Trace-based Code Coverage Tooling for Firmware projects

@TF-A Technical Forum

Basil Eljuse & Saul Romero

Nov 2020

Agenda

- Introduction
- Rationale
- Technical Overview
- Tooling Access and Usage
- Future Direction
- Q&A

About Us

SW Quality organization

within Arm's

Open Source Software Group

Basil Eljuse - Principal SW Engr – Tech Lead

Saul Romero - Staff SW Engr – Tooling Specialist

Focus is on

Quality improvement initiatives

Common hard tooling problems

Automation improvements

Mostly internal faced

Public contributions

big.LITTLE sched-tests (precursor to LISA tool)

scmi-tests (part of ACS)

qa-tools (most recent contribution)

arm

Rationale

Why we went down this path?

Motivation

- Emphasis on 'demonstrable quality' more than ever
- Lack of measures => 'flying blind'
- Code coverage is one useful measure
- Code coverage – feedback with potential for actionable outcomes
  - indicator of test coverage
    - Is my test-set good enough?
    - Can I direct my test effort better?
  - residual risk to quality
    - What am I not covering with my current tests?

Problem Statement

- Firmware projects - Traditional coverage tooling with code instrumentation not an option
  - Memory constraint platforms
    - code size limitations
  - Higher degree of platform code dependency
    - emulation expensive and less desirable
  - No COTS tooling available
- Need: Perform code coverage measurement without doing code instrumentation.

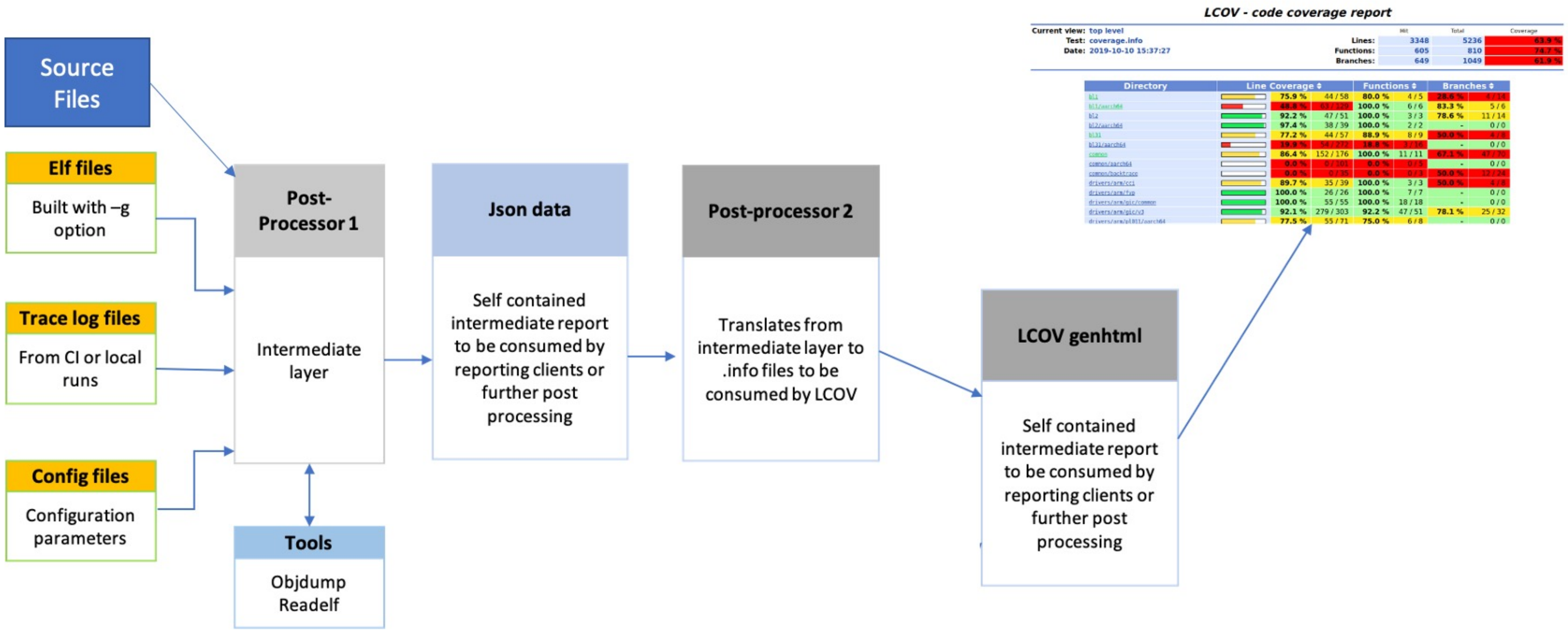
arm

Trace-based Coverage Tooling Design

https://gitlab.arm.com/tooling/qa-tools/-/blob/master/coverage-tool/docs/design_overview.md

- Capture Phase
  - Fastmodel - MTI based custom plugin captures trace with instructions executed
- Analysis Phase
  - Dwarf signature (-g compiler flag) – C source mapping
  - Object dump data – Instruction level mapping
- Visualisation Phase
  - Lcov reports

- Overview



Current Tooling Capability

What is supported today?

- Statement coverage
- Function coverage
- Branch Coverage
- Merging of related coverage reports
- Baseline viewing of coverage info

- Lcov Report View

LCOV - code coverage report

ew: top level
st: coverage.info
te: 2019-10-10 15:37:27

	Hit	Total	Coverage
Lines:	3348	5236	63.9 %
Functions:	605	810	74.7 %
Branches:	649	1049	61.9 %

Directory	Line Coverage		Functions		Branches	
bl1	75.9 %	44 / 58	80.0 %	4 / 5	28.6 %	4 / 14
bl1/aarch64	48.8 %	63 / 129	100.0 %	6 / 6	83.3 %	5 / 6
bl2	92.2 %	47 / 51	100.0 %	3 / 3	78.6 %	11 / 14
bl2/aarch64	97.4 %	38 / 39	100.0 %	2 / 2	-	0 / 0
bl31	77.2 %	44 / 57	88.9 %	8 / 9	50.0 %	4 / 8
bl31/aarch64	19.9 %	54 / 272	18.8 %	3 / 16	-	0 / 0
common	86.4 %	152 / 176	100.0 %	11 / 11	67.1 %	47 / 70
common/aarch64	0.0 %	0 / 101	0.0 %	0 / 5	-	0 / 0
common/backtrace	0.0 %	0 / 35	0.0 %	0 / 3	50.0 %	12 / 24
drivers/arm/cci	89.7 %	35 / 39	100.0 %	3 / 3	50.0 %	4 / 8
drivers/arm/fvp	100.0 %	26 / 26	100.0 %	7 / 7	-	0 / 0
drivers/arm/gic/common	100.0 %	55 / 55	100.0 %	18 / 18	-	0 / 0
drivers/arm/gic/v3	92.1 %	279 / 303	92.2 %	47 / 51	78.1 %	25 / 32
drivers/arm/pl011/aarch64	77.5 %	55 / 71	75.0 %	6 / 8	-	0 / 0
drivers/arm/smmu	56.5 %	13 / 23	100.0 %	3 / 3	57.1 %	8 / 14
drivers/arm/sp805	66.7 %	2 / 3	66.7 %	2 / 3	-	0 / 0
drivers/arm/tzc	90.2 %	46 / 51	100.0 %	7 / 7	50.0 %	2 / 4
drivers/cfi/v2m	0.0 %	0 / 28	0.0 %	0 / 5	50.0 %	6 / 12
drivers/console	88.6 %	31 / 35	100.0 %	6 / 6	62.5 %	10 / 16
drivers/delay_timer	86.7 %	13 / 15	75.0 %	3 / 4	-	0 / 0
drivers/io	65.6 %	204 / 311	75.0 %	30 / 40	78.8 %	52 / 66
include/arch/aarch64	55.9 %	38 / 68	74.7 %	65 / 87	-	0 / 0
include/common	100.0 %	6 / 6	100.0 %	4 / 4	-	0 / 0
include/lib	100.0 %	9 / 9	87.8 %	65 / 74	-	0 / 0
include/lib/el3_runtime	100.0 %	3 / 3	100.0 %	1 / 1	-	0 / 0
include/lib/libfdt	100.0 %	1 / 1	100.0 %	1 / 1	-	0 / 0
include/lib/psci	100.0 %	3 / 3	100.0 %	4 / 4	-	0 / 0
lib/aarch64	65.6 %	86 / 131	92.3 %	12 / 13	-	0 / 0
lib/cpus	55.6 %	10 / 18	50.0 %	1 / 2	50.0 %	4 / 8
lib/cpus/aarch64	14.9 %	93 / 625	11.5 %	10 / 87	-	0 / 0
lib/el3_runtime/aarch64	94.9 %	185 / 195	92.9 %	13 / 14	83.3 %	15 / 18
lib/extensions/amu/aarch64	27.8 %	15 / 54	50.0 %	4 / 8	70.0 %	7 / 10
lib/extensions/spe	42.1 %	8 / 19	75.0 %	3 / 4	50.0 %	4 / 8
lib/extensions/sve	50.0 %	9 / 18	75.0 %	3 / 4	62.5 %	5 / 8
lib/libc	70.5 %	79 / 112	72.7 %	8 / 11	59.1 %	13 / 22
lib/libfdt	81.0 %	115 / 142	100.0 %	19 / 19	58.1 %	43 / 74
lib/locks/bakery	100.0 %	23 / 23	100.0 %	2 / 2	75.0 %	3 / 4
lib/locks/exclusive/aarch64	100.0 %	10 / 10	100.0 %	2 / 2	-	0 / 0
lib/pmf	51.4 %	37 / 72	100.0 %	6 / 6	65.4 %	17 / 26
lib/psci	76.3 %	477 / 625	83.1 %	54 / 65	60.7 %	139 / 229
lib/psci/aarch64	96.8 %	30 / 31	100.0 %	3 / 3	-	0 / 0

```
          :           1 :         assert(val & SCTLR_M_BIT);
          :           1 :         assert(val & SCTLR_C_BIT);
          :           1 :         assert(val & SCTLR_I_BIT);
          :             :         /*
          :             :          * Check that Cache Writeback Granule (CWG) in CTR_EL0 matches the
          :             :          * provided platform value
          :             :          */
          :             :         val = (read_ctr_el0() >> CTR_CWG_SHIFT) & CTR_CWG_MASK;
          :             :         /*
          :             :          * If CWG is zero, then no CWG information is available but we can
          :             :          * at least check the platform value is less than the architectural
          :             :          * maximum.
          :             :          */
 [ -  - ]:           1 :         if (val != 0)
          :           0 :                 assert(CACHE_WRITEBACK_GRANULE == SIZE_FROM_LOG2_WORDS(val));
          :             :         else
          :             :                 assert(CACHE_WRITEBACK_GRANULE <= MAX_CACHE_LINE_SIZE);
          :             : #endif /* ENABLE_ASSERTIONS */
          :             :
          :             :         /* Perform remaining generic architectural setup from EL3 */
          :           1 :         bl1_arch_setup();
          :             :
          :             : #if TRUSTED_BOARD_BOOT
          :             :         /* Initialize authentication module */
          :             :         auth_mod_init();
          :             : #endif /* TRUSTED_BOARD_BOOT */
          :             :
          :             :         /* Perform platform setup in BL1. */
          :           1 :         bl1_platform_setup();
          :             :
          :             :         /* Get the image id of next image to load and run. */
          :           1 :         image_id = bl1_plat_get_next_image_id();
          :             :
          :             :         /*
          :             :          * We currently interpret any image id other than
          :             :          * BL2_IMAGE_ID as the start of firmware update.
          :             :          */
 [ -  + ]:           1 :         if (image_id == BL2_IMAGE_ID)
          :             :                 bl1_load_bl2();
          :             :         else
          :           1 :                 NOTICE("BL1-FWU: *******FWU Process Started*******\n");
          :             :
          :           1 :         bl1_prepare_next_image(image_id);
          :             :
          :           1 :         console_flush();
          :           1 : }
          :             :
          :             : /*****************************************************************************
```

Capture Phase - Details

Model Trace Interface Plugin

- Instantiate the MTI plugin instance
- Register plugin instance with Simulation
- Discover a trace source "INSTR"
- Register callback handler to record trace "field" capture in memory
- At termination dump the trace info from memory to file

Useful Reference - Model Trace Interface Reference Manual v1.1

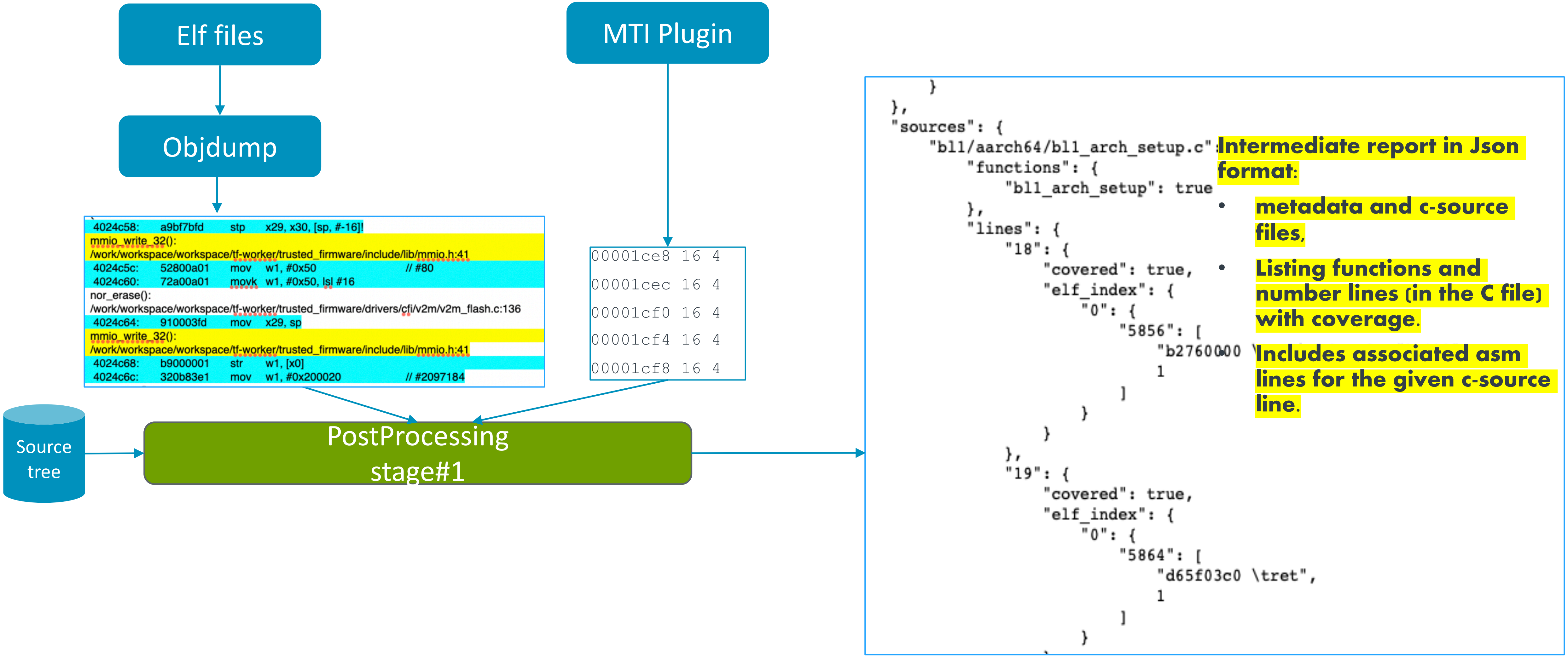
https://gitlab.arm.com/tooling/qa-tools/-/blob/master/coverage-tool/docs/plugin_user_guide.md

```
PREPARE:: Building the model plugin
-------------------------
make -C model-plugin PVLIB_HOME=/path/to/modellib
For TF-A CI:
PVLIB_HOME=$warehouse/SysGen/PVModelLib/$model_version/$model_build/external
Toolchain: aarch64-linux-gnu (we reused the same used by their CI)
Objects created: CoverageTrace.so, CoverageTrace.o, PluginUtils.o

EXECUTE:: Capturing a trace
-----------------
You need to add two options to your model command-line:
   --plugin /path/to/CoverageTrace.so
   [-C TRACE.CoverageTrace.trace-file-prefix="/path/to/TRACE-PREFIX"]
Example from TF-A CI:
/arm/warehouse/SysGen/Models/11.6/45/models/Linux64_GCC-4.9/FVP_Base_RevC-2xAEMv8A \
--data cluster0.cpu0=el3_payload.bin@0x80000000 \
--data cluster0.cpu0=ns_bl1u.bin@0x0beb8000 \
--plugin=/work/workspace/workspace/tf-worker/test-
definitions/scripts/tools/code_coverage/fastmodel_baremetal/bmcov/model-
plugin/CoverageTrace.so \
-C bp.flashloader0.fname=fip.bin \
-C bp.secureflashloader.fname=bl1.bin \
-C bp.ve_sysregs.exit_on_shutdown=1 \
-C pctl.startup=0.0.0.0  -Q 1000   "$@"

OUTPUT:: Coverage Trace sample output:
-----------------------------
00001ce8 16 4
00001cec 16 4
00001cf0 16 4
00001cf4 16 4
00001cf8 16 4
```


arm

Analysis Phase - Details

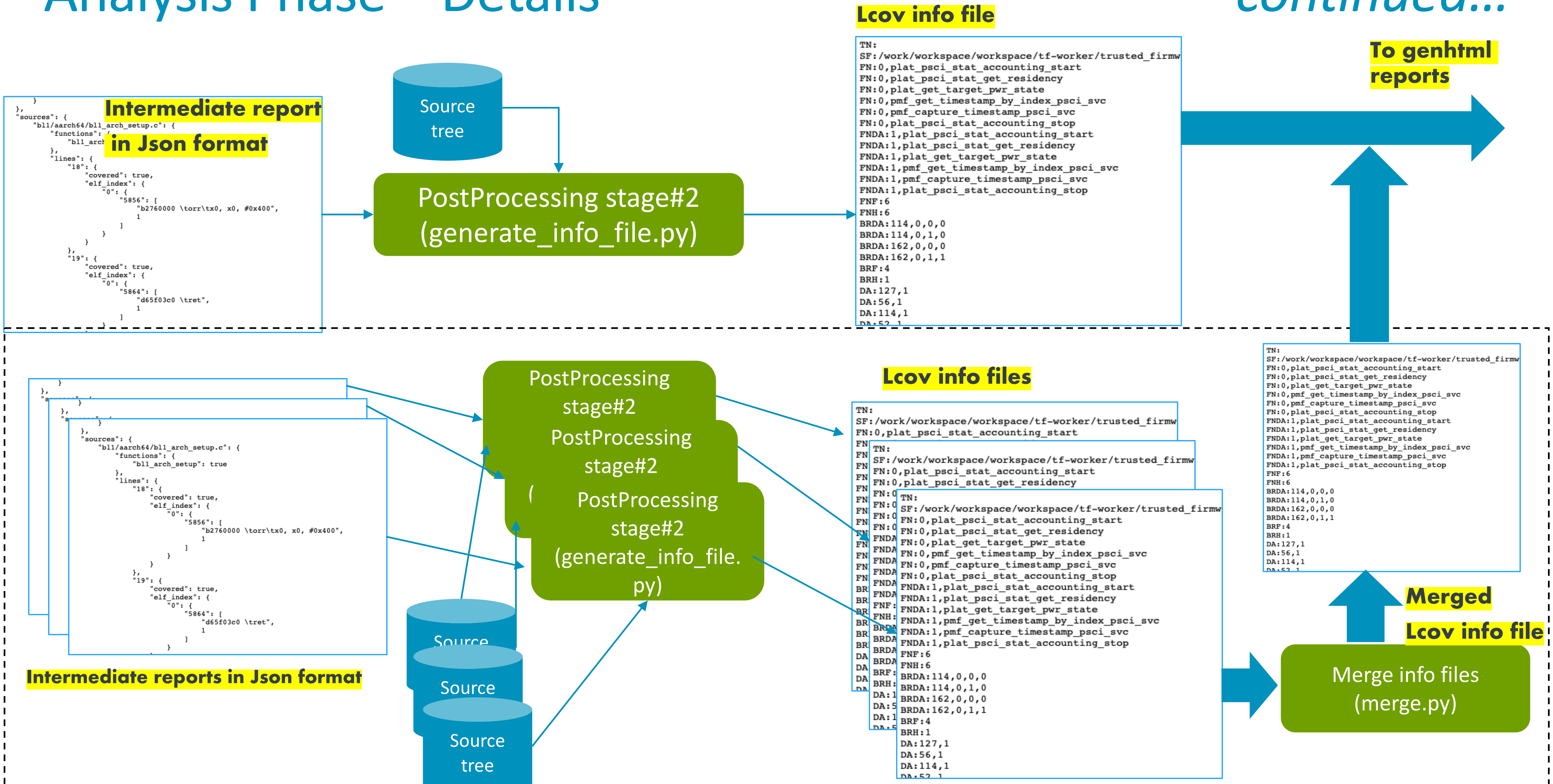
https://gitlab.arm.com/tooling/qa-tools/-/blob/master/coverage-tool/docs/reporting_user_guide.md



arm

Analysis Phase – Details

continued…



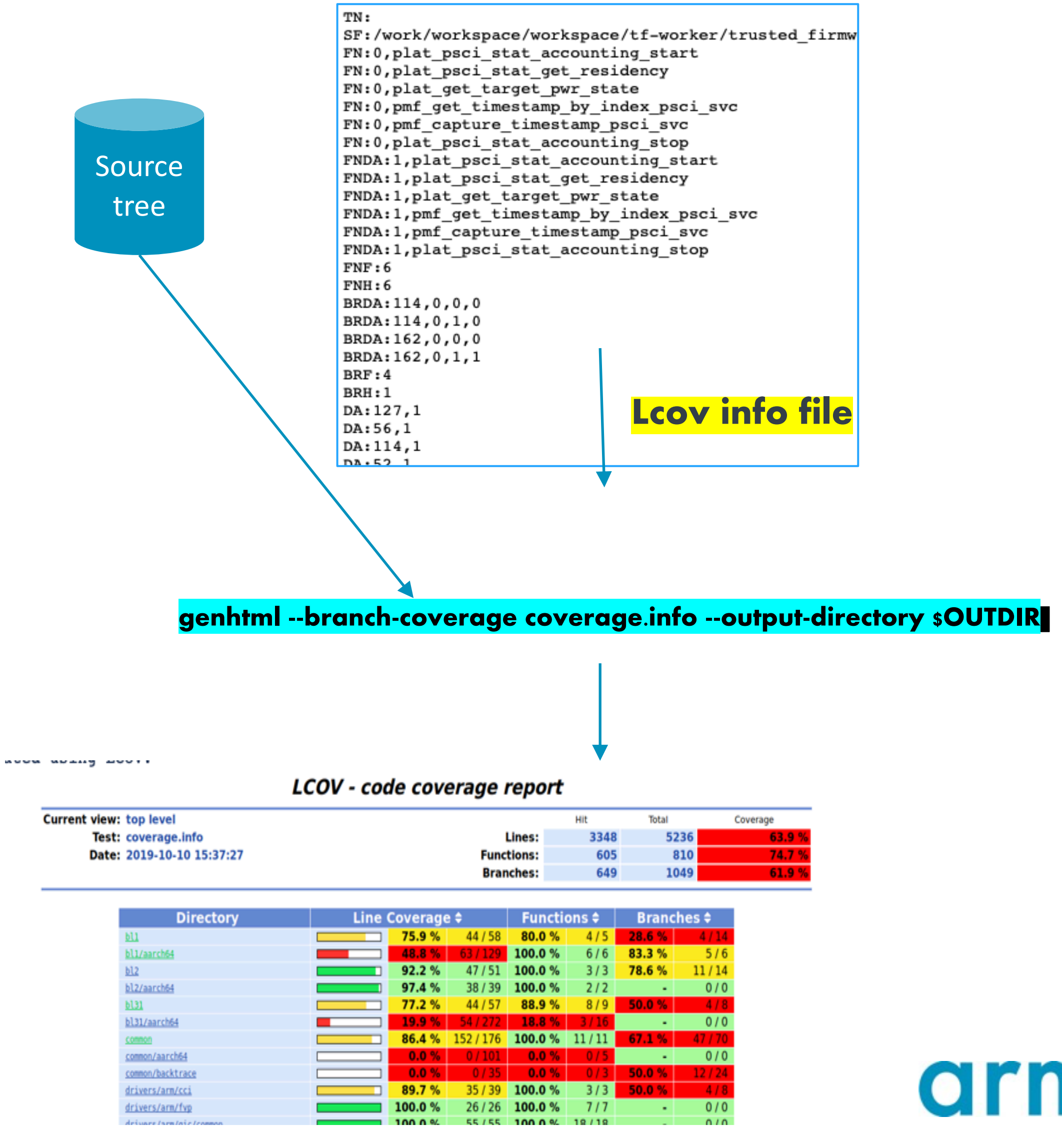
arm

Visualisation Phase - Details

https://gitlab.arm.com/tooling/qa-tools/-/blob/master/coverage-tool/docs/reporting_user_guide.md

The LCOV open source project (http://ltp.sourceforge.net/coverage/lcov.php) for visualisation.

- Starting from the JSON file a .info (LCOV) file(s) is generated
- The HTML code is produced starting from the .info file and the original C source code.
  - includes information about line, function and branch coverage
  - allows to browse through the source files and check their coverage.



	Hit	Total	Coverage
Lines:	3348	5236	63.9 %
Functions:	605	810	74.7 %
Branches:	649	1049	61.9 %

Directory	Line Coverage		Functions		Branches	
bl1	75.9 %	44 / 58	80.0 %	4 / 5	28.6 %	4 / 14
bl1/aarch64	48.8 %	63 / 129	100.0 %	6 / 6	83.3 %	5 / 6
bl2	92.2 %	47 / 51	100.0 %	3 / 3	78.6 %	11 / 14
bl2/aarch64	97.4 %	38 / 39	100.0 %	2 / 2	-	0 / 0
bl31	77.2 %	44 / 57	88.9 %	8 / 9	50.0 %	4 / 8
bl31/aarch64	19.9 %	54 / 272	18.8 %	3 / 16	-	0 / 0
common	86.4 %	152 / 176	100.0 %	11 / 11	67.1 %	47 / 70
common/aarch64	0.0 %	0 / 101	0.0 %	0 / 5	-	0 / 0
common/backtrace	0.0 %	0 / 35	0.0 %	0 / 3	50.0 %	12 / 24
drivers/arm/cci	89.7 %	35 / 39	100.0 %	3 / 3	50.0 %	4 / 8
drivers/arm/fvp	100.0 %	26 / 26	100.0 %	7 / 7	-	0 / 0

Gotchas and Learnings

Is there any catch?

- Optimisation levels (especially -Os) influence coverage stats
  - Only source lines with dwarf signature can yield coverage info
  - Optimisation can lead to functions be inlined or code removed from binary
- File encoding issues affects post processing
- Lexical analyser to help with source code parsing did not help
  - Finally used simple python text parsing logic
- Toolchain bugs affect coverage generation

Tooling Access and Usage

Where to get this tool from?

- Open sourced the MTI plugin implementation and the associated post processing scripts
  - https://gitlab.arm.com/tooling/qa-tools
- Any feedback or contributions very much welcomed.
  - See https://gitlab.arm.com/tooling/qa-tools/-/tree/master/coverage-tool#contributing
- Internally used for both TF-A, TF-M and SCP projects
  - TF-M project uses an early proof-of-concept workflow which uses LAVA setup

How can it help you?

- Tell you where to redirect your testing effort
- Address potential quality risks due to uncovered code-paths
- Data from the tool can be used to visualize ongoing coverage trend as your project evolves
- Can provide you with profiling data on executed instructions – potentially identify bottlenecks or need for better code reuse

arm

Future Direction

What more?

- Extend the trace-based coverage measurement methodology to Silicon platforms
  - Early prototype done with Juno platform
  - Feasible; but some automation challenges persist
- MC/DC coverage
  - We can dump register values in addition to instructions executed
  - Early prototype done to show the MTI extension; but more work needed
- Alternative to a custom plugin (MTI)
  - Few possibilities with some standard fastmodel trace extensions; Early exploration!

Q&A

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