Trusted Firmware Explorer (TFX)

A short introduction

Mate Toth-Pal

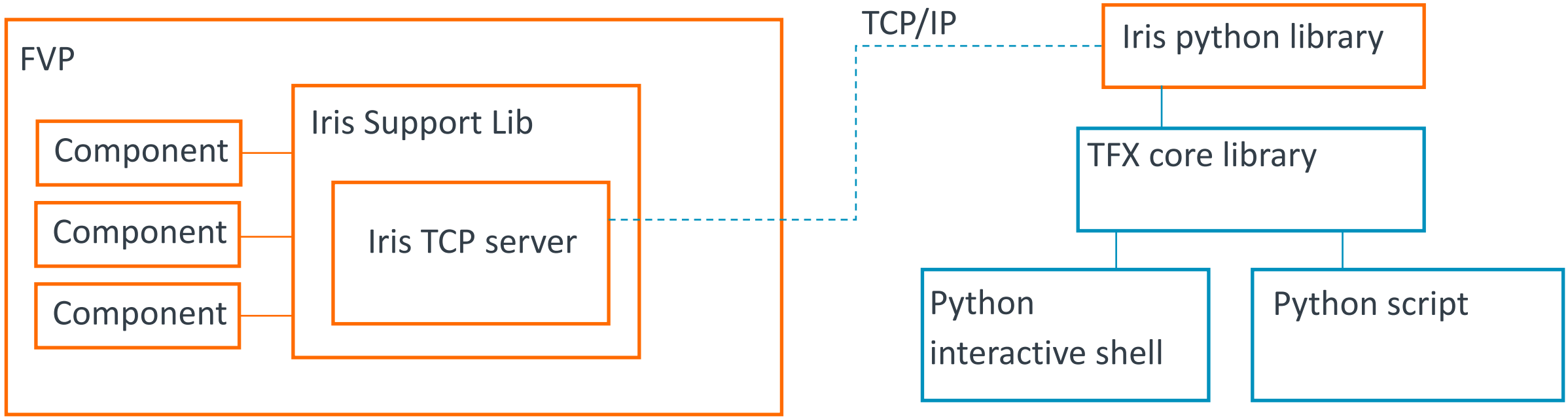
2023.02.16

Overview

- Trusted Firmware Explorer (TFX) aims to be a Trusted Firmware aware debugger enabling the following use cases:
  - Interactive exploration of SMC interfaces and Trusted Firmware SW state
  - Explore and prototype example flows for SMC interfaces by interactive injection of SMC calls into the debug target from the command line.
  - Inspect the system state.
  - Development of concurrency tests
  - Script concurrent execution flows which trigger interesting interleavings which might be difficult to test otherwise.

Overview

- TFX is a Python library which communicates with the system under test running in an FVP via the Iris Debug Interface.
- The TFX makes use of the Iris python library provided for FVPs which provides basic abstractions for debug targets (e.g. CPU), breakpoints, memory operations (read/write) in memory spaces (Secure Monitor, Physical Memory, NS Hyp)



Basic Features

- Inject and execute code in FVP
- Trigger Normal world interrupt (writing ICC_SGI1R_EL1)
- Issue RMI (Realm Management interface) calls
  - Signature is parsed from RMM machine readable spec
  - Structures types are parsed from RMM ELF.
- Call C functions on the debug target
- Ability to start an interactive “TFX shell” via which the state of the target can be interrogated and modified.

Advanced Features

- Execute multiple SMC calls in a single run
  - Compile asm code on the fly and inject it to the FVP memory
- Create Realms from ELF files
  - Execute complex set of RMI calls to delegate memory granules, set up Stage 2 translation tables, create RECs (Realm Execution Context)
- Memory overlay class for accessing memory using C structure names

Given the following C definition:

```
struct realm_params {
    /* ... */
    unsigned int ipa_width;
    /* ... */
};
```

This can be accessed in Python via:

```
params = create_overlay("realm_params", addr)
print(params.ipa_width)
```

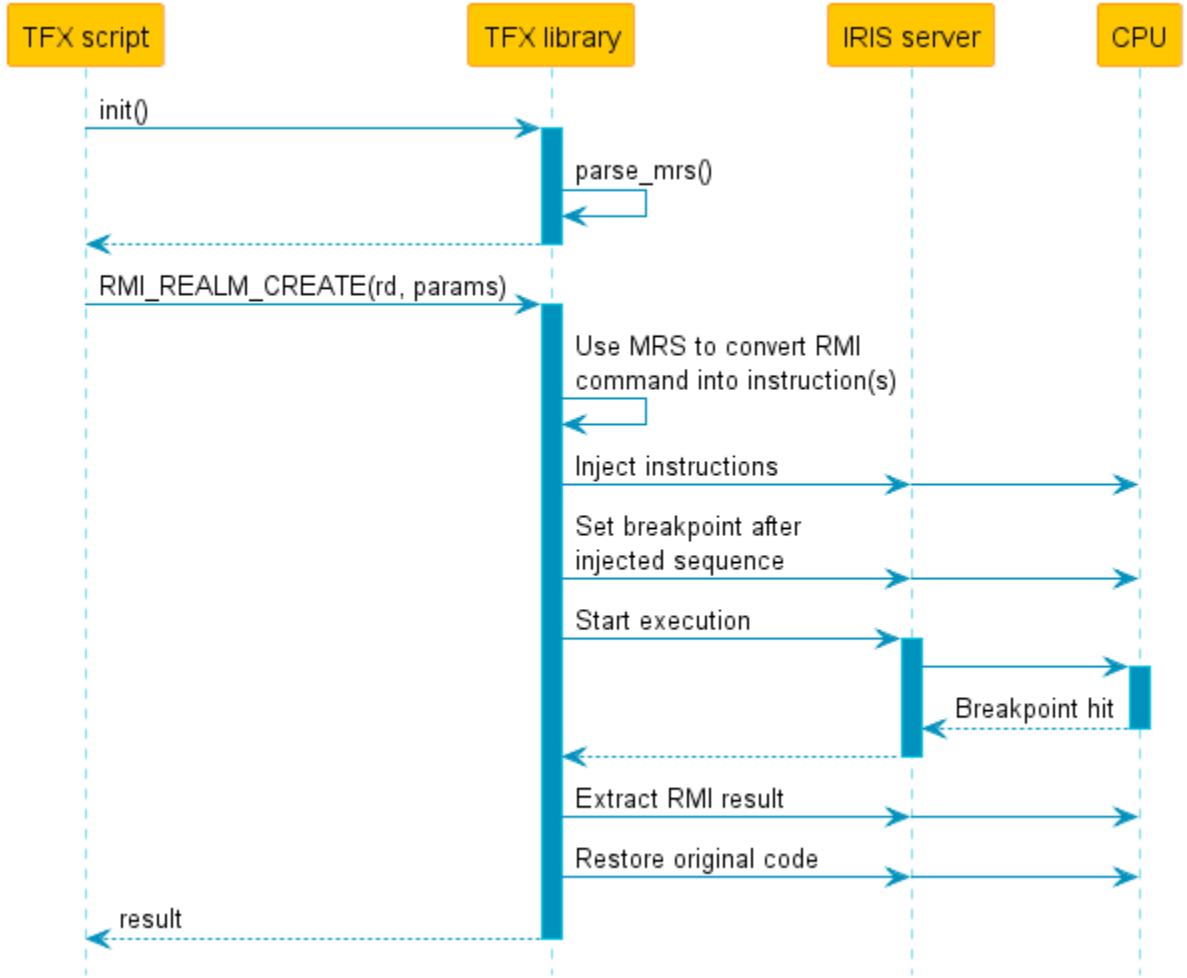
In this example, “addr” is a physical address. Similar Python code can be used to access Realm memory, in which case TFX arranges appropriate stage 2 translation.

Flows

- Flows in TFX are python generator functions
  - They describe a sequence of actions to be executed (RMI calls, Function calls)
  - Might contain sanity checks on RMI call results, and on other state of the firmware.
- There are two ways to run a flow
  - `Flow.run_f(flow)`
    - The flow is expected to be a generator that yields every time it could/should be interleaved.
    - With this type of flows, the flow should start/stop the debug target and execute the RMI/function calls
  - `Flow.run_c(flow)`
    - The flow is expected to be a generator that yields `CodeInject` objects
    - These generators are expected not to start/stop the debug target
    - The generator is fully exhausted by the function, the `CodeInject`s are merged, and all the generated code is executed at once

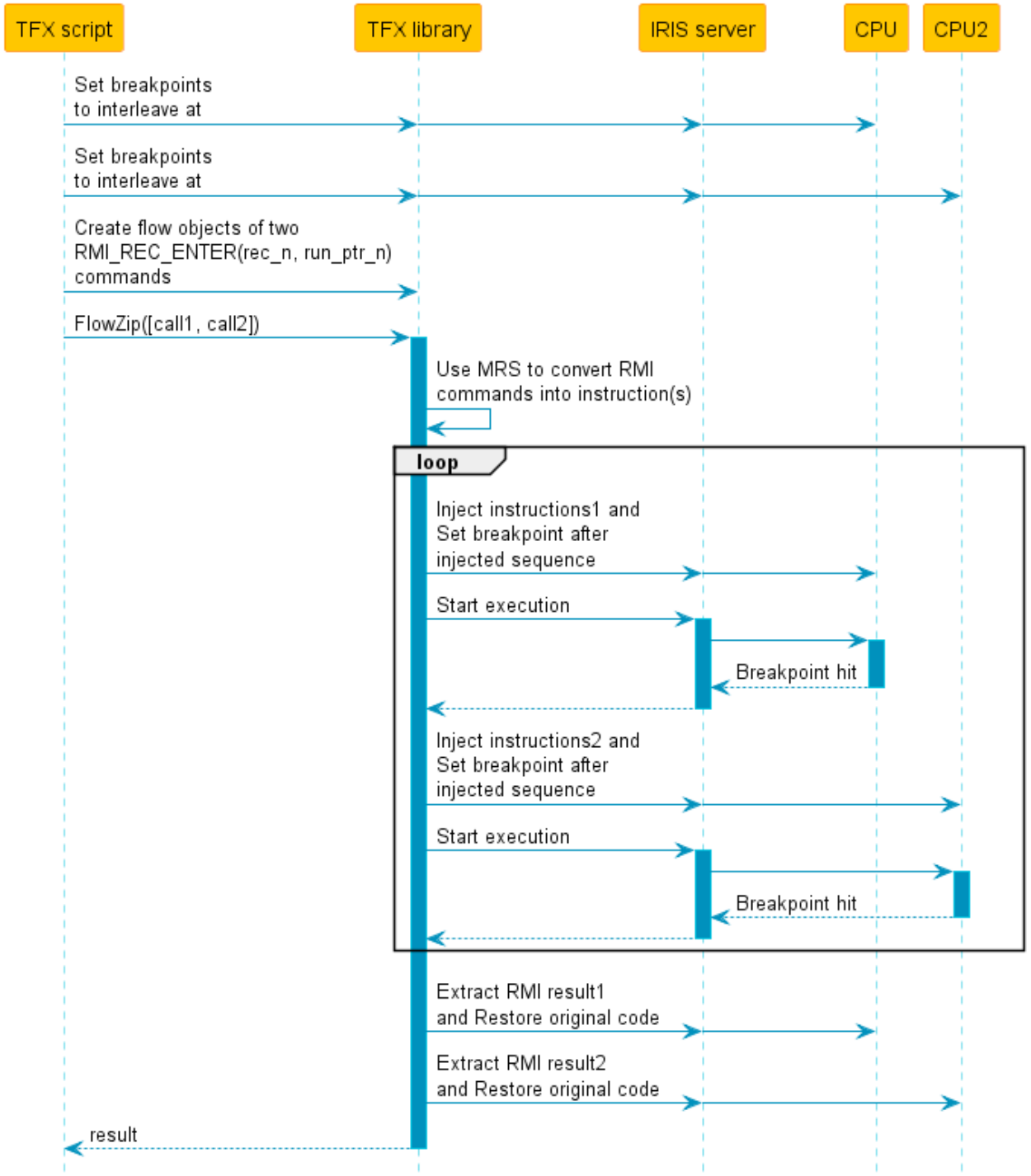
Executing an RMI call

- MRS is parsed (Only once before the first call)
- Machine code is assembled using info from MRS
- The assembled code is injected at the current PC
- Breakpoint is set after the injected code
- The debug target is started
- When the breakpoint at the end of the injected code is hit, extract RMI results
- Restore original code in memory and register content (including PC)



Executing overlapping RMI calls

- Set breakpoints for both calls
- Create flow objects, and call FlowZip to start the interleaving of the flows
- Then in a loop (until both flows are finished):
  - Inject instructions for the flow1
  - Start the debug target assigned to flow1
  - When a breakpoint is hit, the execution returns to the TFX library
  - Inject instructions for the flow2
  - Start the debug target assigned to flow2
  - When a breakpoint is hit, the execution returns to the TFX library
- Extract results for both flows and restore memory and CPU states



Test cases

- Realm creation/destroy
- PSCI Controls
  - Realm calls PSCI SMC calls, the test checks that the return values to the NS host are the expected.
- Data Abort
  - Try to access unmapped addresses and outside of PAR
- Attestation
  - Parallel execution of attestation requests from different RECs
  - Overlapping attestation request and measurement update