METrICS behavioral traces collected on the FMS avionic use-case

© Thales SA

This document describes the behavioral traces collected on the Flight Management System avionic use-case running of the iMx8 board with the LinuxRT operating system. These traces are collected with the METrICS measurement framework in the form of a set of CSV files.

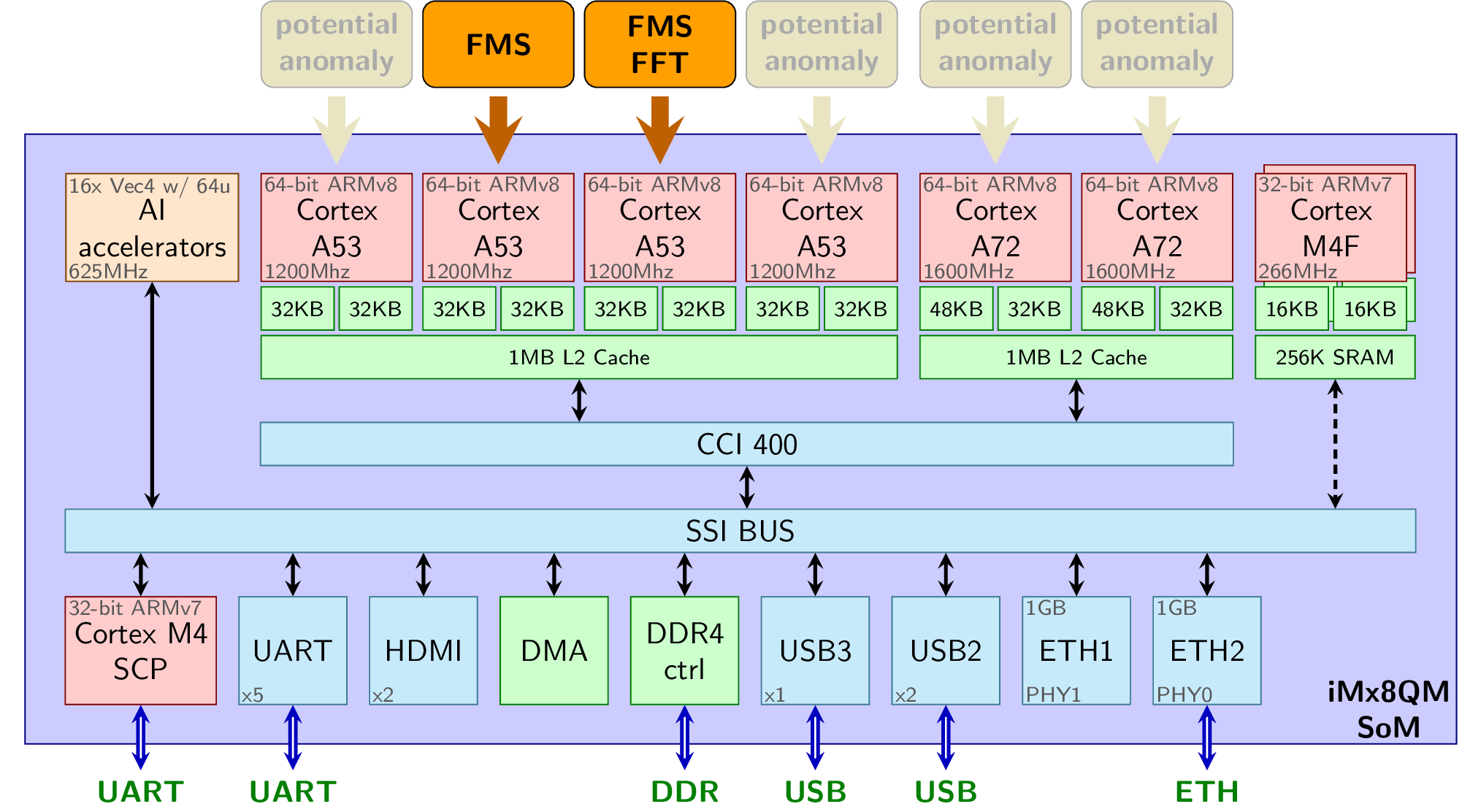
# Trace folder organization

Each folder corresponds to a full experimental campaign with the Fight Management System (FMS) application following a specific flight plan and confronted to a specific anomaly.

* For instance, LFBT\_LFBL corresponds to a fixed flight plan from TARBES Lourdes-Pyrénées (LFBT) to LIMOGES Bellegarde (LFBL) while being potentially confronted to a DDoS attack to the L2 cache

Each of such folder contains a set subfolders corresponding to different mapping of the avionics application as well as the stressing benchmarks / anomalies / cyber-attack deployments.

Each of these subfolder as a name that looks like ABCD.EF. The 4 first letters indicates which applications are deployed on the 4 dependable Cortex A53 cores of the iMx8 platform presented below, the last two letters on the application deployed on the non-dependable Cortex A72 cores.



Each letter indicates which application is deployed:

* x: Nothing is deployed on this core.
* F: The FMS is deployed on this core.
* T: The FFT computation is deployed on this core (filtering of sensor data for the FMS).
* S: A continuous **DDoS attack on the L2** is deployed on this core
* Z: An intermittent **DDoS attack on the L2** is deployed on this core.
* U: Both the FMS and a continuous **CPUTheft** anomaly running on this core.
* V: Both the FMS and a intermittent **CPUTheft** anomaly running on this core.
* I: A continuous **Spectre attack** is deployed on this core
* J: An intermittent **Spectre attack** is deployed on this core
* K: Both the FMS and a continuous Spectre**t** attack running on this core.
* B: A continuous **branch predictor hammering** anomaly is deployed on this core
* C: An intermittent **branch predictor hammering** anomaly is deployed on this core
* D: Both the FMS and another infected application with continuous **branch predictor hammering** anomaly running on this core
* E: FMS running on this core while itself being internally infected with the continuous **branch predictor hammering** anomaly
* G: FMS running on this core while itself being internally infected with the intermittent **branch predictor hammering** anomaly

So, for instance;

* **xFTx.xx** corresponds to the avionic use-case running standalone in the system, serving as a nominal / normal reference, without any security-related anomalies.
* **SFTS.xx** and **SFTS.SS** correspond to maximum stress scenarios with permanent anomalies due to DoS attack on the memory resource. The first one only issue stress on the neighboring Cortex A53 cores, whereas the second one extends this to the Cortex A72 cores.
* **ZFTZ.xx** and **ZFTZ.ZZ** correspond to the same scenario as above, but replacing the continuous stress by intermittent stress pausing and restarting every 15 seconds approximately.
* a **README.MD** file usually describes the specifics in each subfolder.

For each execution of the FMS with a fixed set of parameters for both the FMS and METrICS, three output trace files are generated:

* run\_##.csv: Information about the parameters of the current execution, such as which hardware events are collected by METrICS, and so on...
* raw\_##.csv: Raw collected trace data. Each row of the file correspond to a time serie that correspond to a pair of probes in the source code. These probes are positioned around the software periodic tasks to collect behavioral information about these tasks.
* log\_##.csv: user provided trace information, currently empty, might be used in the future to drop TEACHING KPI related information.

# Trace collection campaign

A complete tar.gz file is currently composed of 50 of theses triplets, corresponding to 50 execution of the FMS with different FMS/METrICS settings, with the following pseudo-code script:

$N=0  
for config in [0,nb\_metrics\_config]   
 for it in [0,nb\_fms\_iterations]   
 run the FMS   
 METrICS dumps collected info in run\_$N.csv, raw\_$N.csv and log\_$N.csv   
 $N++

**nb\_fms\_iteration:** The number of FMS iteration (used for statistical purpose) is currently 10, and traces of different iterations of the same METrICS setting should provide similar results. Note that is is easy for me to increase this number, it just implies longer experimental times.

**nb\_metrics\_config:** Number of different possible configuration of METrICS. Each of these configurations correspond to a different set of hardware events being collected (and hence a different meaning of the PMC columns in the raw.csv file). As each core support 6 performance monitor counter, each configuration corresponds to a set of 6 hardware events.

In the provided trace files, 5 different configurations have been used:

* { L1D\_CACHE, L1D\_CACHE\_REFILL, L1D\_CACHE\_WB, L2D\_CACHE, L2D\_CACHE\_REFILL, L2D\_CACHE\_WB} that focuses on the memory data path and caches
* { INST\_RETIRED, LD\_RETIRED, ST\_RETIRED, BR\_PRED, BR\_MIS\_PRED, PREFETCH} that focuses on application characterization in terms of instruction type
* { LD\_SPEC, ST\_SPEC, BR\_PRED, DP\_SPEC, VFP\_SPEC, L1I\_CACHE\_REFILL } that focuses on the speculative aspects of the architecture, especially for the A72 core.
* { LD\_SPEC, ST\_SPEC, L1D\_CACHE, INST\_SPEC, L1I\_CACHE, L1I\_CACHE\_REFILL } that focuses on speculative memory and control flow accesses.
* { L1D\_TLB\_REFILL, L1I\_TLB\_REFILL, L1D\_CACHE, L1I\_CACHE, BUS\_ACCESS, MEM\_ACCESS } that focuses on the TLB / MMU aspects.

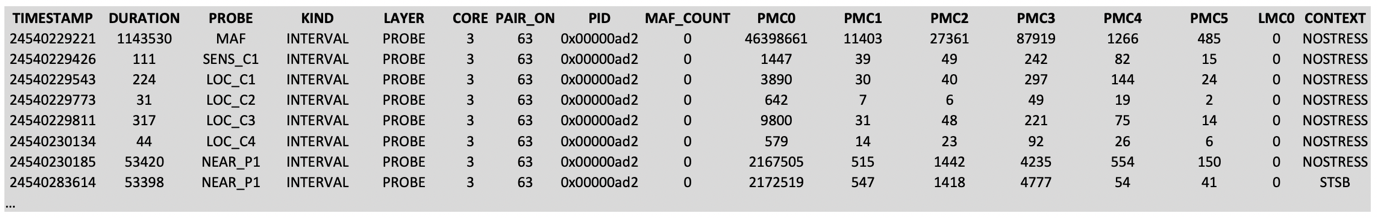
The hardware perfromance events are further detailed in the official documentation, Section 12.4.2 of doc/cortex\_a53\_trm.pdf and Section 11.8 of doc/cortec\_a72\_trm.pdf.

More hardware event could be taken into account. Defining the meaningful event set is I guess part of discussion between WP5 and WP4.

# Trace file format

In this section, we explain how to read the collected traces of behavioral and timing information. To do so, we refer to 2 of the generated trace files: raw.csv and run.csv, as well as a third hw\_events.csv file that is hardware architecture dependent, listing the available hardware events that could be probed:

## Raw.csv



* time series: one per row
* data already paired and corresponding to intervals (no more required to pair being events with end events)
* Relevant hardware event information: timestamp, duration and PMC#
* Timestamp: in #cycle since boot
* Duration: In cycle if KIND=interval, else 0
* Probe: Name of the software task if LAYER=probe
* Kind: probe kind: Interval, Begin or End
* Layer: Probe for software probe inserted in the code.
* Core: Number of the core the probe where executed on
* Pair\_on: (please ignore, internal usage while pairing begin and end probes)
* Pid: process ID of the monitored application
* Maf\_count: (please ignore, sometime used to gather the iteration number in the monitored periodic software)
* PMC#: Collected performance monitor counter (check run.csv to known which one)
* LMC#: Collected monitor counter from the software (might later be used to store power, current, temperature information)
* Context: Information about co-running context such as co-running stressing benchmarks or cyber-attacks.
  + NoStress indicates a standalone execution without anomalies
  + STSB: Indicates the anomaly of co-running with an application performing a DoS attack on memory resource producing a large number of store requests inducing cache misses.
  + LDSB: Indicates the anomaly of co-running with an application performing a DoS attack on memory resource producing a large number of load requests inducing cache misses.

## Aggregating data

