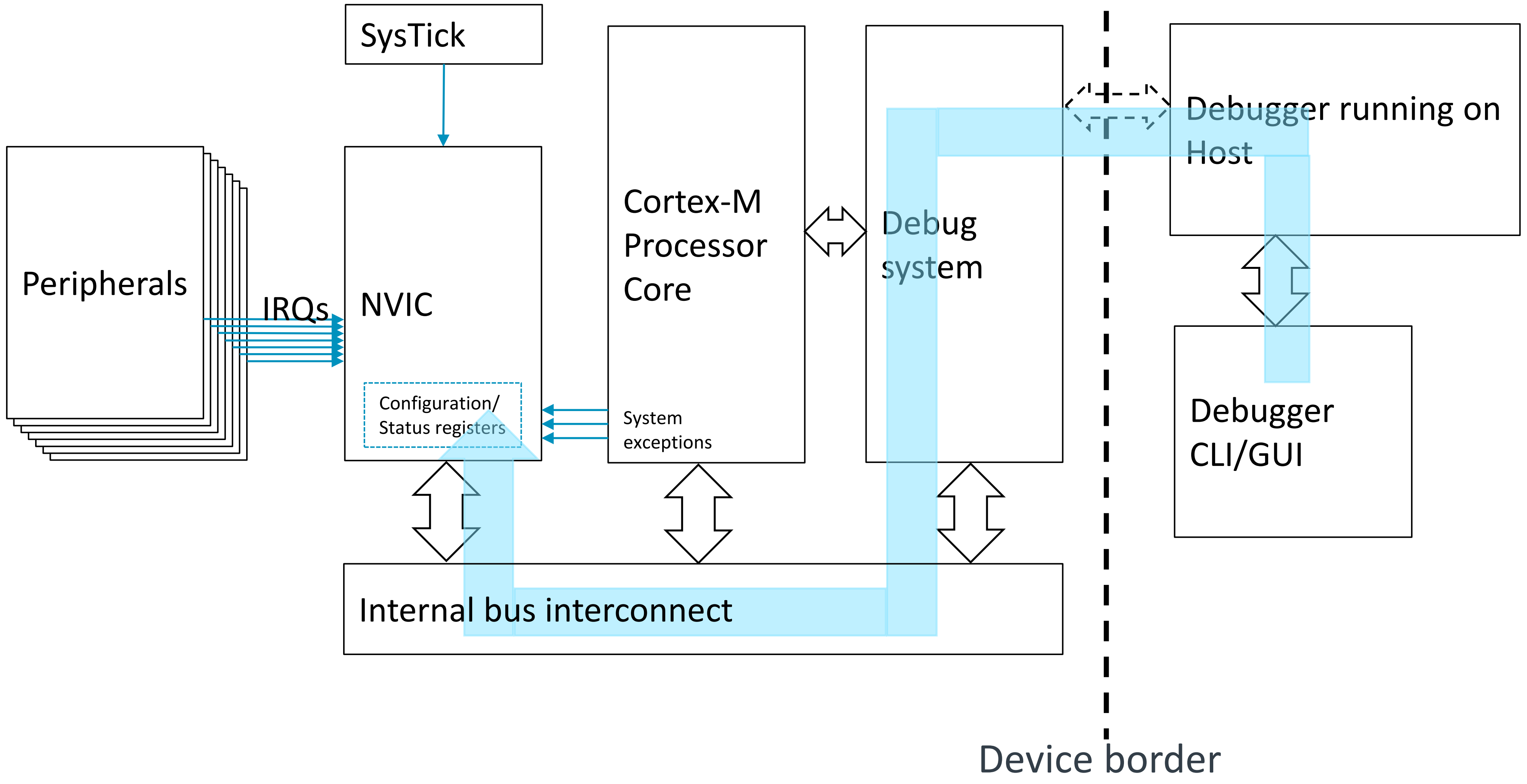
Alternative approaches

- Trigger the interrupt in place (using the **STIR** or **NVIC_ISPRn** registers)
- Limitations: Can only be used in privileged mode, and not possible to trigger Secure interrupt from Non-Secure code.

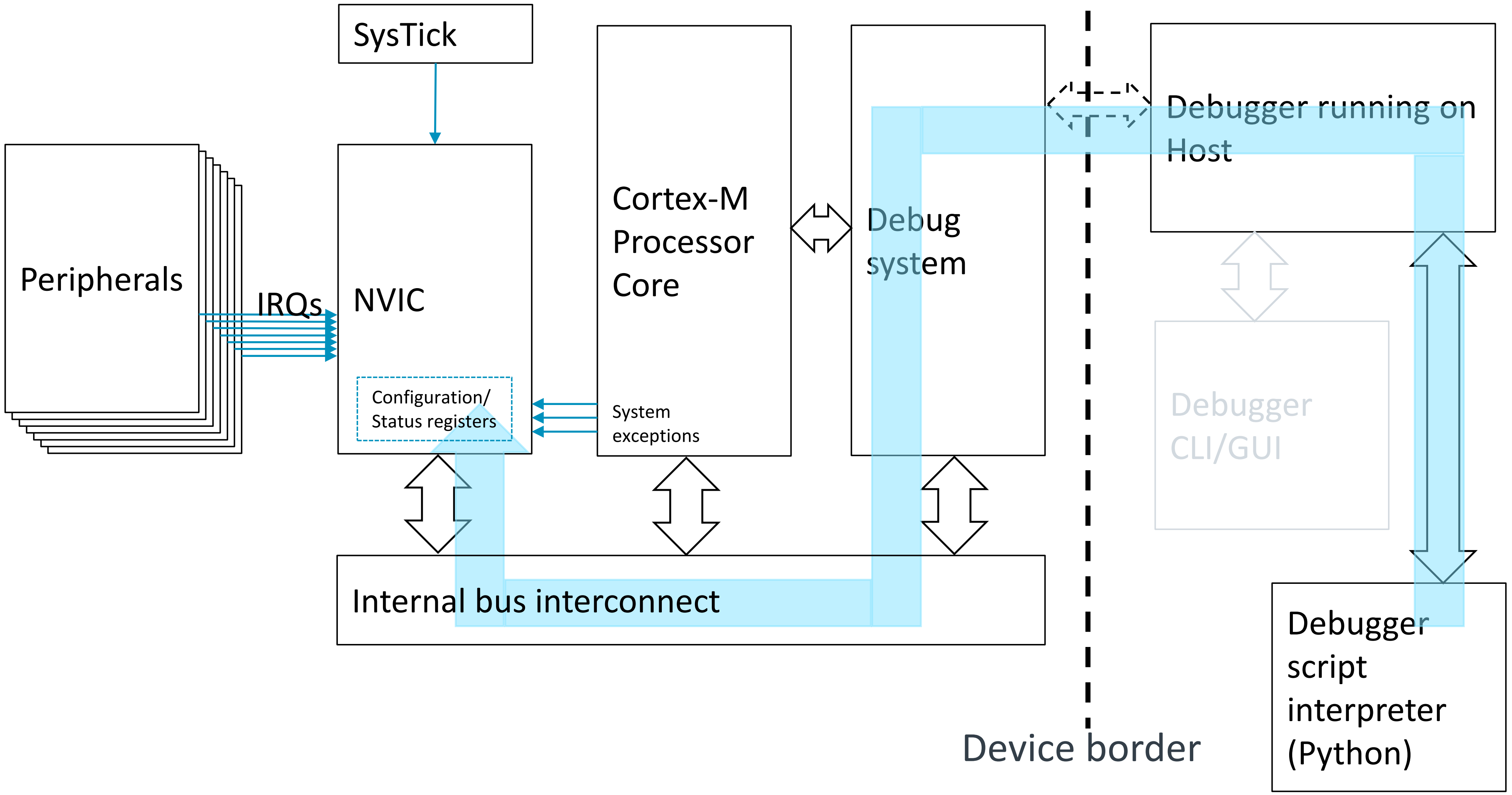
- Trigger interrupt from the debugger (using the **STIR** or **NVIC_ISPRn** registers)
- Limitations: $\boxed{\text{labor intensive, slow, and cannot be used in automated CI.}}$

Is this necessarily true?

“Normal” way of debugging

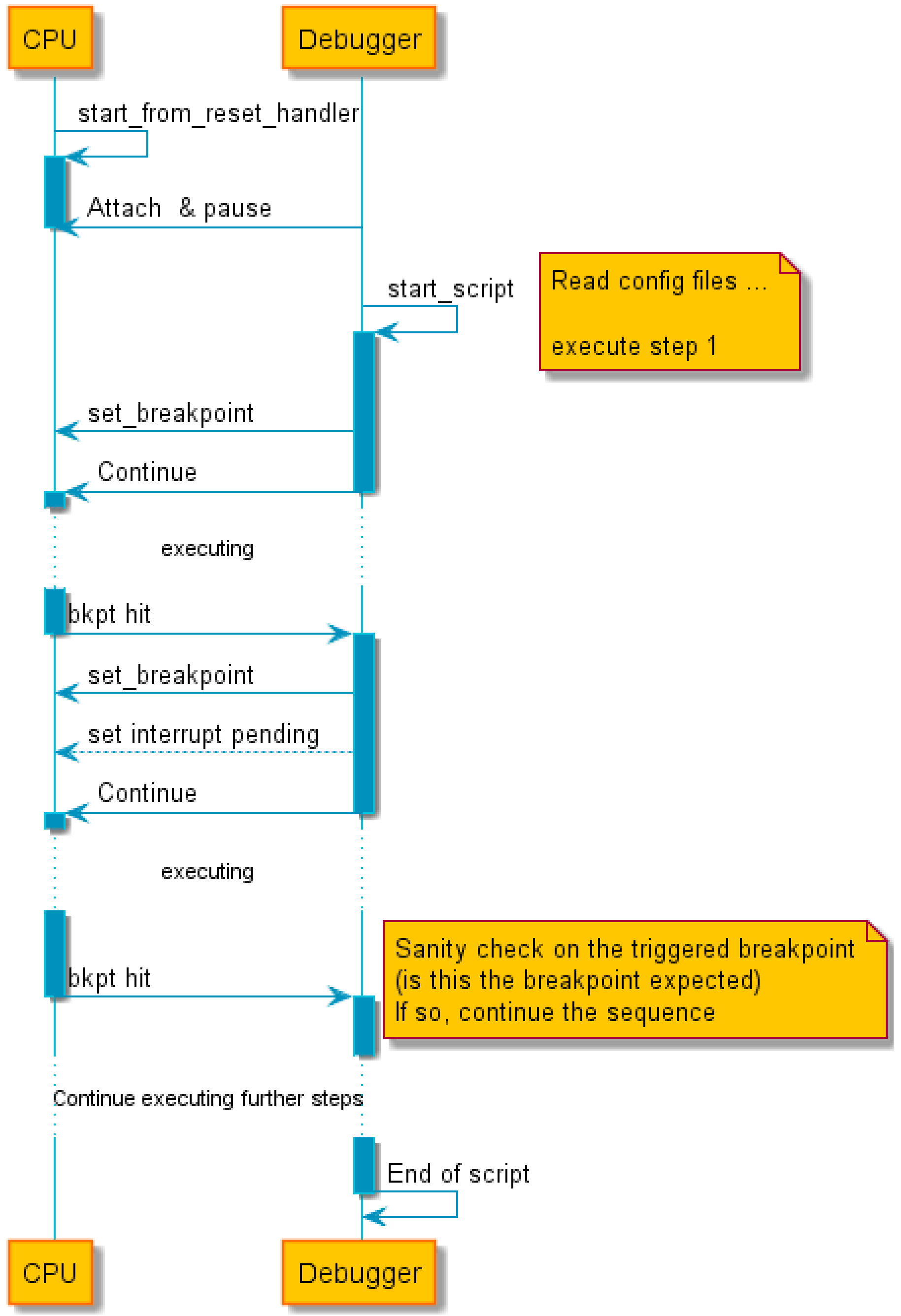


“Automated” way of debugging

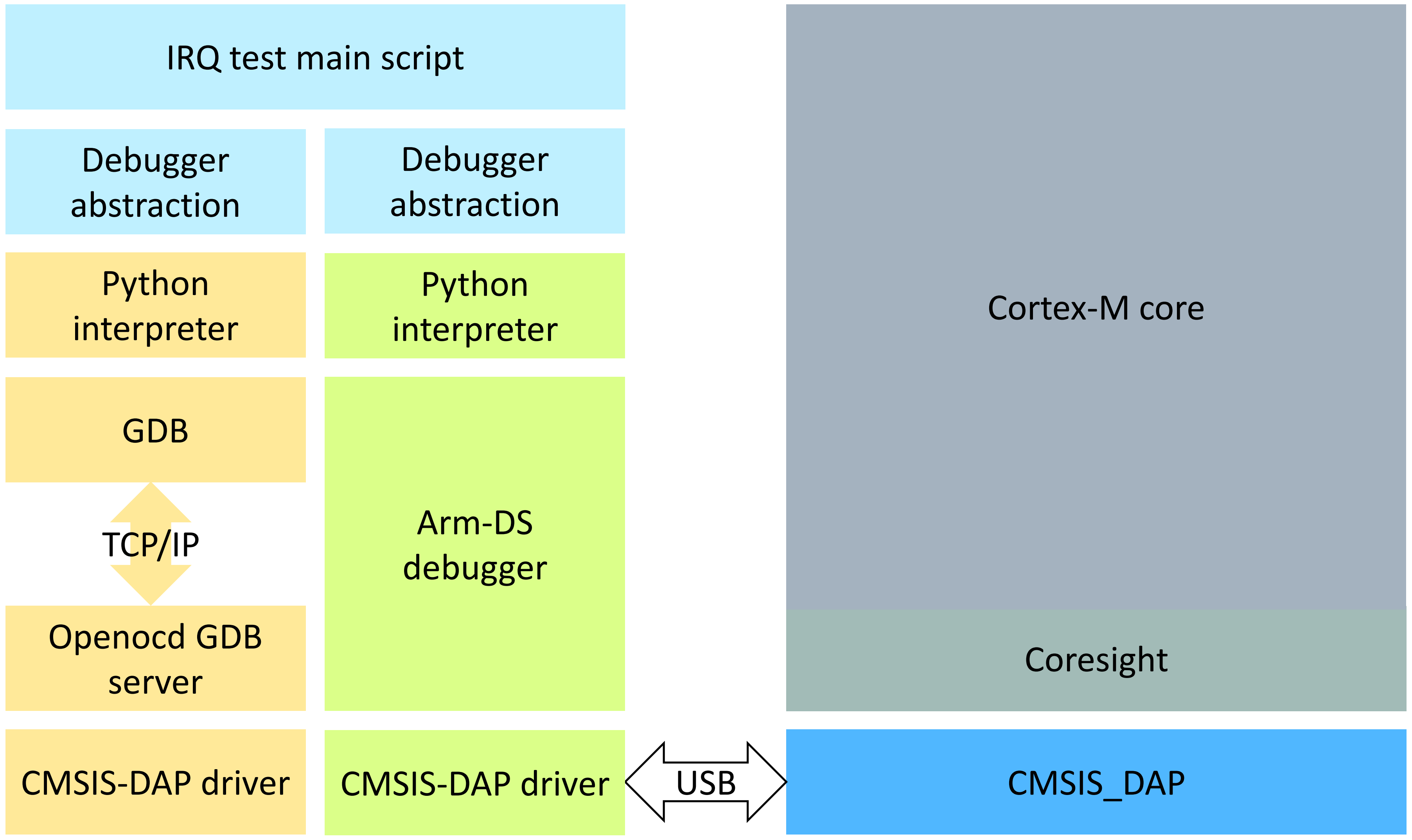


The sequence executed by the script

- Steps are executed by the debugger one after another
- For each step:
  - If not the first step, the breakpoint that was hit is sanity checked (more on that later)
  - Previously set breakpoints are deleted
  - One or more (more on that later) breakpoint is set
  - Interrupt is set pending, if required by the test step
  - Target execution continues



The stack


IRQ test main script
Debugger abstraction
Debugger abstraction
Python interpreter
Python interpreter
GDB
TCP/IP
Openocd GDB server
Arm-DS debugger
CMSIS-DAP driver
CMSIS-DAP driver
USB
Cortex-M core
Coresight
CMSIS_DAP

The stack (Continued)

- A debugger abstraction class (see next slide) is created, so that the main logic of the script, and the testcase parsing can be implemented only once, and to be used by multiple debuggers
- Debug abstraction is necessary because the python API to control the debugger is not standard
- To add a new debugger support, only the debugger abstraction class needs to be implemented
- The prototype can support GDB and ARM-DS debuggers

API for debugger abstraction

```
def set_breakpoint(self, name, location):
    pass
def trigger_interrupt(self, interrupt_line):
    pass
def continue_execution(self):
    pass
def clear_breakpoints(self):
    pass
def get_triggered_breakpoint(self):
    pass
```

Script input files

```
{
  "breakpoints": {
    "irq_test_iteration_start": {
      "file": "core_ns_positive_testsuite.c",
      "line": 656
    },
    "irq_test_service1_high_handler": {
      "symbol": "IRQ_TEST_1_HIGH_isr"
    }
  }
}
```

```
{
  "irqs": {
    "test_service1_low":
    { "line_num" : 51 },
    "test_service1_medium":
    { "line_num" : 52 }
  }
}
```

```
{
  "description" : [“Some test"],
  "steps": [
    {
      "wait_for" : "irq_test_service2_prepare"
    },
    {
      "expect" : "irq_test_service1_low_handler",
      "trigger" : "test_service1_low"
    },
    {
      "wait_for" : "irq_test_iteration_start"
    }
  ]
}
```

Script input files (Continued)

- All the input files are in ‘json’ format
- Breakpoints file
  - Assign symbolic name to code locations
  - Contains all the code locations where the testcase should set a breakpoint to
- IRQs file
  - Assign symbolic name to IRQ numbers
- Testcase file
  - The actual description of a testcase.
  - Keywords:
    - ‘wait_for’: Set a breakpoint at that location
    - ‘expect’: Set a breakpoint at that location **and** the location specified by the next ‘wait_for’. In the next step when execution stopped at the expected location, or not (for example to check whether the interrupt handler is executed actually)
    - ‘trigger’: Set the specified interrupt pending

Future improvement

- Randomized execution
- Improve config file formats
- Add further functionality (e.g. setting variables)
- Improve usability (Load symbols from script, attach to target automatically)
- Integration with CI

Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Köszönöm
Kiitos
감사합니다
धन्यवाद
شكرًا
ধন্যবাদ
תודה