D2.11 Final user manual for the industry-ready RTOS multicore solutions

Version 1.2

Document Information

Contract Number	611085
Project Website	www.proxima-project.eu
Contractual Deadline	M36, 30-November-2016
Dissemination Level	RE
Nature	O
Author	Francis Vatrinet (SYS)
Reviewer	Iain Bate (UoY)
Keywords	RTOS baseline, User manual, PikeOS

Notices:

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement n° 611085.

Change Log

Version	Description of Change
v0.1	Initial Draft released for internal review at BSP v1.2.3 delivery
V0.2	Draft Add FPGA bitstream identification and constraint at installation section
V0.3	BSP v1.3.0 early upgrades for testing: - New Built In Test API and Time Composability Checker tool - Add configuration info about FPGA serial line in section 4.1. - change in FPGA-HW-Randomization configurable seeds initialisation - PAK: add instrumentation for performance counters, - Configuration to put kernel code and data into no cache area for time composability
V0.4	BSP upgrade: - add PRNG for user level, - TC checker : added and documented calibration method. - adding the math library into the apex application - adding the math library to the native application
V0.5	BSP v1.2.5: - Add P4080 PSP - Update demonstration projects to be architecture independent - Update LEON3 PSP (configuration only for stat driver declaration) - Remove PikeOS 3.5 Support PR#1473-Projects Add math library (libm) for PikeOS 3.4
V0.6 draft	Roadmap update due to FPGA availability BSP v1.3.0 upgrades : - PSP kinfo page for user level fast access - Add kernel TAG 0x308 to run kernel with NO cache attribute BSP v1.3.1 upgrades : - support for ABPRANDBANK v3 - add instrumentation callback for RVS ipoints - Time Composability : run tp0 into core 1 to reduce interference with running partition PRNG provides random value cross reboot using timer value at initialization time
V0.7	BSP v1.3.2 upgrades/fixes :

	- Adding get_MAF_time() (requires dynamic ticker on FPGA) PRNG shall return current seed
V0.8	BSP v1.3.5 upgrades/fixes : - add events and perf measurement to proxima-pak demo.
V0.9	BSP v1.4.0 upgrades/fixes : - taking into account integration findings and new requirements for VICI step - MBTA: how to remove undefined reference raised at link time prng_set_BP2 shall now support 8 bits window size
V1.0	Internal review input
V1.1	Taking into account review comments: - add more gentle introduction on "how to read this document in the context of PROXIMA" => rework Executive Summary section - need to provide clear, itemized list of the technology features that were added to the SYSGO technology to meet the PROXIMA needs should appear early on in the document with pointers to where to find the corresponding descriptions. => rework section 5, change some sections' title accordingly. - Add sub section "How to populate HW with a given seed value from test code" into section 6.3.1.3.
V1.2	Taking into account final review comments: - include a clear legend of the notes accompanying the SYSGO internal analysis about compliancy of the PROXIMA results to the PikeOS industrial constraints => change executive summary and section 7 accordingly - copy-paste issues from previous deliverables are to be addressed => remove deliverables issues from change log that are not related to the document itself

Table of Content

1 Executive Summary

This deliverable presents the final user manual of the PikeOS RTOS baseline that was used to implement the use case studies in the PROXIMA project for both FPGA and COTS multi-core targets. A prominent objective in PROXIMA is to enable Probabilistic Timing Analysis (PTA) on both PTA and non-PTA-compliant processors and this documentation specifically covers how to use the extensions to PikeOS that implement the necessary features such as randomization and tracing to support these timing analysis methods. .

Those use cases illustrates the way to implement Measurement Based PTA using PikeOS in a mixed criticality environment. PikeOS is not intended to provide any probabilistic timing behavior but will provide some Time Composability features allowing both standard and probabilistic analysis at application level.

This document is not the overall timing analysis tool chain documentation that should come from tool providers. It is not a safety manual but provides the elements that have been added to PikeOS in the frame of PROXIMA to enable Measurement Based Probabilistic Timing Analysis of critical application in mixed criticality environment.

This deliverable is structured as follows:

- The section 2 “Installation” provides installation procedure for both FPGA and COTS targets using the deliveries provided into the directory CODE/SYSGO of the PROXIMA SVN project.
- The section 3 “Installation verification” provides installation verification procedure for both FPGA and COTS targets using the deliveries provided into the directory CODE/SYSGO of the PROXIMA SVN project.
- The section 4 “SYSGO documentation” provides the PikeOS product line generic documentation for both FPGA (LEON3 architecture) and COTS (P4080 architecture) used in the frame of PROXIMA project.
- The section 5 “PikeOS features developed for PROXIMA” provides a matrix of features added or configuration tricks for the PROXIMA project with a link to the user documentation and a link to its PROXIMA contribution description.
- The section 6 “PikeOS tool chain specific documentation for PROXIMA” provides the necessary project specific add on to the PikeOS product line documentation for both FPGA (LEON3 architecture) and COTS (P4080 architecture) used in the frame of PROXIMA project.
- The section 7 of this document “PikeOS Time Composability and MBPTA” is probably the most important one explaining how to go to Time Composability for some services and to globally reduce jitters. For any of the features there is the SYSGO internal analysis about compliancy of the PROXIMA results to the PikeOS industrial constraints through colored notes; the feature may be implemented in one or more components from one or several contributors:

The following convention is used:

- Green: when feature implementation is compliant with industrial need
- *Italic orange: when feature partially implemented or partially compliant with the industrial need*
- **Bold red: for implementations that are not compliant with industrial need.**

2 Installation

The PROXIMA baseline provided by SYSGO is based on ELinOS and PikeOS SYSGO's products to be installed first on the development host.

At M24 the following elements can be used:

- PikeOS 3.4 for both FPGA and COTS P4080 is requested first
    - PikeOS on ftp@sysgo.com
        - R3p4_PIKEOS_MT_LIN_PPC_E500MC_S3725.iso
        - R3p4_PIKEOS_MT_WIN_ PPC_E500MC _S3725.iso
        - R3p4_PIKEOS_MT_LIN_SPARC_V8_S3725.iso
        - R3p4_PIKEOS_MT_WIN_SPARC_V8_S3725.iso
    - BSPs 1.4.0 in http://svn.proxima-project.eu/repos/proxima/Code/SYSGO:
        - R3p4_PIKEOS_DK_LIN_BSP_PROXIMA_FPGA_3.4_S**4759**
        - R3p4_PIKEOS_DK_WIN_BSP_PROXIMA_FPGA_3.4_S**4759**
- ELinOS 5.2
    - PikeOS on ftp@sysgo.com
        - R5p2_ELINOS_DK_CMB_PPC_E500MC_PIKEOS_S3737.iso
        - R5p2_ELINOS_DK_CMB_PPC_E500MC_ S3737.iso
        - R5p2_ELINOS_DK_CMB_ SPARC_V8_PIKEOS_S3289.iso
        - R5p2_ELINOS_DK_CMB_ SPARC_V8_ S3289.iso

3 Installation verification

As the PikeOS installation includes several added components, it is recommended to verify your installation as presented next.

3.1 For PikeOS BSP v1.4.0 for FPGA running with PikeOS 3.4:

Run the following command:

```
$ /opt/pikeos-3.4/bin/pikeos-version
```

And check you have at least the following lines:

```
PikeOS Version 3.4 (05-10-2013)
Host: …

Installed CDKs and Products:
  SPARC_V8    (R3p4_PIKEOS_DK_LIN_SPARC_V8_BASE_S3725)
              (R3p4_PIKEOS_DK_LIN_SPARC_V8_APEX_S3725)
              (R3p4_PIKEOS_DK_LIN_SPARC_V8_INTEGRATION_S3725)
              (R3p4_PIKEOS_DK_LIN_SPARC_V8_PIKEOS_S3725)
              (R3p4_PIKEOS_DK_LIN_SPARC_V8_RTEMS_S3725)
              (R3p4_PIKEOS_DK_LIN_SPARC_V8_UKERNEL_S3725)
  noarch      (R3p4_PIKEOS_DK_LIN_BSP_PROXIMA_FPGA_3.4_S4759)

Installed PSPs:
  pikeos-psp-leon3-proxima-smp-sparc_v8-3.4-96

…
```

Run the following command:

```
$ ls -1 /opt/pikeos-3.4/demos*
```

And check you have at least the following lines:

```
/opt/pikeos-3.4/demos-apex:
-

/opt/pikeos-3.4/demos-integration:
proxima-ibit
proxima-pak
portprovider-tc

/opt/pikeos-3.4/demos-pikeos:
proxima-ibit
proxima-pak
port-provider-client
portprovider-tc-loop
portprovider-tc-seed
```

3.2 For PikeOS BSP v1.4.0 for COTS P4080 running with PikeOS 3.4:

Run the following command:

```
$ /opt/pikeos-3.4/bin/pikeos-version
```

And check you have at least the following lines:

```
PikeOS Version 3.4 (05-10-2013)
Host: …

Installed CDKs and Products:
  PPC_E500MC (R3p4_PIKEOS_DK_LIN_PPC_E500MC_BASE_S3725)
              (R3p4_PIKEOS_DK_LIN_PPC_E500MC_APEX_S3725)
              (R3p4_PIKEOS_DK_LIN_PPC_E500MC_INTEGRATION_S3725)
              (R3p4_PIKEOS_DK_LIN_PPC_E500MC_PIKEOS_S3725)
              (R3p4_PIKEOS_DK_LIN_PPC_E500MC_POSIX_S3725)
              (R3p4_PIKEOS_DK_LIN_PPC_E500MC_UKERNEL_S3725)
  noarch      (R3p4_PIKEOS_DK_LIN_BSP_PROXIMA_FPGA_3.4_S4759)
```

```
Installed PSPs:
  pikeos-psp-debug-smp-p4080-proxima-ppc_e500mc-3.4-38
  pikeos-psp-smp-p4080-proxima-ppc_e500mc-3.4-38
…
```

Run the following command:

```
$ ls -1 /opt/pikeos-3.4/demos*
```

And check you have at least the following lines:

```
/opt/pikeos-3.4/demos-apex:
-

/opt/pikeos-3.4/demos-integration:
proxima-ibit
proxima-pak
portprovider-tc

/opt/pikeos-3.4/demos-pikeos:
proxima-ibit
proxima-pak
port-provider-client
portprovider-tc-loop
portprovider-tc-seed
```

4 SYSGO documentation

The PikeOS RTOS has been designed to share as far as possible the same components into its multiple implementations and so for the documentation and tool chain. The PikeOS distribution comprises multiple components that can be combined. In the frame of PROXIMA the following will be used:

- Integrator Suite
- Application Suite PikeOS
- Application Suite ELinOS
- Application Suite APEX (ARINC 653)
- Application Suite RTEMS

PikeOS is shipped with the PikeOS Installation Guide and a copy of the document "Using PikeOS". The rest of the documentation is provided in electronic form (PDF) and can be found, after product installation, in /opt/pikeos-${PIKEOS_VERSION}/documentation/. The directory /doc of the PikeOS Product Installation Media contains a version of this document, as well as the PikeOS Release Notes. The following sections provide an overview of the major documentation groups and the individual documents within each group.

The ELinOS personality is considered as a BSP of the ELinOS product, then its documentation is part of ELinOS installation in /opt/elinos-5.2/doc

Release Notes

Release notes describe differences, corrections and enhancements done to prior releases. If you have been using PikeOS/ELinOS products before, or if you want to migrate existing projects to the current software release, you should study the release notes carefully to update your knowledge and to avoid pitfalls.

Table 3.1-1 – PikeOS and ELinOS Release Notes.

	Description
releasenotes-pikeos-v${PIKEOS_VERSION}.pdf	PikeOS: Depending on the components you have installed, there may be other, personality specific release notes
releasenotes.pdf	ElinOS specific release note

User Manuals

This group covers the basic information about installation, basics and first steps with PikeOS/ELinOS and offers useful tutorials for a guided start.

Table 3.1-2 – PikeOS/ELinOS User Manuals.

	Description
usingpike.pdf	Using PikeOS
fundamentals. Pdf	PikeOS Fundamentals
tutorials.pdf	PikeOS Tutorials
ElinOS-Users-Guide.pdf	The ElinOS users guide

The PikeOS Tutorials mainly applies to the Integrator Suite. However, the manual is shipped with each application suite, because selected portions are useful for the application developer as well.

Reference Manuals

This group contains reference information on specific PikeOS components

Table 3.1-3 – PikeOS Reference Manuals.

	Description
kernelref.pdf	PikeOS Kernel Reference Manual
psswref.pdf	PikeOS System Software Reference Manual
ddkref.pdf	PikeOS Device Driver Programming Reference Manual Manual
pspdevguide.pdf	PikeOS PSP Developer’s Guide
libstand.pdf	PikeOS Standalone Utility

Platforms Manuals

PikeOS supports more than ten processor architectures. Platform specific information is available in the respective PikeOS Platform Manual. These manuals are named plat[architecture].pdf. Note that for ELinOS the PikeOS is a platform therefore there is no ELinOS personality documentation.

Table 3.1-4 – PikeOS / ELinOS Platforms Manuals.

	Description
platppc-e500mc.pdf	PikeOS Freescale QorIQ P5040/P4080/P2041
platsparc_v8.pdf	PikeOS QEMU SPARC, LEON3 and LEON4
ELinOS-Platform-Manual	ELinOS platform Manual, Includes a section for PikeOS as a target.

Personality Manuals

Table 3.1-5 – PikeOS Personality Manuals.

	Description
apex.pdf	PikeOS Personality Manual: APEX
rtems.pdf	PikeOS Personality Manual: RTEMS

5 PikeOS features developed for PROXIMA

The following table provides a status and roadmap of Time Composability features the PikeOS environment provided for the PROXIMA project:

Legend:

- Red cell stands for a feature not working, the PikeOS BSP has been delivered only to ease hardware issue under debug; no possibility to validate delivery.
- Orange cell stands for a feature not working, the PikeOS BSP has been delivered to ease hardware issue under debug. Delivery partially validated.

Feature / availability	M8 6/2014	M12 10/2014	M14 12/2014	M15 1/2015	M20 6/2015	M20 6/2015	M24 10/2015	M25 11/2015	M27 01/2016	M30 04/2016	M32 06/2016	M33 07/2016	Native	RTEMS	APEX	APEX TC	ELinOS	User Manual	PROXIMA contribution Description
BSP v1.0 PikeOS 3.4 only	x												X	V4.1	P1s1		V5.2	D2.1 v1.1	D2.1 v1.1
BSP v1.1 PikeOS 3.5 only		x											X					D2.1 v1.2	D2.1 v1.2
BSP v1.2.0 PikeOS 3.4			x										X	X	X	X	X	D2.1 v1.3	D2.1 v1.3
BSP v1.2.0 PikeOS 3.4			x										X		X	X		D2.1 v1.3	D2.1 v1.3
BSP v1.2.1 PikeOS 3.4				x									X	X	X	X	X	D2.1 v1.4	D2.1 v1.4
BSP v1.2.1 PikeOS 3.5				x									X		X	X		D2.1 v1.4	D2.1 v1.4
BSP v1.2.2 PikeOS 3.5					x								X		X	X		D2.1 v1.4	D2.1 v1.4
BSP v1.2.3 PikeOS 3.5						x							X		X	X		D2.11 v0.1	D2.11 v0.1
BSP v1.2.4 PikeOS 3.5							X											D2.11 v0.2 D2.11 v0.3 D2.11 v0.4	Patch for HW debug
BSP v1.2.5 PikeOS 3.4							X											D2.11 v0.5	Patch for HW debug
BSP v1.3.0 PikeOS 3.4								X					X	X	X	X	X	D2.11 v0.6	Not delivered waiting for stable HW

Feature / availability	M8 6/2014	M12 10/2014	M14 12/2014	M15 1/2015	M20 6/2015	M20 6/2015	M24 10/2015	M25 11/2015	M27 01/2016	M30 04/2016	M32 06/2016	M33 07/2016	Native	RTEMS	APEX	APEX TC	ELinOS	User Manual	PROXIMA contribution Description
BSP v1.3.1 PikeOS 3.4									X				X	X	X	X	X	D2.11 v0.6	D2.11 v0.6
BSP v1.3.2 PikeOS 3.4										X			X	X	X	X	X	D2.11 v0.7	D2.11 v0.7
BSP v1.3.3 PikeOS 3.4										X			X	X	X	X	X	D2.11 v0.8	Patch only
BSP v1.3.4 PikeOS 3.4										X			X	X	X	X	X	D2.11 v0.8	Patch only
BSP v1.3.5 PikeOS 3.4											X		X	X	X	X	X	D2.11 v0.8	D2.11 v0.8
BSP v1.4.0 PikeOS 3.4												X	X	X	X	X	X	D2.11 v1.0	D2.11 v1.0
FPGA bitsream v1.4	x	x																APBRANDBANK V1	APBRANDBANK V1
FPGA bitsream v1.6			x	x	x													APBRANDBANK V1	APBRANDBANK V1
FPGA bitsream v1.7						x												APBRANDBANK V1	APBRANDBANK V1
FPGA bitsream v1.8						x												APBRANDBANK V1	Not working
FPGA bitsream v1.9 a,b,c							X											APBRANDBANK V1	Debug versions
FPGA bitsream v2.0							▨	X										APBRANDBANK V2	Not working
FPGA bitsream v2.1								▨	X	X	X							APBRANDBANK V2	PTA not working
FPGA bitsream v2.9								▨	▨	X	X							APBRANDBANK V2	PTA not working
FPGA bitsream v3.3								▨	▨		X							APBRANDBANK V3	PTA not working
FPGA bitsream v3.4								▨	▨		▨	X						APBRANDBANK V3	APBRANDBANK V3

Feature / availability	M8 6/2014	M12 10/2014	M14 12/2014	M15 1/2015	M20 6/2015	M20 6/2015	M24 10/2015	M25 11/2015	M27 01/2016	M30 04/2016	M32 06/2016	M33 07/2016	Native	RTEMS	APEX	APEX TC	ELinOS	User Manual	PROXIMA contribution Description
PAK support (init, run, alloc)	x	x	x	x	x	x	X	X	X	X	X	X	X	X	X	X		D2.11 v1.1 § 6.3.1	D2.11 v1.1 § 6.3.1
PAK extension (precise time)	x	x	x	x	x	x	X	X	X	X	X	X	X	X	X	X		D2.11 v1.1 § 6.3.1.2	D2.11 v1.1 § 6.3.1.2
PAK: user level persistent PRNG for software randomization						x	X	X	X	X	X	X	X	X	X	X		D2.11 v1.1 § 6.3.1.3	D2.11 v1.1 § 6.3.1.3
PAK: demo provides events and measurements											X	X						D2.11 v1.1 § 6.4.1 & § 6.4.6.2	D2.11 v1.1 § 6.4.1 & § 6.4.6.2
PRNG: support at BSP level for Apbrandbank				x	x	x	X	X	X	X	X	X	X	X	X	X	X	D2.11 v1.1 § 6.3.1.3	D2.11 v1.1 § 6.3.2.1.1
PRNG: configurable BSP seed					x	x	X	X	X	X	X	X	X	X	X	X	X	D2.11 v1.1 § 6.3.1.3	D2.11 v1.1 § 6.3.2.1.1
PRNG: power OFF persistent BSP seed					x	x	X	X	X	X	X	X	X	X	X	X	X	D2.11 v1.1 § 6.3.1.3	D2.11 v1.1 § 6.3.2.1.1
PRNG: BP2 seed support									X	X	X	X	X	X	X	X	X	D2.11 v1.1 § 6.3.1.3	D2.11 v1.1 § 6.3.2.1.1
PRNG: BP2 seed support for 8 bits window size												X	X	X	X	X	X	D2.11 v1.1 § 6.3.1.3	D2.11 v1.1 § 6.3.2.1.1
Apbrandbank support at Partition switch				x	x	x	X	X	X	X	X	X	X	X	X	X	X	D2.11 v1.1 § 6.3.2.1.3	D2.11 v1.1 § 6.3.2.1.3
Apbrandbank support at APEX process switch				x	x	x	X	X	X	X	X	X				X		D2.11 v1.1 § 6.3.2.1.3	D2.11 v1.1 § 6.3.2.1.3

Feature / availability	M8 6/2014	M12 10/2014	M14 12/2014	M15 1/2015	M20 6/2015	M20 6/2015	M24 10/2015	M25 11/2015	M27 01/2016	M30 04/2016	M32 06/2016	M33 07/2016	Native	RTEMS	APEX	APEX TC	ELinOS	User Manual	PROXIMA contribution Description
Apbrandbank use random value for any seed change instead of keeping same along a time frame.								X	X	X	X	X				X		D2.11 v1.1 § 6.3.2.1.3	D2.11 v1.1 § 6.3.2.1.3
MBTA support, interface for writing ipoints to RTBx device or memory buffer								X	X	X	X	X	X					D2.11 v1.1 § 6.4.1	D2.11 v1.1 § 6.4.1
FPGA L2 Cache configurable			x	x	x		X	X	X	X	X	X	X	X	X	X	X	D2.11 v1.1 § 6.1.1	
FPGA L2 Cache partitioning per core			x	x	x			X	X	X	X	X	X	X	X	X	X	D2.11 v1.1 § 6.1.1	D2.11 v1.1 § 7.3.2.1.1
P4080 L2 Cache configurable			x	x	x		X	X	X	X	X	X	X	X	X	X	X	D2.11 v1.1 § 6.2.1	
P4080 L2 Cache partitioning per core			x	x	x			X	X	X	X	X	X	X	X	X	X	D2.11 v1.1 § 6.2.1	D2.11 v1.1 §7.3.2.1.2
P4080 L3 Cache configurable			x	x	x		X	X	X	X	X	X	X	X	X	X	X	D2.11 v1.1 § 6.2.1	D2.11 v1.1 §7.3.2.3
TC: Kernel code in no cache			x	x	x	x	X	X	X	X	X	X	X	X	X	X	X	D2.11 v1.1 § 6.4.2	D2.11 v1.1 §7.3.3.1
TC: Kernel data in no cache							X	X	X	X	X	X	X	X	X	X	X	D2.11 v1.1 § 6.4.2	D2.11 v1.1 §7.3.3.1

Feature / availability	M8 6/2014	M12 10/2014	M14 12/2014	M15 1/2015	M20 6/2015	M20 6/2015	M24 10/2015	M25 11/2015	M27 01/2016	M30 04/2016	M32 06/2016	M33 07/2016	Native	RTEMS	APEX	APEX TC	ELinOS	User Manual	PROXIMA contribution Description
TC: pssw and APEX TC lib code and data moved to no cache area						x	X	X	X	X	X	X				X		D2.11 v1.1 § 6.4.2 & 6.4.8.2 & § 6.4.11	D2.11 v1.1 §7.3.3.3.1
TC: pssw and Guest OS lib code and data moved to no cache area							X	X	X	X	X	X	X			X		D2.11 v1.1 § 6.4.2	D2.11 v1.1 §7.3.3.3.2
TC: APEX TC ports			x	x	x	x	X	X	X	X	X	X				X		D2.11 v1.1 § 6.4.8.2	D2.11 v1.1 §7.5.3
TC: Guest OS agnostic TC ports										X	X	X	X	X		X	X	D2.11 v1.1 § 6.4.11	D2.11 v1.1 §7.5.3
TC: Guest OS agnostic bit library provides calibration support and easy to use measurement methods						draft				X	X	X	X	X	X	X	X	D2.11 v1.1 § 6.4.6.2.1	D2.11 v1.1 § 7.4.3
TC: Tool checker						draft	draft	CICI	CICI	VICI	VICI	VICI	X		X	X		D2.11 v1.1 § "Time Composability checker tool – User Guide"	D3.12 v1.0
Performance counters access					x	x	X	X	X	X	X	X	X		X	X		D2.11 v1.1 § 6.4.6.2	
Workaround for libc/libsupc++					x	x	X	X	X	X	X	X	X		X	X		D2.11 v1.1 § 6.4.3	
PikeOS 3.4 libm support						x	X	X	X	X	X	X	X	X	X	X	X	D2.11 v1.1 § 6.4.4	

6 PikeOS tool chain specific documentation for PROXIMA

PikeOS tutorial as well as ELinOS user guide are based on the same method which consists in cloning a demonstration project to ease first step using the product.

In the frame of PROXIMA, for each of the case studies it will be allocated a demo project for the preconfigured partitions and a demo project for the integration role. All will be provided for both targets: FPGA and COTS (P4080). Every demo project includes a README document that helps understanding the content.

From BSP 1.3.X a new set of demo templates are provided to supply the Time Composability tool checker. They implement the various typical configurations as helper to the project.

6.1 PROXIMA-FPGA board

6.1.1 PSP PROXIMA-FPGA TAGS

The following sections is a complement to the kernelref.pdf and the platsparc_v8.pdf document when this PSP is project specific or added to the document when delivered as product line.

The following table lists the PSP tags and their default values for the leon3 PROXIMA-FPGA PSP for both PikeOS 3.4 and 3.5:

Tag	Description	Default value
PSP_CPU_CLOCK	This tag specifies the clock frequency of the LEON3 processor for bitstream v2.0 only	80 000 000 Hz
PSP_BAUDRATE	This tag specifies the baud rate for the console port. Possible values are 1200, 2400, 4800, 9600, 19200, 38400, 57600, and 115200.	38400
PSP_CONSOLE_PORT	This tag specifies the console output port. 1 directs the output to serial controller UART0. The value 0 disables the console output. **Please note the SLIDE_SW0 slide switch on the board shall be set OFF (right, 1) to select the APBUART system console (UART0).**	1

PSP_MEMSIZE	This tag specifies the size of the on-board RAM at physical address 0x40000000. Typically, this is the size of the SRAM in a LEON3 system. If the SRAM is disabled through the LEON3 memory configuration register 2 the SDRAM appears at 0x40000000.	0x40000000 (1GB)
PSP_L2_ENABLE	This tag allows controlling the L2 cache replacement policy. The tag values are as follows:	OFF

Value	**symbol**	**description**
0	OFF	Disabled, no seeds initialization
1	LRU	Least-recently-used replacement policy
2	IDX	deprecated
3	MOD	deprecated
4	RND	pseudo-random replacement policy (PTA mode is ON)
5	L1PTA	L2 disabled but PTA mode is ON for L1 and BP

Note when PTA switch is set for PTA mode, the L2 cache will be partitioned in 4 ways, one way per core, to avoid cache history interference between cores.

(see section 8.2.2.2 for more details about the configuration or the random replacement strategy)

Please note the SLIDE_SW2 slide switch on the board shall be set ON (left, 0) to select the PTA mode.

0x300	This tag specifies the size of the on-board AHBRAM at physical address 0x30000000. Typically, this is the size	0x00020000 (128kB)

	of the SDRAM in a LEON3 system, here it is Single-port RAM with AHB interface area. It is PROXIMA specific; when set it can be used to allocate the kernel-sram code reducing cache history impact.	
0x304	This tag specifies the APB address offset of the LEON3 timer unit.	0x300
0x305	This tag specifies the APB address offset of the LEON3 interrupt controller.	0x200
0x306	This tag specifies the APB address offset of the LEON3 UART0 serial controller.	0x100
0x307	This tag specifies the APB address offset of the LEON3 UART1 serial controller.	0x120
0x308	This tag specifies whether the kernel region is mapped to a cacheable (default 0) or not cacheable area (1). This region includes the BSP/kernel code and data area in their normal & tracesys configurations. The kernel section is larger than the region and will be partly not cached when activated.	0x0
0x309	This tag specifies the MBTA buffer size where to write events: - 0 when MBTA recording is not supported - 4 when using RTBx plugged to the GPIO port - Buffer of N * P4PAGE_SIZE for a valid buffer that is allocated in the main memory	0x4

	with no cache attribute	
0x310	This tag specifies the interrupt ID of the LEON3 timer unit 0.	8
0x320	Some LEON3 and LEON4 variants have a bug in their power-down support. Known boards with this bug are the GR-CPCI-UT699, the GR712RC, and the GRLEON4- ITX board. If this tag is set to 1 a workaround for the power-down bug is enabled in the PSP.	0

6.1.2 Application debug support

There are 2 ways for application debug within PROXIMA atop PikeOS;

The first one is the use of CODEO CDT debugger connected to the target through MUXA and the monitored application. When "make" an application with PikeOS, an unstripped version is also provided for the debug purpose you have to declare as application to the CODEO CDT. This is the standard way for source code level debug of algorithm, running either the real board or the QEMU simulator. A limitation exists as some guest OS does not provide the required monitoring; that is the case for RTEMS v4.10 and ELinOS but the second approach is usable in this case.

The second way is to use the board provider gdb (GRMON2/GRMON on FPGA board) to monitor application through JTAG/DSU device or a third part JTAG debug environment available for the COTS board. As for the first approach, you have to load the unstripped application into the gdb host side so that it can provide source level debug. This way is probably the preferred and most efficient one when running timing analysis or debugging at low level.

Note:
It has been discovered that using the emulated serial IO mode of the DSU through USB may result in loss of characters on the serial line. It is recommended to use a real cable for the serial line.

It has been discovered from bitstream v2.0 that grmon connection using Ethenet link is not stable. We suggest to launch grmon using the following command:

```
grmon -nl2c -altjtag
```

6.1.2.1 Setting Hardware Breakpoint

Using JTAG for debug or testing requires understanding well the way to do with several applications having same virtual space.

To set a hardware breakpoint you have to provide the virtual address where to install it. This is strait forward as it is provided by the elf file and the map file you get when you make install your application project. The application is statically linked with the virtual space provided in the linker script file used in the makefile of the application project.

The concern is that a given virtual address may correspond to several physical addresses when several partitions are running because they have the same virtual address space by default.

Thus when a hardware break point is reached, the hardware debug support has no mean to identify the current partition that is running.

To work around this generic issue, the idea is to map the partition under debug to a unique virtual space so that no other partition can overlap and the virtual HW BP correspond to only one virtual address so that debugger can identify it.

The building process of the PikeOS allows user to do so modifying the linker script of the application project.

6.1.3 Running FPGA board in PTA mode

FPGA board has been modified to supply random placement/replacement strategy in hardware called PTA mode. This feature is optional and shall be enabled by following settings:

- Set the SLIDE_SW0 board slide switch in position OFF for UART mode to get readable trace on the system console
- Set the SLIDE_SW2 board slide switch in position ON (see proxima_qsg_1.6.pdf section 4.2. Probabilistic Timing Analysis Features).
- By default, PikeOS won't enable the L2 Cache. You need to activate it setting the PSP_L2_ENABLE of PSP PROXIMA-FPGA TAGS to RND mode. If it is OFF, the L1PTA mode will be enforced to support L1 and BP.

6.1.4 Running FPGA board in non PTA mode

FPGA board can be set to default LRU placement/replacement strategy through the following settings:

- Set the SLIDE_SW0 board slide switch in position OFF for UART mode to get readable trace on the system console
- Set the SLIDE_SW2 board slide switch in position OFF (see proxima_qsg_1.6.pdf section 4.2. Probabilistic Timing Analysis Features).
- By default, PikeOS won't enable the L2 Cache. You need to activate it setting the PSP_L2_ENABLE of PSP PROXIMA-FPGA TAGS to LRU mode.

6.2 PROXIMA-COTS board

The following sections is a complement to the kernelref.pdf and the platppc-e500mc.pdf document when this PSP is project specific or added to the document when delivered as product line.

6.2.1 PSP PROXIMA-COTS TAGS

The following table lists the PSP tags and their default values for the leon3 PROXIMA-COTS PSP (derived from P4080 reference board) for both PikeOS 3.4 and 3.5:

Tag	Description	Default value
PSP_QUARTZ_FREQ	This tag specifies the core complex bus (CCB) clock frequency. It is modified if the FDT is used.	800000000 (800 MHz)
PSP_BAUDRATE	This tag specifies the baud rate for the console port. Possible values are 1200, 2400, 4800, 9600, 19200, 38400, 57600, and 115200.	115200
PSP_CONSOLE_PORT	This tag specifies the console output port. The value 0 disables the console output, a value of 1 or 2 cause the console output to be directed to the first or second UART channel. Any other value disables the console output.	1
PSP_MEMSIZE	This tag, if present, specifies the amount of memory to use for the system application.	0x80000000 (2 GBytes) (TBD)
PSP_PORT_CONFIG	This tag specifies the position of the CCSRBAR (configuration, control, and status registers base address register). This position is shifted left by 24 bits to get the physical address. This address is usually set by the bootloader and	0xffe (0x0000000ffe000000)

	the integrator needs to adopt this setting for the PSP to detect the CCSRBAR correctly. It is modified if the FDT is used.	
PSP_L2_ENABLE	This tag allows to control the L2 cache behaviour. The tag values are as follows: **Value** — **description** 0 — L2 disable 1 — L2 instruction only 2 — L2 data only 3 — L2 data + instruction	3 (data + instruction)
0x301	This tag controls the L3 cache-strategy behavior (if the L3 cache is available on the board). The tag values are as follows: **Value** — **description** 0 — disabled 1 — write-through 2 — copy-back **Note:** The tag is only supported on PowerPC e5500	2 (copy-back)
0x302	This tag controls the size of CPC (CoreNet Platform Cache) used as SRAM. The remaining size is used as cache. (Note that CPC may not be available on some QorIQ boards. **Note:** The tag is only supported on PowerPC e5500	0x100000
0x308	This tag specifies whether the kernel region is mapped to a cacheable (default 0) or not cacheable area (1). This region includes the BSP/kernel code and data area in their normal &	0x0

	tracesys configurations. The trace configuration is larger than the region and will be partly not cached when activated.	
0x309	This tag specifies the MBTA buffer size where to write events: - 0 when MBTA recording is not supported - N * P4PAGE_SIZE for a valid buffer. Will be allocated in the main memory with no cache attribute	0x400000 (4MB)

6.2.2 Application debug support

There are 2 ways for application debug within PROXIMA atop PikeOS;

The first one is the use of CODEO CDT debugger connected to the target through MUXA and the monitored application. When "make" an application with PikeOS, an unstripped version is also provided for the debug purpose you have to declare as application to the CODEO CDT. This is the standard way for source code level debug of algorithm, running either the real board or the QEMU simulator. A limitation exists as some guest OS does not provide the required monitoring; that is the case for RTEMS v4.10 and ELinOS but the second approach is usable in this case. With this approach, you have to load the unstripped application into the gdb host side so that it can provide source level debug.

The second way is to monitor application through a third part JTAG debug environment available for the COTS board. This way is probably the preferred and most efficient one when running timing analysis or debugging at low level.

6.3 Board independent Tool chain UM

6.3.1 PAK specific APIs

The project is required to provide a PAK resource to supply the test case identified by the PROXIMA work packages. The proxima-pak demo project implements the requested API and a simple application for validation. It has been extended with a high precision timer service for performance measurement and with a pseudo random generator to support software randomization at user level.

This PAK service is provided as source files in the proxima-pak demo project so that it can be adapted to the required configuration.

To use the PAK services just clone the provided:

- /opt/pikeos-3.4/demos-pikeos/proxima-pak as your own application project directory and adapt the src/archdep/main.c module to your need.
- /opt/pikeos-3.4/demos-integration/proxima-pak as your integration project directory

The main to start your implementation is look to the following:

- src/archdep/main.c; could be let as is. It make call of par_run() service that is expected to be the SCUA. Then this call is encapsulated in the necessary test harness support to do performance measurement that will be reported by the dump application.
- src/core/pak.c provides pak_run() service that is shall be replace by the code you want get performance measurement.
- src/dump/dump.c is for trace reporting. You may adapt it to format results reporting for integration with your tool chain if required.

6.3.1.1 Memory allocation API

PAKs specify a function prototype as a way to dynamically allocate memory.

To use this PAK services, please add the following line into your code:

```
#include "src/core/alloc.h"
```

To implement the service, a partition pool named “RAM_POOL” has been associated to each PAK. With the new function “*alloc(size)*”, contiguous memory is allocated from this memory pool and with the specified size.

The total allocated memory must be less than the partition pool size. Besides the mapping in the partition address space is statically set so it is mandatory to check if this address space is not used elsewhere.

To modify this address you have to modify the *src/archdep/alloc.c* file provided with the proxima-pak.app demo template; this may vary on per board basis and per personality:

```
        P4_address_t current_addr = 0x08220000;
```

6.3.1.2 Timer API

The PikeOS Kernel API already implements some functions related to the system timer (*p4_get_time*, *p4_sleep*, ...) . Nevertheless, we do not want to pay the overhead of the API call in this specific use case dedicated to measurement. Besides, the PikeOS Timeout API is based on the scheduling period which is less precise than the HW timer. This is why we implemented a timer which will bypass these system calls and directly access to the hardware registers.

To use the PAK services, please add the following line into your code:

```
#include "src/core/time.h"
```

The purpose of this API is to synchronise the PAK on each core with the minimum jitter. It implements the following services:

```
/**
 *  @purpose
 *    Get reference time for the PAK applications
 *
 *  @return
 *    Reference time in nanoseconds
 */
P4_time_t pak_get_ref(void);


/* Get current time since system reset in nanoseconds */
P4_time_t pak_get_time(clockInfo_t  *info);


/* Active wait until absolute time in nanoseconds */
void pak_sleep(clockInfo_t  *info, P4_time_t  time);
```

6.3.1.3 PRNG User level support

In order to provide the user level with a random seed value that differs on any partition startup, a specific service is also provided for debug purpose to read and to set the current seed of the pseudo random generator.

For now at M20 the software randomization can use the pak_get_random_value service adding the following line to the code:

```
#include "psp/prng.h"
```

The default random value at initialization time is defined in the `apbrandbank_prng_default` variable of the BSP so that it can be modified by an external srec or elf file at load time using same technique that described in the section 6.3.2.1.1 “PRNG support”.

After the board initialization the random generator use a seed value stored in the BSP area (PSP kinfo page) so that it is partition restart independent. That allows the function to be called from a partition providing a new random number on any occurrence, over partition restart.

6.3.1.3.1.1 How to get a PRNG that differs on each reset/reboot:

The available FPGA does not provide any means (battery backed up area…) to save the last current seed value as current on next reset or reboot. So on reset/reboot the current value is 0 and the first seed is provided from the default value.
In the frame of the POXIMA project we suggest the test harness to print the last known seed value using get_current_seed() so that it can be provided as input parameter to the next session. The starting seed value can be written as current seed by the test harness using set_current_seed() and then new test session will start with a PRNG that is not the default value. At deployment the set_current_seed() can be used to set the PRNG seed to random value from some hardware means that provides enough variability at startup like current temperature, or voltage, calendar time…

```
/**
 *  @purpose
*    Initialize the package:
 *    - provide access to the prop:/board/drv/apbrandbank_fp psp device
 *    - initialize current seed with p4_get_time()
 *
 */
void pak_prng_init(void);

 /**
 *  @purpose
 *     return a pseudo random value
 *
 *    if current seed is 0
 *      Set PRN to default value when current seed is {0,0}
 *      seed = current seed;
 *      set current seed to apbrandbank_prng_default
 *    if current seed is {1,1}
 *      Freeze PRN to default value when current seed is {1,1}
 *      seed = apbrandbank_prng_default;
 *    else
 *      Provides PRN
 *      seed = current seed;
 *      current seed =
 *      {
 *        .z = 36969*(.z&65535)+(.z>>16);
 *        .w = 18000*(.w&65535)+(.w>>16);
 *      }
 *    return (seed.z<<16)+(seed.w&65535);
 *
 *  @pre
 *    pak_prng_init
 */
P4_uint32_t pak_prng_get_random_number(void)
```

But in some case for debug purpose it has been requested to provide a mean to read and a mean to set the current value of the seed used to implement the pak_get_random_value() function:

```
/**
 *  @purpose
 *    pseudo random generator debug support:
```

```
 *    set the current seed to a given value
 *
 *   @param seed
 *    Pointer to input seed value
 *
 *   @pre
 *    pak_prng_init
 */
void pak_prng_set_seed(
/* IN  */ prng_vector_t *seed);

/**
 *   @purpose
 *    pseudo random generator debug support:
 *    return the current seed value
 *
 *   @param seed
 *    Pointer where to write seed vector
 *
 *   @pre
 *    pak_prng_init
 */
void pak_prng_get_seed(
/* OUT */ prng_vector_t *seed);


/**
 *   @purpose
 *    pseudo random generator debug support:
 *    return the default seed value
 *
 *   @param seed
 *    Pointer where to write seed vector
 *
 *   @pre
 *    pak_prng_init
 */
void pak_prng_get_default_seed(
/* OUT */ prng_vector_t *seed);

/**
 *   @purpose
 *    Bus Permutation configuration.
 *    Set the size of the arbitration window to allow a
 *    time-composable behaviour of bus arbitration on
 *    cores requests to the shared L2.
 *
 *   @param awc
 *    The Arbitration Window Cycles value in a range 0 to 255.
 *    Value 0 sets the system to deployment mode.
 *
 *   @pre
 *    pak_prng_init
 */
void pak_prng_set_BP2(/* IN */ P4_uint8_t awc);
```

6.3.1.3.1.2 How to populate HW with a given seed value from test code:

For some test purpose it has also been requested to provide a mean to set any HW seed to a given value that will not change over several time partition switches.
This is possible using a PSP patch provided as support when writing this document that changes the **pak_prng_get_random_number**() behavior to return the value of "apbrandbank_prng_default" without changing the current seed value
This behaviour is requested setting current PRNG seed to a specific pattern value {1,1} using the **pak_prng_set_seed**() service . When PRNG seed is set to this value a call from PSP to change any hardware seed will fill it with apbrandbank_prng_default value.
To set the "apbrandbank_prng_default" constant to a given value purpose, the test application needs to have Read/Write access rights to this constant that is allocated into the PSP. For test purpose it is acceptable to proceed this way:

```
/* you can get the apbrandbank_prng_default address by following
 * command in your integration project:
 *    fgrep "pbrandbank_prng_default" $PIKEOS_PSP_DIR/$PIKEOS_PSP/
 *                   objects/psp-kernel-smp-$MKROMIMAGE_VERBOSE.map
 */
prng_vector_t *apbrandbank_prng_default = (prng_vector_t *)
0x8003a260;  /* for normal image */
//0x80041330;  /* for trace image */
//0x8003b320;  /* for tracesys image */
P4_address_t PSP_DATA_PAGE =
    0x40000000
    | ( (P4_address_t)apbrandbank_prng_default & 0x0FFFF000);
/* Note1: set a PSP_DATA page memoryrequirement
 * of IO_MEM kind
 * with RDWR access rights
 * and physical address computed as above
 * the resulting address could be:
 *   0x4003a000;   for normal image
 *   0x40041000;   for trace image
 *   0x4003b000;   for tracesys image
 *   */

/* Allocate an application variable relatively to the
 * virtual address you shall set in a
 * PSP_DATA_PAGE ProcessMapEntry
 * for PSP_DATA page memoryrequirement
 * with RDWR access attribute
 * and virtual address set to 0x07F08000
 */
prng_vector_t *user_view_on_default_seed = (prng_vector_t *)
(0x07F08000 | ( (P4_uint32_t) apbrandbank_prng_default & 0xFFF) );

prng_vector_t SOME_SEED_TEST_VALUE = {2,3};

*user_view_on_default_seed = SOME_SEED_TEST_VALUE;

/* Now ask to load any HW seed with above default one */
prng_vector_t FROZEN_SEED_COMMAND = {1,1};
pak_prng_set_seed(FROZEN_SEED_COMMAND);
```

6.3.2 MBPTA support documentation

This section provides the design ideas and quick support/references the user could need to implement MBPTA compliant architecture atop PikeOS. This is first based on some optional PTA techniques to be supported and then interface with measurement based tooling.

6.3.2.1 Probabilistic Timing Analysis support

6.3.2.1.1 PRNG support

MBPTA requires availability of robust PRNG that is implemented in the bit library and in the proxima-pak demo project to be available for both the hardware and the software randomization technology.

To get it in the under the test application or to implement the software randomization, simply add the following include line in the source:

```
#include "bit/prng.h"
```

It will provide the services described at section 6.3.1.3 “PRNG User level support”.

The software pseudo random number generator is initialized using a seed variable “apbrandbank_prng_default” at kernel level into the BSP when the current seed is 0.

When the current seed is set to the pattern {1,1} PRNG value is frozen to “apbrandbank_prng_default” value.

```
static P4_uint32_t get_random_number(void)
{
   prng_vector_t *kprn = (prng_vector_t *) &pspkinfo->current_prng_seed;
   /* set current seed as default PRN */
   prng_vector_t prn = *kprn;
   if ( ( prn.w == 0 ) && ( prn.z == 0 ) )
   {
      /* fill current seed with default value as next PRN */
      kprn->z = apbrandbank_prng_default.z;
      kprn->w = apbrandbank_prng_default.w;

   } else if ( ( prn.w == 1 ) && ( prn.z == 1 ) ) {
      /* freeze PRN to default value, don’t change current seed */
      prn = apbrandbank_prng_default;

   } else {
      /* compute next PRN as current seed */
      kprn->z = 36969*(kprn->z&65535)+(kprn->z>>16);
      kprn->w = 18000*(kprn->w&65535)+(kprn->w>>16);
   }
   return (prn.z<<16)+(prn.w&65535);
};
```

This can be overwritten and read using the other services described at section 6.3.1.3 “PRNG User level support”.

6.3.2.1.2 Hardware support for PTA

When a board provides some hardware support for PTA this is part of the PSP implementation to be deployed with or without configuration option (see section 6.1.3).

6.3.2.1.3 BSP implementation

The FPGA optionally provides hardware support to PTA. The main PROXIMA contribution is to provide L1 and L2 cache with randomized placement/replacement algorithm instead of legacy LRU. Are concerned the L1 and L2 caches of each core and the bus permutation.

This new algorithm requires a pseudo random generator (PRNG) initialized with a seed so that it can be restarted at a given known state for a debug purpose. Then the OS has to properly set the hardware seeds in a known state at initialization time.

Note that immediately after changing a hardware seed value, the cache has to be flushed and invalidated so that there will be no placement/replacement error of already loaded cache lines.

The following algorithm is implemented by PikeOS to properly set the hardware seeds at initialization time:

- Set the hardware L1&L2 replacement & placement seeds when PSP_L2_ENABLE TAG is either RND or L1PTA and PTA mode switch is set (see 6.1.1):

```
SEED_ADDR_C1_L1L2_REPLACEMENT = get_random_number();
SEED_ADDR_C2_L1L2_REPLACEMENT = get_random_number();
SEED_ADDR_C3_L1L2_REPLACEMENT = get_random_number();
SEED_ADDR_C4_L1L2_REPLACEMENT = get_random_number();

SEED_ADDR_C1_L1L2_PLACEMENT = get_random_number();
SEED_ADDR_C2_L1L2_PLACEMENT = get_random_number();
SEED_ADDR_C3_L1L2_PLACEMENT = get_random_number();
SEED_ADDR_C4_L1L2_PLACEMENT = get_random_number();
```

- Then set the hardware bus permutation seeds PSP_L2_ENABLE TAG is either RND or L1PTA and PTA mode switch is set (see 6.1.1):

```
SEED_ADDR_BP1 = get_random_number();
SEED_ADDR_BP2 = get_random_number();
```

The `prng module` provides services to set values for debug purpose from user application.
Note that each time a seed is modified, the value is published into the corresponding pspkinfo page register:

- pspkinfo->L1L2_PLACEMENT[core]
- pspkinfo->L1L2_REPLACEMENT[core]
- pspkinfo->BP1
- pspkinfo->BP2

The seed values can be read by the test harness to know about the active configuration.

6.3.2.2 Software randomization

The goal of this component is to modify allocation of data/code in the physical memory at starting time of partition as described in the [D2.2.doc § 3.1 SW randomization support].
As a starting point it is suggested to use system extension similar to the pirshd provided by PikeOS that allows to start/stop partition and download new code for the partition; see documentation [usingpike.pdf §11 The PikeOS Remote Shell (pirsh)]. This design allows implementation that is both guest OS independent and board independent.
Then an exception handler can be added to supply the relocating method.
The algorithm can call PikeOS kernel services to map physical memory from statically allocated pool to expected virtual space.
This work can be started with QEMU since not dependent on any specific PROXIMA-FPGA feature.

6.4 Measurement support for Timing Analysis

From PikeOS point of view measurement based timing analysis has same requirements for both standard and probabilistic approach. The need is to provide tooling for the various measurement processes usually involved:

- Interface with the external third part measurement tooling to catch the expected time stamped events (IPOINTS for example),
- API (libbit) that can be used in any test case for
  - service characterization process at qualification time
  - in the Built In Test at maintenance mode
  - at startup time for sanity check.
- API may be available for any personality

To get it in the under the test application or to implement the software randomization, simply add the following include line in the source:

```
#include "bit/bit_perf.h"
```

It will provide the services described at section 6.4.6.2 “Performance measurement support”.

6.4.1 Rapita IPOINTS

The Rapita RVS tools need some event trace points called ipoints to be recorded on a dedicated memory location at some time.

This is not requested by the PikeOS Time Checker Tool but used for analysis of application that uses PikeOS.

The initial implementation will provide 2 callbacks:

1. At kernel level in the PikeOS PSP to generate ipoint at time partition switch
2. At partition level to generate ipoints from user level without the need to do intrusive syscall

The following ipoints are written at PSP level as a 32 bits word on the GPIO for the FPGA, or timestamped in a memory buffer the dump application will report to host:

```
/* purpose:
 *   Measurement Based Timing Analysis support.
 *
 *   The MBTA Event data format is as follows:
 *   - 32 bits : timestamp (unit depends on BSP),
 *     not recorded when used with external plug like RTBx
 *   - 3 bits  : a TAG identifying the event kind
 *   - 5 bits  : the core id the event occurred
 *   - 16 bits : depends on event kind
 *
 */

/* Interrupt arrival
 *
 * The interrupt ID (Hardware ID) is recorded at preamble level
 * for any interrupt kind arrival.
 *
 * EVENT data output format:
 * |       |       |       |       0
 *  000                               MBTA_TAG_INTERRUPT_ARRIVAL
 *     xxxxx                          cpu_id
 *          000000000000000000        unused
 *                            xxxxxx  it_id * */

/* PikeOS time partition switch
 *
 * The incoming time partition ID is recorded at time partition switch
 * when the dedicated PSP service is called.
 *
 * EVENT data output format:
 * |       |       |       |       0
 *  001                               MBTA_TAG_TP_SWITCH
 *     xxxxx                          cpu_id
 *          000000000000000000        unused
 *                            xxxxxx  timepart_id
 * */

/* PikeOS task switch, and thread switch since local to task
 *
 * The incoming task ID and thread ID is recorded at task switch
 * time.
 *
 * EVENT data output format:
 * |       |       |       |       0
 *  010                               MBTA_TAG_TASK_SWITCH
 *     xxxxx                          cpu_id
 *          0000                      unused
 *              xxxxxxxxxxx           task_id
 *                         xxxxxxxxx  thread_id
 * */

/* PikeOS thread switch
 *
 * The incoming thread ID is recorded at thread switch time.
 *
 * EVENT data output format:
 * |       |       |       |       0
 *  011                               MBTA_TAG_THREAD_SWITCH
 *     xxxxx                          cpu_id
 *          000000000000000           unused
 *                         xxxxxxxxx  thread_id
 * */

/* Guest OS process switch
 *
 * The incoming Guest OS specific process/pthread ID is recorded at
 * Guest OS switch time.
 *
 * EVENT data output format:
 * |       |       |       |       0
 *  100                               MBTA_TAG_GUEST_OS_PROC_SWITCH
 *     xxxxx                          cpu_id
 *          00000000                  unused
 *                  xxxxxxxxxxxxxxxx  guest OS process_id 0..255
 * */

/* MAF TimeStamp
 *
 * This event is only written by core #0 on Time partition switch
 * when to the first time slot of the scheduling schema that correspond
 * the main MAF.
 * It is used in the dump application to compute MAF relative time..
 *
 * EVENT data output format:
 * |       |       |       |       0
 *  101                               MBTA_TAG_MAF_TS
 *     xxxxxxxxxxxxxxxxxxxxxxxxxxxxx  bits 61..32 of last_maf_ns
 * */
```

Any user application can get control on recording mechanism to:

- restart recording
- Select events to be recoded
- Record a user specific event

This done using the libbit by including the following in the source code:

```
#include "bit/bit_mtba.h"
```

This provide the following services:

```
/* Initialize the buffer used for writing Events
 * Reset recording buffer,
 * any previously recorded event is lost.
 * No action in case of IO port
 * */
void mbta_reset_buffer (
/* IN */ PSP_kinfopage_t *pspkinfo);

/* Set filter to record event only when of active tag kind
 * @param tag
 *    the tag array of activity state to be set,
 *    tag is from MBTA_TAG_INTERRUPT_ARRIVAL to
 *    MBTA_TAG_USER_EVENT
 *    Any other value has no effect.
 * @param activities the new state for each tag as a bit array
 *    compacted into a P4_uint32_t.
 *    when TRUE any event of tag kind is recorded
 *    when FALSE any event of tag kind is not recorded
 * */
void mbta_tags_activity (
/* IN */ PSP_kinfopage_t *pspkinfo,
/* IN */ P4_uint32_t  activities
);

/* User specific event
 *
 * Intended to be written from user space
 *
 * EVENT data output format:
 * |       |       |       |       0
 *  111                              MBTA_TAG_USER_EVENT
 *     xxxxxxxxxxxxxxxxxxxxxxxxxxxxx user specific 0..2^29-1
 * */
void mbta_write_event_user (
/* IN */ PSP_kinfopage_t *pspkinfo,
/* IN */ P4_uint32_t user_event)
;
```

6.4.2 Running kernel in TC mode

PikeOS assumption to get best level of Time Composability is to run the PikeOS kernel into either a dedicated no cache attribute area or locked into the cache. The purpose of this is to reduce the impact on user cache history from system call.

This is achieved setting non zero value into the specific kernel TAG 0x308 (see above section 6.1.1 for FPGA and 6.2.1 for COTS P4080).

6.4.3 Using libsupc++

Despite it is highly not recommended to use libsupc++ inside critical application since it will raise certification concerns, we provide here the way to work around some undefined references (encountered at time of SW randomization implementation). The need was raised when the following 2 undefined symbols occurred. In this case, rather than providing a full libsupc++ that will generate unexpected side effects we suggest to include the following code in the relevant application:

```
/* Global (usually per library) DSO handle (unused here) */
void *__dso_handle = &__dso_handle;

typedef void (*cxa_atexit_callback)(void *);

/* list of destructors, called "atexit" */
struct __cxa_atexit_list_s {
    cxa_atexit_callback func;
    void                *arg;
    struct __cxa_atexit_list_s *next;
};

struct __cxa_atexit_list_s *__cxa_atexit_list = NULL;

/* Registers additional Destructors for execution "atexit()" */
int __cxa_atexit(cxa_atexit_callback func, void* arg,
                 __attribute__((unused))const void* dso)
{
    struct __cxa_atexit_list_s *new_entry;

    new_entry = malloc(sizeof(struct __cxa_atexit_list_s));
    if (new_entry == NULL)
        return -1;

    new_entry->next = __cxa_atexit_list;
    new_entry->func = func;
    new_entry->arg = arg;
    __cxa_atexit_list = new_entry;

    return 0;
}
```

It will be decided later in the project if this feature is to be added as requested for MBPTA support and in this case the way to get it available is still to be defined.

6.4.4 Using math library (libm) on PikeOS 3.4

6.4.4.1 Overview

The PikeOS native personality extension (P4ext) provides the C and C++ programming environment but this extension is not included in PikeOS 3.4. The libm and libc libraries from the C programming environment have been included in the project BSP v1.2.4 for both SPARC V8 and PPC E500MC targets.

The GNU C library has been included to resolve libm dependencies but this library depends itself on P4ext and has not been fully ported to PikeOS 3.4. **This is why some of the functions of the libc (e.g. memory allocation functions) won’t be functional; with PikeOS the usable set of function is the one provided in libstand.**

The C programming environment documentation from PikeOS 3.5 is available at $PIKEOS_PREFIX/documentation/cprogenv.pdf.

6.4.4.2 Use libm library

To use libm in your project, select "C programming environment" in the PikeOS Project Configurator and check "enable math library". Besides it is also recommended to include the libc by checking "enable GNU C library" in order to resolve some dependencies from libm. Finally, add the following include line in the source:

```
#include <math.h>
```

More details can be found in *cprogenv.pdf* section "Library Summary: Math Library".

6.4.4.3 Redefine errno variable

Alternatively, you can use your own libc or resolve yourself some of the dependencies. Most of the functions of libm use thread local error handling so you need to redefine the errno variable as follow:

```
int errno;

int *__error(void)
{
return &errno;
}
```

6.4.5 Running scua with software randomization

In case there is no hardware support for randomization any application can be loaded using software randomization emulation.

A PikeOS privileged application is used to load the scua into the memory. This PikeOS privileged partition can be modified to supply SW-Randomization support.

Boot the ROM Image (see platform manual for more details about the boot configuration). You should see this in the console:

```
core 0: init PAK...
core 2: init PAK...
core 3: init PAK...
core 0: exec time = 23093990ns
core 2: exec time = 23315640ns
core 3: exec time = 23098800ns
[…]
```

The core1 partition is not started at boot time because it needs a binary (the SCUA) which should be copied into the shared memory using PIRSH modified for SW-Randomization.

Open PIRSH and send the following commands to load the SCUA and start the partition:

```
PIRSH> set localfile $SCUA_PATH
PIRSH> set targetfile shm:scua
PIRSH> put
PIRSH> start 2
```

6.4.6 Built In Test and Time Composability checker tool support

From BSP v1.3.0 rather than extending PAK that is PROXIMA project specific and is upgraded in both specification and implementation by other contributors, the BSP now provides a new Built In Test library used by the Time Checker tool.
In this BSP delivery as a first lesson using the performance counters along with PAK Timer API, it has been decided to provide a simplified API to support performance measurement.
As it is provided as a library including low level hardware abstraction this library can also be mapped into a no cache area increasing measurement precision.
Thanks to the hardware abstraction this simplified API is also architecture independent and personality independent so that the test code can be hugely simplified and board independent.
Note that this library provides also services to extend with specific performance counters and adapted output format

The library provides services for both the ibit support as test master / server and then the scua and iload applications as slave/client.

6.4.6.1 Test automaton library support

The bit library provides a set of services that helps implementing the ibit application providing the required services for easy configuration and control of both scua and iload.

Whatever the Guest OS is for the scua, there is only one ibit implementation for any.

To get this library support available, simply add the following include line in the source:

```
#include "bit/bit_test.h"
```

And don’t forget to have the libbit available at link time. The demo project implementing the Time Checker tool shows how to use it.

The communication between automaton as server and clients (scua and iloads) make usage of a sampling port.

The bit lib provides the following API for test automaton implementation:

```
/**
 *  @purpose
 *    Initialize the Built In Test test library for client side
 *
 *  @param is_SCUA
 *    true when the client side is the SCUA, false when caller is an
ILOAD instance.
 *
 *  @pre
 *    vm_init
 */
extern void bit_test_init(P4_bool_t is_SCUA);

/**
 *  @purpose
 *    Wait for the next time slot occurrence in the next MAF
 *
 *  @pre
 *    bit_test_init.
 *
 */
extern void bit_test_wait_next_period();


/**
 *  @purpose
 *    Return the command line the client application
 *    shall perform for this step on the current core.
 *    Selection in between scua or iload depends on the
 *    core mask allocated by bit_test_set_next_action().
 *    The action also includes tsync start time that will
 *    be used by further bit_test_wait_tsync() call.
 *    Return a default no action command line when:
 *    - action message not received or not valid
 *    - action message is not valid
 *    - we are scua and no action assigned for scua
 *    - we are iload and no action assigned for this iload
 *    Return a P4_uint64_t * when the action is an array of args.
 *    Distinguishing can be based on arg[0] value corresponding
 *    to the command/test case :
 *    >= 0x2020202020202020 as a string
 *    <  0x2020202020202020 as the corresponding enum value
 *
 *  @param cmd_line
 *    the command line to be applied on the client side.
 *
 *  @pre
 *    bit_test_set_next_action()
 */
extern P4_uint64_t *bit_test_get_action();


/**
 *  @purpose
 *    Return the Tsync start time value for the client that
 *    corresponds to the following variable in the scenario:
 *    - SCUA.Tsync for the SCUA
 *    - Tsync for any iload
 *    - not applicable to server
 *    Allows synchronization of actions to be performed
 *    on all the cores.
 *
 *  @pre
 *    bit_test_get_action()
 */
extern P4_time_t bit_test_get_tsync();

/**
 *  @purpose
 *    Do busy wait until given time.
 *
 *  @pre
 *    bit_test_get_action()
 */
extern void bit_test_busy_wait(P4_time_t until);

/**
```

```
 *  @purpose
 *    scua to tell server action is completed or on going
 *
 *  @param ack_msg
 *    NUL terminated string to report to server.
 *    ASCII ACK on completion
 *
 *  @pre
 *    bit_test_set_next_action()
 */
extern void bit_test_action_report(char *ack_msg);
```

The automaton will work as the following simplified sequence on each MAF (see Figure 1Time Composability checker tool time frame):

Core 0	**Core I**
T(n) : MAF - period start	T(n) : MAF - period start
ibit: - bit_perf_measurement_conf() - bit_test_wait_next_period() - bit_test_broadcast_action () o will wait for scua action completion - bit_perf_measurement_print() - bit_perf_dump_events()	No partition
IO partition	IO partition
Scua: - bit_test_get_action() - *K: prepare for api_under_test* - bit_perf_measurement_conf() - bit_test_wait_tsync() - bit_perf_measurement_start() - api_under_test() - bit_perf_measurement_stop() - bit_test_action_report () - bit_test_wait_next_period()	Iload(I): - bit_test_get_iload_action() - *K: prepare for iload action* - bit_test_wait_tsync() - iload() - *iload goes on here* - *iload shall end here* - bit_test_wait_next_period()
IO partition	IO partition

6.4.6.2 Performance measurement support

The bit library provides a set of services simplifying the measurement implementation in the scua. This simplified API will hide the hardware dependency details through an internal hardware abstraction level.

To get it in the under test application, simply add the following include line in the source:

```
#include "bit/bit_perf.h"
```

And don't forget to have the libbit available at link time (-lbit linker option). The demo project implementing the Time Checker tool shows how to use it.

The bit lib provides the following API for performance measurement of the scua:

```
/**
 *  @purpose
 *    Initialize the Built In Test performance library.
 *    Will be called from application where software under
 *    test is exercised, on the main core.
 *    1) Allocates a circular buffer into "BIT_POOL"
 *    2) compute the max number of events that can be recorded
 *     in the circular buffer and display it to user.
 *
 *    Size for one measurement is 256 bytes.
 *
 *  @pre
 *    bit_perf_init()
 */
extern void bit_perf_measurement_init(void);


/**
 *  @purpose
 *    Set measurement configuration for performance counters.
 *    This allows to do measurement with various sets of counters,
 *    thinking numbers of counters activated simultaneously is
 *    ARCH dependent, example:
 *    - 16 for LEON3 PROXIMA
 *    - 4 for e500MC
 *    After changing configuration the service does:
 *    1) calibration measurement computing overheads
 *    2) advance pointer to first free measure in buffer
 *
 *    The DEFAULT first measurement configuration is the 16 counters
 *    defining the 16 printed columns. Others shall be a subset of
this
 *    DEFAULT so that any perf counter of the subset is always
printed
 *    in the same column position to ease post analysis.
 *
 *    In case of invalid or unregistered config name, the
 *    service will print an error message and select the
 *    default configuration.
 *
 *  @pre
 *    bit_perf_measurement_conf_register()
 */
extern P4_e_t bit_perf_measurement_conf();


/**
 *  @purpose
 *    start a measurement
 *    The start time is recorded and performance counters cleared
 *
 *  @pre
 *    bit_perf_measurement_init()
 */
extern void bit_perf_measurement_start();

/**
 *  @purpose
 *    stop a measurement and record the values
 *    The stop time is recorded
 *    Performance counters are recorded.
 *
 *  @pre
 *    bit_perf_measurement_start()
 */
extern void bit_perf_measurement_stop();

/**
 *  @purpose
 *    Add scua return code value to the last recorded measurement
 *
 *  @param scua_return_code
 *    the return code to report.
 *
 *  @pre
 *    bit_perf_measurement_stop()
 */
extern void bit_perf_measurement_rc(P4_uint32_t scua_return_code);
```

The typical usage for an API measurement / characterization is as following

```
#include <bit/bit_perf.h>


bit_test_init();
/* we are now starting a new MAF */

bit_perf_measurement_init();

/* notify TCC the scua is ready,
 * sending the name as notification */
Scua_Name = "vm34";
bit_test_action_report(Scua_Name);



/* set cache history context to worst case at init time to reduce
 * time overhead on very first test */
any_client_set_caches_fully_dirty();

bit_perf_measurement_conf();

for (i = 0; i < NB_ITERATIONS; ++i) {
    bit_perf_measurement_start();
    rc = do_it() /* Do stuff here */
    bit_perf_measurement_stop();
    /* report the API return code if any */
    bit_perf_measurement_rc(rc);
}

bit_test_action_report (Ack_String);
```

The configuration of all the API can be easily managed when starting your integration cloning the provided demo proxima-pak application and integration project.

6.4.6.2.1 Calibration of Execution Time Measurement

In order to provide a good quality and valid Execution Time measurement of a given API from inside a test software, it is required to avoid some pitfalls taking into account the interference introduced by the measurement tool itself:

1. Measuring the execution time of an API shall be reduced with overhead introduced by the measurement API it self
2. Reading the performance counters also introduces overhead in the ET of the API.
3. The performance counters value is also modified by both the ET measurement API and the reading of counters itself.

Following is the default `Perf_Configuration` of FPGA performance counters implementation in the test server. This correspond to the TC checker tool need to characterize APIs.

To reduce interferences of BIT library services a calibration step has been introduced at initialization time of the library. Then it reads and internally records the overheads for both time and counters. Those overheads are taken into account to compute the results for an API call measurement.

The print service provides the overheads values as first line for information and the other lines provides the values after overhead correction.

6.4.6.2.2 Performance counters configuration.

Implementation of the library is based on hardware abstraction layer to provide the required resource reducing hardware dependency.

The performance counters are specific to each hardware and so cannot be hidden. But the setting of this configuration might be set on the core the measurement is to be done (for example, on P4080 any PMC is local to core registers set). So the computation of initialization values is done on the test master/server side, but configuration values are stored in the command message broadcasted to the scua. Then scua can apply the configuration before running the test.

The specific service “**`bit_perf_measurement_conf`** provides the way for setting performance counter configurations in time.

On the server side the default configuration is the list of 16 performance counters you want to use. If more is needed, the server implementation need to be modified.

6.4.6.2.3 FPGA L3STAT implementation

The following configuration for counters is used for test server:

```
BIT_PERF_CONF_TABLE_TYPE g_bit_perf_conf_table = {
    .conf_nb = 6,
    .conf_table = {
        {
            .conf_name = "DEFAULT",
    .counters_nb = 15,
    .counters = {
    PERF_COUNTER ("  cycles",   E_EXECUTION_TIME_EXCLUDING_DEBUG_MODE,
DEFAULT_VALUE_L3STAT_REGISTER)
    PERF_COUNTER ("  Icount", E_TOTAL_INSTRUCTION_COUNT,
DEFAULT_VALUE_L3STAT_REGISTER)
    PERF_COUNTER (" IL1miss",   E_INSTRUCTION_CACHE_MISS,
DEFAULT_VALUE_L3STAT_REGISTER)
    PERF_COUNTER ("ITlbmiss", E_INSTRUCTION_MMU_TLB_MISS,
DEFAULT_VALUE_L3STAT_REGISTER)
    PERF_COUNTER ("  Dcount", E_LOAD_AND_STORE_INTSTRUCTIONS,
DEFAULT_VALUE_L3STAT_REGISTER)
    PERF_COUNTER (" DL1miss",   E_DATA_CACHE_MISS,
DEFAULT_VALUE_L3STAT_REGISTER)
    PERF_COUNTER ("DTlbmiss", E_DATA_MMU_TLB_MISS,
DEFAULT_VALUE_L3STAT_REGISTER)
    PERF_COUNTER (" L2Count", E_L2_CACHE_ACCESS,
DEFAULT_VALUE_L3STAT_REGISTER)
    PERF_COUNTER ("   L2Hit",   E_L2_CACHE_HIT,
DEFAULT_VALUE_L3STAT_REGISTER)
    PERF_COUNTER ("AHBMastr", E_AHB_UTILIZATION_PER_AHB_MASTER,
DEFAULT_VALUE_L3STAT_REGISTER)
    PERF_COUNTER ("AHBCount", E_AHB_UTILIZATION_TOTAL,
DEFAULT_VALUE_L3STAT_REGISTER)
    PERF_COUNTER (" AHBIdle",   E_AHB_IDLE_CYCLES,
DEFAULT_VALUE_L3STAT_REGISTER)
    PERF_COUNTER (" AHBBusy", E_AHB_BUSY_CYCLES,
DEFAULT_VALUE_L3STAT_REGISTER)
    PERF_COUNTER ("IntCount", E_INTEGER_INSTRUCTIONS,
DEFAULT_VALUE_L3STAT_REGISTER)
    LAST_COUNTER ("FPUCount", E_FPU_INSTRUCTION_COUNT,
DEFAULT_VALUE_L3STAT_REGISTER)
    }
        },
        {
            .conf_name = "HWMET", /* High Water Mark Execution Time */
    .counters_nb = 4,
    .counters = {
    PERF_COUNTER ("  cycles",   E_EXECUTION_TIME_EXCLUDING_DEBUG_MODE,
DEFAULT_VALUE_L3STAT_REGISTER)
    PERF_COUNTER ("  Icount", E_TOTAL_INSTRUCTION_COUNT,
DEFAULT_VALUE_L3STAT_REGISTER)
    PERF_COUNTER ("  Dcount", E_LOAD_AND_STORE_INTSTRUCTIONS,
DEFAULT_VALUE_L3STAT_REGISTER)
    LAST_COUNTER ("AHBMastr", E_AHB_UTILIZATION_PER_AHB_MASTER,
DEFAULT_VALUE_L3STAT_REGISTER)
    }
        },
    {   .conf_name = "Instr", /* Instruction relative */
    .counters_nb = 4,
    .counters = {
    PERF_COUNTER ("  cycles",  E_EXECUTION_TIME_EXCLUDING_DEBUG_MODE,
DEFAULT_VALUE_L3STAT_REGISTER)
```

```
    PERF_COUNTER ("  Icount", E_TOTAL_INSTRUCTION_COUNT,
DEFAULT_VALUE_L3STAT_REGISTER)
    PERF_COUNTER (" IL1miss",   E_INSTRUCTION_CACHE_MISS,
DEFAULT_VALUE_L3STAT_REGISTER)
    LAST_COUNTER ("ITlbmiss", E_INSTRUCTION_MMU_TLB_MISS,
DEFAULT_VALUE_L3STAT_REGISTER)
    }
    },
    {   .conf_name = "Data", /* Data relative */
    .counters_nb = 4,
    .counters = {
    PERF_COUNTER ("  cycles",   E_EXECUTION_TIME_EXCLUDING_DEBUG_MODE,
DEFAULT_VALUE_L3STAT_REGISTER)
    PERF_COUNTER ("  Dcount", E_LOAD_AND_STORE_INTSTRUCTIONS,
DEFAULT_VALUE_L3STAT_REGISTER)
    PERF_COUNTER (" DL1miss",   E_DATA_CACHE_MISS,
DEFAULT_VALUE_L3STAT_REGISTER)
    LAST_COUNTER ("DTlbmiss", E_DATA_MMU_TLB_MISS,
DEFAULT_VALUE_L3STAT_REGISTER)
    }
    },
    {   .conf_name = "BUS", /* AHB relative */
    .counters_nb = 4,
    .counters = {
    PERF_COUNTER ("  cycles",   E_EXECUTION_TIME_EXCLUDING_DEBUG_MODE,
DEFAULT_VALUE_L3STAT_REGISTER)
    PERF_COUNTER ("AHBCount", E_AHB_UTILIZATION_TOTAL,
DEFAULT_VALUE_L3STAT_REGISTER)
    PERF_COUNTER (" AHBIdle",   E_AHB_IDLE_CYCLES,
DEFAULT_VALUE_L3STAT_REGISTER)
    LAST_COUNTER (" AHBBusy", E_AHB_BUSY_CYCLES,
DEFAULT_VALUE_L3STAT_REGISTER)
    }
    },
    {   .conf_name = "L2", /* L2 cache relative */
    .counters_nb = 4,
    .counters = {
    PERF_COUNTER ("  cycles",  E_EXECUTION_TIME_EXCLUDING_DEBUG_MODE,
DEFAULT_VALUE_L3STAT_REGISTER)
    PERF_COUNTER ("  Dcount", E_LOAD_AND_STORE_INTSTRUCTIONS,
DEFAULT_VALUE_L3STAT_REGISTER)
    PERF_COUNTER (" L2Count", E_L2_CACHE_ACCESS,
DEFAULT_VALUE_L3STAT_REGISTER)
    PERF_COUNTER ("   L2Hit",   E_L2_CACHE_HIT,
DEFAULT_VALUE_L3STAT_REGISTER)
    }
    },
    },
};
```

The cpustat abstraction layer implementation uses the following services in the BSP v1.4.0 provided by l3stat module and defined in the follwing C header file:

```
#include "../archdep/l3stat.h"
```

```
/* -------------------- CONSTANT / MACRO DEFINITIONS ----------------
```

```
-------- */

#define PSPPERF_DRIVER_ID 17
/* 32 according to the documentation.
* But only 4 are implemented in FPGA at the moment */
#define PSPPERF_MAX_COUNTERS 4
#define NB_L3STAT_CORES 4

#define L3STAT_NCPU_SHIFT 28
#define L3STAT_NCNT_SHIFT 23
#define L3STAT_MC_SHIFT 22
#define L3STAT_IA_SHIFT 21
#define L3STAT_DS_SHIFT 20
#define L3STAT_EE_SHIFT 19
#define L3STAT_R_SHIFT 18
#define L3STAT_EL_SHIFT 17
#define L3STAT_CD_SHIFT 16
#define L3STAT_SU_SHIFT 14
#define L3STAT_CL_SHIFT 13
#define L3STAT_EN_SHIFT 12
#define L3STAT_EVENTID_SHIFT 4
#define L3STAT_CPU_AHBM_SHIFT 0

#define DEFAULT_VALUE_L3STAT_REGISTER ( \
3 << L3STAT_NCPU_SHIFT \
| 1 << L3STAT_IA_SHIFT \
| 1 << L3STAT_DS_SHIFT \
| 1 << L3STAT_EE_SHIFT \
| 1 << L3STAT_CL_SHIFT \
| 1 << L3STAT_EN_SHIFT )


typedef enum { /* see table 122 on um 1.7 */
E_INSTRUCTION_CACHE_MISS = 0x00,
E_INSTRUCTION_MMU_TLB_MISS = 0x01,
E_INSTRUCTION_CACHE_HOLD = 0x02,
E_INSTRUCTION_MMU_HOLD = 0x03,
E_DATA_CACHE_MISS = 0x08,
E_DATA_MMU_TLB_MISS = 0x09,
E_DATA_CACHE_HOLD = 0x0A,
E_DATA_MMU_HOLD = 0x0B,
E_DATA_WRITE_BUFFER_HOLD = 0x10,
E_TOTAL_INSTRUCTION_COUNT = 0x11,
E_INTEGER_INSTRUCTIONS = 0x12,
E_FPU_INSTRUCTION_COUNT = 0x13,
E_BRANCH_PREDICTION_MISS = 0x14,
E_EXECUTION_TIME_EXCLUDING_DEBUG_MODE = 0x15,
E_AHB_UTILIZATION_PER_AHB_MASTER = 0x17,
E_AHB_UTILIZATION_TOTAL = 0x18,
E_INTEGER_BRANCHES = 0x22,
E_CALL_INSTRUCTIONS = 0x28,
E_REGULAR_TYPE_2_INSTRUCTIONS = 0x30,
E_LOAD_AND_STORE_INTSTRUCTIONS = 0x38,
E_LOAD_INTSTRUCTIONS = 0x39,
E_STORE_INTSTRUCTIONS = 0x3A,
E_AHB_IDLE_CYCLES = 0x40,
E_AHB_BUSY_CYCLES = 0x41,
E_AHB_NON_SEQUENTIAL_TRANSFERS = 0x42,
E_AHB_SEQUENTIAL_TRANSFERS = 0x43,
E_AHB_READ_ACCESSES = 0x44,
E_AHB_WRITE_ACCESSES = 0x45,
E_AHB_BYTE_ACCESSES = 0x46,
E_AHB_HALF_WORD_ACCESSES = 0x47,
E_AHB_WORD_ACCESSES = 0x48,
E_AHB_DOUBLE_WORD_ACCESSES = 0x49,
E_AHB_DOUBLE_QUAD_WORD_ACCESSES = 0x4A,
E_AHB_DOUBLE_EIGHT_WORD_ACCESSES = 0x4B,
E_AHB_DOUBLE_WAITSTATES = 0x4C,
E_AHB_RETRY_RESPONSES = 0x4D,
E_AHB_SPLIT_RESPONSES = 0x4E,
E_AHB_SPLIT_DELAY = 0x4F,
E_AHB_BUS_LOCKED = 0x50,
E_L2_CACHE_HIT = 0x60,
E_L2_CACHE_MISS = 0x61,
E_L2_CACHE_ACCESS = 0x62
} enum_event;

typedef struct {
    enum_event Event_ID;
    P4_bool_t bEnable_Counter;
    P4_bool_t bClear_Read;
    P4_uint32_t Cpu_Number;
} sL3stat_ConfReg;

#define FUNC_GET_STAT_COUNTER 0x1
#define FUNC_WRITE_STAT_COUNTER_CONTROL_REGISTER 0x2
#define NB_MAX_L3STAT_COUNTER 32
```

1.1.1.1.1 P4080 implementation

The following configuration for counters is used for test server on P4080 board:

```
BIT_PERF_CONF_TABLE_TYPE g_bit_perf_conf_table = {
    .conf_nb = 2,
    .conf_table = {
        {
            .conf_name = "DEFAULT",
            .counters_nb = 16,
            .counters = {
PERF_COUNTER ("  cycles",   PERF_COUNT_HW_CPU_CYCLES,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
PERF_COUNTER ("  Icount", PERF_COUNT_HW_COMPLETED_INSNS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
PERF_COUNTER ("IL1rload",   PERF_COUNT_HW_IL1_FETCH_RELOADS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
PERF_COUNTER ("ITlbmiss", PERF_COUNT_HW_IMMU_TLB4K_RELOADS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
PERF_COUNTER ("  Dcount", PERF_COUNT_HW_TOTAL_TRANSLATED,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
PERF_COUNTER ("DL1rload",   PERF_COUNT_HW_DL1_RELOADS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
PERF_COUNTER ("DTlbmiss", PERF_COUNT_HW_DMMU_TLB4K_RELOADS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
PERF_COUNTER (" L2Count", PERF_COUNT_HW_L2_ACCESS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
PERF_COUNTER ("   L2Hit",   PERF_COUNT_HW_L2_HIT_ACCESS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
PERF_COUNTER ("DL2Count", PERF_COUNT_HW_L2_DATA_ACCESS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
PERF_COUNTER ("  DL2Hit",   PERF_COUNT_HW_L2_HIT_DATA_ACCESS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
PERF_COUNTER ("IL2Count", PERF_COUNT_HW_L2_INST_ACCESS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
PERF_COUNTER ("  IL2Hit",   PERF_COUNT_HW_L2_HIT_INST_ACCESS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
PERF_COUNTER ("IBIUCoun", PERF_COUNT_HW_BIU_MASTER_D_REQUESTS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
PERF_COUNTER ("DBIUCoun", PERF_COUNT_HW_BIU_MASTER_I_REQUESTS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
LAST_COUNTER ("BIUCount", PERF_COUNT_HW_BIU_MASTER_REQUESTS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
            }
        },
        {
            .conf_name = "HWMET", /* High Water Mark Execution Time */
    .counters_nb = 4,
    .counters = {
            PERF_COUNTER ("  cycles",   PERF_COUNT_HW_CPU_CYCLES,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
            PERF_COUNTER ("  Icount", PERF_COUNT_HW_COMPLETED_INSNS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
PERF_COUNTER ("  Dcount", PERF_COUNT_HW_TOTAL_TRANSLATED,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
    LAST_COUNTER ("BIUCount", PERF_COUNT_HW_BIU_MASTER_REQUESTS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
    }
        },
    {   .conf_name = "Instr", /* Instruction relative */
    .counters_nb = 4,
    .counters = {
        PERF_COUNTER ("  cycles",   PERF_COUNT_HW_CPU_CYCLES,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
        PERF_COUNTER ("  Icount", PERF_COUNT_HW_COMPLETED_INSNS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
PERF_COUNTER ("IL1rload",   PERF_COUNT_HW_IL1_FETCH_RELOADS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
LAST_COUNTER ("ITlbmiss", PERF_COUNT_HW_IMMU_TLB4K_RELOADS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
    }
    },
    {   .conf_name = "Data", /* Data relative */
    .counters_nb = 4,
    .counters = {
PERF_COUNTER ("  cycles",   PERF_COUNT_HW_CPU_CYCLES,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
PERF_COUNTER ("  Dcount", PERF_COUNT_HW_TOTAL_TRANSLATED,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
PERF_COUNTER ("DL1rload",   PERF_COUNT_HW_DL1_RELOADS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
LAST_COUNTER ("DTlbmiss", PERF_COUNT_HW_DMMU_TLB4K_RELOADS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
    }
    },
    {   .conf_name = "BUS", /* BIU relative */
    .counters_nb = 4,
    .counters = {
```

```
    PERF_COUNTER ("  cycles",   PERF_COUNT_HW_CPU_CYCLES,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
PERF_COUNTER ("IBIUCoun", PERF_COUNT_HW_BIU_MASTER_D_REQUESTS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
PERF_COUNTER ("DBIUCoun", PERF_COUNT_HW_BIU_MASTER_I_REQUESTS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
LAST_COUNTER ("BIUCount", PERF_COUNT_HW_BIU_MASTER_REQUESTS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
    }
    },
    {   .conf_name = "DL2", /* Data L2 cache relative */
    .counters_nb = 4,
    .counters = {
PERF_COUNTER ("  cycles",   PERF_COUNT_HW_CPU_CYCLES,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
PERF_COUNTER ("  Dcount", PERF_COUNT_HW_TOTAL_TRANSLATED,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
PERF_COUNTER ("DL2Count", PERF_COUNT_HW_L2_DATA_ACCESS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
LAST_COUNTER ("  DL2Hit",   PERF_COUNT_HW_L2_HIT_DATA_ACCESS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
    }
    },
    {   .conf_name = "IL2", /* Instruction L2 cache relative */
    .counters_nb = 4,
    .counters = {
PERF_COUNTER ("  cycles",   PERF_COUNT_HW_CPU_CYCLES,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
        PERF_COUNTER ("  Icount", PERF_COUNT_HW_COMPLETED_INSNS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
PERF_COUNTER ("IL2Count", PERF_COUNT_HW_L2_INST_ACCESS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
LAST_COUNTER ("  IL2Hit",   PERF_COUNT_HW_L2_HIT_INST_ACCESS,
PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER)
    }
    },
    },
};
```

The bit library cpustat abstraction layer implementation uses the following services in the BSP v1.4.0 provided by pspperf module and defined in the following C header file:

```
#include <psp/pspperf.h>
```

```

#define PSPPERF_DRIVER_ID (17)
#define PSPPERF_FUNC_GET_STAT_COUNTERS (0x1)
#define PSPPERF_FUNC_START_STAT_COUNTERS (0x2)
#define PSPPERF_FUNC_STOP_STAT_COUNTERS (0x3)
#define PSPPERF_FUNC_RESET_STAT_COUNTERS (0x4)
#define PSPPERF_MAX_COUNTERS (4)
#define PSPPERF_DEFAULT_VALUE_PMLCA_REGISTER (PMLCA_CE | PMLCA_FCM1)

/* Freescale Book E Performance Monitor APU Registers */
#define PMRN_PMC0 0x010 /* Performance Monitor Counter 0 */
#define PMRN_PMC1 0x011 /* Performance Monitor Counter 1 */
```

```
#define PMRN_PMC2 0x012 /* Performance Monitor Counter 2 */
#define PMRN_PMC3 0x013 /* Performance Monitor Counter 3 */
#define PMRN_PMLCA0 0x090 /* PM Local Control A0 */
#define PMRN_PMLCA1 0x091 /* PM Local Control A1 */
#define PMRN_PMLCA2 0x092 /* PM Local Control A2 */
#define PMRN_PMLCA3 0x093 /* PM Local Control A3 */
#define PMRN_PMGC0 0x190 /* PM Global Control 0 */

#define PMLCA_FC 0x80000000 /* Freeze Counter */
#define PMLCA_FCS 0x40000000 /* Freeze in Supervisor */
#define PMLCA_FCU 0x20000000 /* Freeze in User */
#define PMLCA_FCM1 0x10000000 /* Freeze when PMM==1 */
#define PMLCA_FCM0 0x08000000 /* Freeze when PMM==0 */
#define PMLCA_CE 0x04000000 /* Condition Enable */
#define PMLCA_FGCS1 0x00000002 /* Freeze in guest state */
#define PMLCA_FGCS0 0x00000001 /* Freeze in hypervisor state */

#define PMLCA_EVENT_MASK 0x01ff0000 /* Event field */
#define PMLCA_EVENT_SHIFT 16

#define PMGC0_FAC 0x80000000 /* Freeze all Counters */
#define PMGC0_PMIE 0x40000000 /* Interrupt Enable */
#define PMGC0_FCECE 0x20000000 /* Freeze countes on
Enabled Condition or
Event */

/*
* @see "e500mc Core Reference Manual, Rev. 3", Table 9-47.
* Performance Monitor Event Selection
*/
typedef enum {
/*
* General Events
*/
/* Every processor cycle */
PERF_COUNT_HW_CPU_CYCLES = 1,
/* Completed Instructions (0, 1, or 2 per cycle) */
PERF_COUNT_HW_COMPLETED_INSNS = 2,
/* Completed Micro-ops (counts 2 for load/store w/update) */
PERF_COUNT_HW_COMPLETED_OPS = 3,
/* Instruction fetches */
PERF_COUNT_HW_INSTRUCTION_FETCHES = 4,
/* Micro-ops decoded */
PERF_COUNT_HW_DECODED_OPS = 5,
/* 0 to 1 transitions on the pm_event input */
PERF_COUNT_HW_PM_EVENT_TRANSITION = 6,
/* Processor cycles that occur when the pm_event input is asserted */
PERF_COUNT_HW_PM_EVENT_CYCLES = 7,
/*
* Instruction Types Completed
*/
/* Branch Instructions completed */
PERF_COUNT_HW_COMPLETED_BRANCHES = 8,
/* Load micro-ops completed */
PERF_COUNT_HW_COMPLETED_LOAD_OPS = 9,
/* Store micro-ops completed */
PERF_COUNT_HW_COMPLETED_STORE_OPS = 10,
/* Number of completion buffer redirects */
PERF_COUNT_HW_COMPLETION_REDIRECTS = 11,
/*
* Branch Prediction and Execution Events
*/
/* Branches finished */
PERF_COUNT_HW_BRANCHES_FINISHED = 12,
/* Taken branches finished */
PERF_COUNT_HW_TAKEN_BRANCHES_FINISHED = 13,
/* Biffed branches finished */
PERF_COUNT_HW_BIFFED_BRANCHES_FINISHED = 14,
/* Branch instructions mispredicted due to direction, target, or IAB
* prediction */
PERF_COUNT_HW_BRANCHES_MISPREDICTED = 15,
/* Branches mispredicted due to direction prediction */
PERF_COUNT_HW_BRANCHES_MISPREDICTED_DIRECTION = 16,
/* Branches that hit in the BTB, or missed but are not taken */
PERF_COUNT_HW_BTB_HITS = 17,
/*
' * Pipeline STalls
*/
/* Cycles the instruction buffer was not empty, but 0 instructions
* decoded */
PERF_COUNT_HW_DECODE_STALLED = 18,
/* Cycles the issue buffer is not empty but 0 instructions issued */
PERF_COUNT_HW_ISSUE_STALLED = 19,
/* Cycles the branch buffer is not empty but 0 instructions issued */
PERF_COUNT_HW_BRANCH_ISSUE_STALLED = 20,
/* Cycles SRS0 is not empty but 0 instructions scheduled */
PERF_COUNT_HW_SRS0_SCHEDULE_STALLED = 21,
/* Cycles SRS1 is not empty but 0 instructions scheduled */
PERF_COUNT_HW_SRS1_SCHEDULE_STALLED = 22,
/* Cycles VRS is not empty but 0 instructions scheduled */
PERF_COUNT_HW_VRS_SCHEDULE_STALLED = 23,
/* Cycles LRS is not empty but 0 instructions scheduled */
PERF_COUNT_HW_LRS_SCHEDULE_STALLED = 24,
/* Cycles BRS is not empty but 0 instructions scheduled Load/Store,
* Data Cache, and dLFB Events */
PERF_COUNT_HW_BRS_SCHEDULE_STALLED = 25,
/*
* Load/Store, Data Cache, and Data Line Fill Buffer (DLFB) Events
*/
/* Total Ldst microops translated. */
PERF_COUNT_HW_TOTAL_TRANSLATED = 26,
/* Number of cacheable L* or EVL* microops translated. (This includes
* microops from load-multiple, load-update, and load-context
* instructions.) */
PERF_COUNT_HW_LOADS_TRANSLATED = 27,
/* Number of cacheable ST* or EVST* microops translated. (This
includes
* microops from store-multiple, store-update, and save-context
* instructions.) */
PERF_COUNT_HW_STORES_TRANSLATED = 28,
/* Number of cacheable DCBT and DCBTST instructions translated (L1
only)
* (Does not count touches that are converted to nops i.e. exceptions,
* noncacheable, hid0[nopti] bit is set.) */
PERF_COUNT_HW_TOUCHES_TRANSLATED = 29,
/* Number of dcba, dcbf, dcbst, and dcbz instructions translated
* (e500 traps on dcbi) */
PERF_COUNT_HW_CACHEOPS_TRANSLATED = 30,
/* Number of cache inhibited accesses translated */
```

```
PERF_COUNT_HW_CACHEINHIBITED_ACCESSES_TRANSLATED = 31,
/* Number of guarded loads translated */
PERF_COUNT_HW_GUARDED_LOADS_TRANSLATED = 32,
/* Number of write-through stores translated */
PERF_COUNT_HW_WRITETHROUGH_STORES_TRANSLATED = 33,
/* Number of misaligned load or store accesses translated. */
PERF_COUNT_HW_MISALIGNED_ACCESSES_TRANSLATED = 34,
/* Total allocated to dLFB */
PERF_COUNT_HW_TOTAL_ALLOCATED_DLFB = 35,
/* Loads translated and allocated to dLFB
* (Applies to same class of instructions as loads translated.) */
PERF_COUNT_HW_LOADS_TRANSLATED_ALLOCATED_DLFB = 36,
/* Stores completed and allocated to dLFB
* (Applies to same class of instructions as stores translated.) */
PERF_COUNT_HW_STORES_COMPLETED_ALLOCATED_DLFB = 37,
/* Touches translated and allocated to dLFB
* (Applies to same class of instructions as touches translated.) */
PERF_COUNT_HW_TOUCHES_TRANSLATED_ALLOCATED_DLFB = 38,
/* Number of cacheable ST* or EVST* microops completed.
* (Applies to the same class of instructions as stores translated.)
*/
PERF_COUNT_HW_STORES_COMPLETED = 39,
/* Number of cache lines locked in the dL1.
* (Counts a lock even if an overlock condition is encountered.) */
PERF_COUNT_HW_DL1_LOCKS = 40,
/* This is historically used to determine dcache miss rate (along
with
* loads/stores completed). This counts dL1 reloads for any reason. */
PERF_COUNT_HW_DL1_RELOADS = 41,
/* dL1 castouts. Does not count castouts due to DCBF. */
PERF_COUNT_HW_DL1_CASTOUTS = 42,
/*
* Data Side Replay Conditions: Times Detected
*/
/* Times detected replay condition - Load miss with dLFB full. */
PERF_COUNT_HW_DETECTED_REPLAYS = 43,
/* Load miss with load queue full. */
PERF_COUNT_HW_LOAD_MISS_QUEUE_FULL_REPLAYS = 44,
/* Load guarded miss when the load is not yet at the bottom of the
* completion buffer. */
PERF_COUNT_HW_LOAD_GUARDED_MISS_NOT_LAST_REPLAYS = 45,
/* Translate a store when the StQ is full. */
PERF_COUNT_HW_STORE_TRANSLATED_QUEUE_FULL_REPLAYS = 46,
/* Address collision. */
PERF_COUNT_HW_ADDRESS_COLLISION_REPLAYS = 47,
/* DMMU_MISS_REPLAYS : DMMU miss. */
PERF_COUNT_HW_DMMU_MISS_REPLAYS = 48,
/* DMMU_BUSY_REPLAYS : DMMU busy. */
PERF_COUNT_HW_DMMU_BUSY_REPLAYS = 49,
/* Second part of misaligned access when first part missed in cache.
*/
PERF_COUNT_HW_SECOND_PART_MISALIGNED_AFTER_MISS_REPLAYS = 50,
/*
* Data Side Replay Conditions: Cycles Stalled
*/
/* Cycles stalled on replay condition - Load miss with dLFB full. */
PERF_COUNT_HW_LOAD_MISS_DLFB_FULL_CYCLES = 51,
/* Cycles stalled on replay condition - Load miss with load queue
full. */
PERF_COUNT_HW_LOAD_MISS_QUEUE_FULL_CYCLES = 52,
/* Cycles stalled on replay condition - Load guarded miss when the
load is
* not yet at the bottom of the completion buffer. */
PERF_COUNT_HW_LOAD_GUARDED_MISS_NOT_LAST_CYCLES = 53,
/* Cycles stalled on replay condition - Translate a store when the
StQ
* is full. */
PERF_COUNT_HW_STORE_TRANSLATED_QUEUE_FULL_CYCLES = 54,
/* Cycles stalled on replay condition - Address collision. */
PERF_COUNT_HW_ADDRESS_COLLISION_CYCLES = 55,
/* Cycles stalled on replay condition - DMMU miss. */
PERF_COUNT_HW_DMMU_MISS_CYCLES = 56,
/* Cycles stalled on replay condition - DMMU busy. */
PERF_COUNT_HW_DMMU_BUSY_CYCLES = 57,
/* Cycles stalled on replay condition - Second part of misaligned
access
* when first part missed in cache. */
PERF_COUNT_HW_SECOND_PART_MISALIGNED_AFTER_MISS_CYCLES = 58,
/*
* Fetch, Instruction Cache, Instruction Line Fill Buffer (ILFB), and
* Instruction Prefetch Events
*/
/* Number of cache lines locked in the iL1. (Counts a lock even if an
* overlock condition is encountered.) */
PERF_COUNT_HW_IL1_LOCKS = 59,
/* This is historically used to determine icache miss rate (along
with
* instructions completed) Reloads due to demand fetch. */
PERF_COUNT_HW_IL1_FETCH_RELOADS = 60,
/* Counts the number of fetches that write at least one instruction
to the
* instruction buffer. (With instruction fetched, can used to compute
* instructions-per-fetch) */
PERF_COUNT_HW_FETCHES = 61,
/*
* Instruction MMU, Data MMU and L2 MMU Events
*/
/* iMMU TLB4K reloads */
PERF_COUNT_HW_IMMU_TLB4K_RELOADS = 62,
/* iMMU VSP reloads */
PERF_COUNT_HW_IMMU_VSP_RELOADS = 63,
/* dMMU TLB4K reloads */
PERF_COUNT_HW_DMMU_TLB4K_RELOADS = 64,
/* dMMU VSP reloads */
PERF_COUNT_HW_DMMU_VSP_RELOADS = 65,
/* Counts iTLB/dTLB error interrupt */
PERF_COUNT_HW_L2MMU_MISSES = 66,
/*
* BIU Interface Usage
*/
/* Number of master transactions. (Number of master TSs.) */
PERF_COUNT_HW_BIU_MASTER_REQUESTS = 67,
/* Number of master I-Side transactions. (Number of master I-Side
TSs.) */
PERF_COUNT_HW_BIU_MASTER_I_REQUESTS = 68,
/* Number of master D-Side transactions. (Number of master D-Side
TSs.) */
PERF_COUNT_HW_BIU_MASTER_D_REQUESTS = 69,
/* Stash request on Ain matches stash IDs for the core and are sent
to LFB */
```

```
PERF_COUNT_HW_BIU_STASH_REQUESTS = 70,
/* LFB signals snarf snoop response for ACRout for stash request */
PERF_COUNT_HW_BIU_STASH_ACCEPT = 71,
/*
* Snoop
*/
/* Number of externally generated snoop requests. (Counts snoop TSs.)
*/
PERF_COUNT_HW_SNOOP_REQUESTS = 72,
/* Number of snoop hits on all D-side resources regardless of the
cache
* state (modified, exclusive, or shared) */
PERF_COUNT_HW_SNOOP_HITS = 73,
/* Number of snoop pushes from all D-side resources.
* (Counts snoop ARTRY/WOPs.) */
PERF_COUNT_HW_SNOOP_PUSHES = 74,
/* Number of ACRout when the core retains a copy of the coherency
granule */
PERF_COUNT_HW_SNOOP_SHARING = 75,
/*
* Chaining Events
*/
/* Counts the number of times PMC0[32] transitioned from 1 to 0. */
PERF_COUNT_HW_PMC0_OVERFLOW = 82,
/* Counts the number of times PMC1[32] transitioned from 1 to 0. */
PERF_COUNT_HW_PMC1_OVERFLOW = 83,
/* Counts the number of times PMC2[32] transitioned from 1 to 0. */
PERF_COUNT_HW_PMC2_OVERFLOW = 84,
/* Counts the number of times PMC3[32] transitioned from 1 to 0. */
PERF_COUNT_HW_PMC3_OVERFLOW = 85,
/*
* Interrupt Events
*/
/* Number of interrupts taken */
PERF_COUNT_HW_INTERRUPTS = 86,
/* Number of external input interrupts taken */
PERF_COUNT_HW_EXTERNAL_INTERRUPTS = 87,
/* Number of critical input interrupts taken */
PERF_COUNT_HW_CRITICAL_INTERRUPTS = 88,
/* Number of system call and trap interrupts */
PERF_COUNT_HW_SC_TRAP_INTERRUPTS = 89,
/*
* Misc Events
*/
/* Counts transitions of the TBL bit selected by PMGC0[TBSEL] */
PERF_COUNT_HW_TBL_TRANSITIONS = 90,
/* Number L2 Linefill requests */
PERF_COUNT_HW_L2_LINEFILL_REQ = 91,
/* Number L2 Victim selects */
PERF_COUNT_HW_L2_VICTIM_SELECT = 92,
/* Speculative reservations in castout buffer that are not needed */
PERF_COUNT_HW_CASTOUTS_RELEASED = 93,
/* Allocations to INTV queue */
PERF_COUNT_HW_INTV_ALLOCATIONS = 94,
/* DLFB retries to MBAR */
PERF_COUNT_HW_STORE_RETRIES_MBAR = 95,
/* Retries to store queue, excluding MBAR case */
PERF_COUNT_HW_STORE_RETRIES_MISC = 96,
/*
* Stashing Events
```

```
*/
/* Stash hits in L1 */
STASH_L1_HIT = 97,
/* Stash hits in L2 */
STASH_L2_HIT = 98,
/* Cycles stash 1 DLFB busy */
STASH_BUSY_1 = 99,
/* Cycles stash 2 DLFB’s busy */
STASH_BUSY_2 = 100,
/* Cycles stash 3 DLFB’s busy */
STASH_BUSY_3 = 101,
/* A Access hits on stash DLFB */
STASH_HIT = 102,
/* Stash hits to a DLFB */
STASH_HIT_DLFB = 103,
/* Stash requests */
STASH_REQUESTS = 106,
/* Stash requests to L1 */
STASHES_L1_DATA_CACHE = 107,
/* Stash requests to L2 */
STASHES_BACKSIDE_L2 = 108,
/* Stalls due to no CAQ or COB */
STALLS_CAQ_COB = 109,
/*
* Backside L2 Events
*/
/* Number L2 cache accesses */
PERF_COUNT_HW_L2_ACCESS = 110,
/* Number L2 hit cache accesses */
PERF_COUNT_HW_L2_HIT_ACCESS = 111,
/* Number L2 data cache accesses */
PERF_COUNT_HW_L2_DATA_ACCESS = 112,
/* Number L2 hit data cache accesses */
PERF_COUNT_HW_L2_HIT_DATA_ACCESS = 113,
/* Number L2 instruction cache accesses */
PERF_COUNT_HW_L2_INST_ACCESS = 114,
/* Number L2 hit instruction cache accesses */
PERF_COUNT_HW_L2_HIT_INST_ACCESS = 115,
/* Number L2 cache allocations */
PERF_COUNT_HW_L2_ALLOC = 116,
/* Number L2 data cache allocations */
PERF_COUNT_HW_L2_DATA_ALLOC = 117,
/* Number L2 dirty data cache allocations */
PERF_COUNT_HW_L2_DIRTY_DATA_ALLOC = 118,
/* Number L2 instruction cache allocations */
PERF_COUNT_HW_L2_INST_ALLOC = 119,
/* Number L2 cache updates */
PERF_COUNT_HW_L2_UPDATE = 120,
/* Number L2 cache clean updates */
PERF_COUNT_HW_L2_CLEAN_UPDATE = 121,
/* Number L2 cache dirty updates */
PERF_COUNT_HW_L2_DIRTY_UPDATE = 122,
/* Number L2 cache clean redundant updates */
PERF_COUNT_HW_L2_CLEAN_REDU_UPDATE = 123,
/* Number L2 cache dirty redundant updates */
PERF_COUNT_HW_L2_DIRTY_REDU_UPDATE = 124,
/* Number L2 cache locks */
PERF_COUNT_HW_L2_LOCKS = 125,
/* Number L2 cache castouts */
PERF_COUNT_HW_L2_CASTOUT = 126,
/* Number L2 cache data dirty hits */
PERF_COUNT_HW_L2_HIT_DATA_DIRTY = 127,
/* Instruction lfb went high priority */
PERF_COUNT_HW_LFB_INSTR_HIGH_PRIO = 128,
/* Snoop throttling turned on */
PERF_COUNT_HW_SNOOP_THROTTLING_ON = 129,
/* Number L2 cache invalidation of clean lines */
PERF_COUNT_HW_L2_INV_CLEAN = 130,
/* Number L2 cache invalidation of incoherent lines */
PERF_COUNT_HW_L2_INV_INCOHER = 131,
/* Number L2 cache invalidation of coherent lines */
PERF_COUNT_HW_L2_INV_COHER = 132,
/*
* IAC, DAC Events
*/
/* Every valid IAC1 detection */
PERF_COUNT_HW_IAC1_DETECTED = 140,
/* Every valid IAC2 detection */
PERF_COUNT_HW_IAC2_DETECTED = 141,
/* Every valid DAC1 detection */
PERF_COUNT_HW_DAC1_DETECTED = 144,
/* Every valid DAC2 detection */
PERF_COUNT_HW_DAC2_DETECTED = 145,
/*
* DVT Events
*/
/* Detection of write to DEVENT with DVT0 set */
PERF_COUNT_HW_DVT0 = 148,
/* Detection of write to DEVENT with DVT1 set */
PERF_COUNT_HW_DVT1 = 149,
/* Detection of write to DEVENT with DVT2 set */
PERF_COUNT_HW_DVT2 = 150,
/* Detection of write to DEVENT with DVT3 set */
PERF_COUNT_HW_DVT3 = 151,
/* Detection of write to DEVENT with DVT4 set */
PERF_COUNT_HW_DVT4 = 152,
/* Detection of write to DEVENT with DVT5 set */
PERF_COUNT_HW_DVT5 = 153,
/* Detection of write to DEVENT with DVT6 set */
PERF_COUNT_HW_DVT6 = 154,
/* Detection of write to DEVENT with DVT7 set */
PERF_COUNT_HW_DVT7 = 155,
/* Number of completion cycles stalled due to Nexus FIFO full */
PERF_COUNT_HW_CYCLES_NEXUS_STALLED = 156,
/*
* FPU Events
*/
/* Double pump penalized ops finished through the pipe. Counts once
* for every multiply family double pump operation */
PERF_COUNT_HW_FPU_DOUBLE_PUMP = 160,
/* N.A. */
PERF_COUNT_HW_FPU_FINISH = 161,
/* Counts once for every cycle of divide execution. (fdivs and fdiv)
*/
PERF_COUNT_HW_FPU_DIVIDE_CYCLES = 162,
/* Counts extra cycles delay due to denormalized inputs. If there is
* one, this is incremented 4 times, Two operands increments it 5
* times. This shows the real penalty due to denorms, not just how
* often they occur */
PERF_COUNT_HW_FPU_DENORM_INPUT = 163,
/* Counts extra cycles due to denorm results, overflow, mass
* cancellation, zero results, carry-in mispredict, exponent range
check */
PERF_COUNT_HW_FPU_RESULT_STALL = 164,
/* N.A. */
PERF_COUNT_HW_FPU_FPSCR_FULL_STALL = 165,
/* Synchronization-op stalls: count once for each cycle that a
* “break-before” FPU is in the RS/issue stage but cannotissue. Also
* count once for each cycle that an FPU op is in the RS/issue stage
* but cannot issue due to “break-after”: of an FPU op currently in
* progress */
PERF_COUNT_HW_FPU_PIPE_SYNC_STALL = 166,
/* FPU data-ready stall: cycles in which there is an op in the
RS/issue
* stage that cannot issue because one or more of its operands is not
* yet available */
PERF_COUNT_HW_FPU_INPUT_DATA_STALL = 167,
/*
* Extended Load Store Events
*/
/* Number of decorated loads. */
PERF_COUNT_HW_DECORATED_LOAD = 176,
/* Number of decorated stores */
PERF_COUNT_HW_DECORATED_STORE = 177,
/* Number of load retries */
PERF_COUNT_HW_LOAD_RETRY = 178,
/* Number of successful stwcx. instructions */
PERF_COUNT_HW_STWCX_SUCCESS = 179,
/* Number of unsuccessful stwcx. instructions */
PERF_COUNT_HW_STWCX_UNSUCCESS = 180,
} pspperf_event_e;
```

6.4.7 Moving a project from PikeOS3.5 back to PikeOS 3.4

When a project has been developed with PikeOS 3.5 and as we want project to move to PikeOS3.4 we provide here the way to proceed with project migration, starting with integration projects and then application project.

Using command line:

```
- cd yourP35project
- source PikeOS.sh
- make prepare-cvs


- cd newP34project
- /opt/pikeos-3.4/bin/pikeos-shareproject import

Configuration files have been generated.
- make boot / install
```

6.4.8 ARINC653

6.4.8.1 APEX P1s2

The ARINC653 P1s2 is available with both PikeOS 3.4. It is an “all in one” implementation that may introduce some jitter on API level due to blocking operation.

6.4.8.2 APEX Time Composable

A Time Composable ARINC653 personality called APEXTC in the PROXIMA scope is available with both PikeOS 3.4. It is derived from the product line ARINC653 P1s2. The Time Composable stands for a modified ARINC653 for operations on queuing and sampling ports as explained bellow (section 6.4.11) for compliancy with PROARTIS results. Then all the intra partition communication APEX services but EVENTS are removed from the personality. This personality is not provided as product line since it is not compliant with ARINC653 specification at API level for queuing and sampling ports. Thanks to PROXIMA it is provided with an optimized implementation when the partition addresses some assumptions:

- Non-blocking IOs
- No race conditions inside the partition (no concurrent access to the same port side in a partition)

Then the Time Composability of services is improved when the configuration is set as following, as provided with the demo projects using the dedicated BSP:

- Kernel code and data is allocated into a no cache area. For a performance issue, the kernel can be allocated into a scratchpad RAM or a L3 cache set as SRAM so that there may be no or only a small latency depending on the selected hardware. This is supported by a dedicated BSP.
- Both the PSSW and the APEX library code and data are allocated into a no cache area. For a performance issue, the no cache segments can be allocated into a scratchpad RAM or a L3 cache set as SRAM so that there may be no or only a small latency depending on the selected hardware. This is only a matter of link process in the application project. The delivered *proxima-ibit{personality}-{board}* demo projects will implements this feature as an how to.

6.4.8.2.1 Overview

The new Time Composable ARINC653 TC implementation is available for both PROXIMA-FPGA and PROXIMA-COTS boards. The Time Composable stands for a modified ARINC653 for anticipated read and deferred write operations on queuing and sampling in compliancy with PROARTIS results. The modifications done on queuing and sampling ports are described in the section *“Queuing and sampling port enhancement”*.

A project demo has been included to demonstrate the usage of this new personality. The personality and the project demo are only available with PikeOS 3.5 in the BSP V1.2.1 and higher delivery.

6.4.8.2.2 Create a new Time Comp. integration project using project demo

First, we will create the integration project using the given integration project demo. This demo contains 2 different PikeOS partitions with the following applications:

- "producer" application writes periodically in two Queuing ports and two sampling ports.
- "consumer" application reads periodically from two Queuing ports and two sampling ports. Read messages are then displayed in the console with the validity for the sampling ports.

Clone the Integration Project using these values:
For the leon3 PROXIMA FPGA:
Project type: *PikeOS Integration Project*
Project name: *simple-tc-port-leon3.int*
Template: *simple-tc-port.int*
Board name: *leon3-proxima-smp*
Boot strategy: *elf*
For the P4080 board:
Project type: *PikeOS Integration Project*
Project name: *simple-tc-port-p4080ds.int*
Template: *simple-tc-port.int*
Board name: *p4080-proxima*
Boot strategy: *uboot-dtb*
For qemu (sparc v8):
Project type: *PikeOS Integration Project*
Project name: *simple-tc-port-qemusparc.int*
Template: *simple-tc-port.int*
Board name: *qemu-sparc*
Boot strategy: *qemu*

6.4.8.2.3 Create a new Time comp. APEX application using demo project

Clone the application project with these values:
For the leon3 PROXIMA FPGA and qemu (sparc v8):
Project type:*APEX*
Project name:*simple-tc-port-sparc.app*
Template:*simple-tc-port*
Architecture:*sparc*
Processor type:*v8*
For the P4080 board:
Project type:*APEX*
Project name:*simple-tc-port-ppc.app*
Template:*simple-tc-port*
Architecture:*ppc*
Processor type:*e500mc*

If you want to use the new Time Composable ARINC653 TC implementation, you have to set the "Apex part Time Composable" in the project.xml.conf with the Project Configurator Editor.

The new APEX API is mainly the same (see APEX Reference Manual); there are only two modifications: the "reading functions" of the queuing ports and the sampling ports now take the reference of the buffer address to read instead of the buffer address itself. This buffer address is now provided by the APEX API and it will have the same size than the MAX_MESSAGE_SIZE given at the creation of the port. The new synopses are:

```
void RECEIVE_QUEUING_MESSAGE(
        QUEUING_PORT_ID_TYPE QUEUING_PORT_ID,
        SYSTEM_TIME_TYPE TIME_OUT,
        MESSAGE_ADDR_TYPE *MESSAGE_ADDR,
        MESSAGE_SIZE_TYPE *LENGTH,
        RETURN_CODE_TYPE *RETURN_CODE)

void READ_SAMPLING_MESSAGE(SAMPLING_PORT_ID_TYPE SAMPLING_PORT_ID,
        MESSAGE_ADDR_TYPE *MESSAGE_ADDR,
        MESSAGE_SIZE_TYPE *LENGTH,
        VALIDITY_TYPE *VALIDITY,
        RETURN_CODE_TYPE *RETURN_CODE)
```

Note: The deferred write implies that user shall not modify the buffer to be written before next period, otherwise, the new content will be written in queuing or sampling port.

6.4.8.2.4 Build and run the project demo

Perform the following steps to build the project (for PROXIMA FPGA):

- Open simple-tc-port-sparc.app → project.xml.conf
- Set "Apex part Time Composable" to enable the queuing and sampling port enhancements
- Set your integration project to "simple-tc-port-*fpga.int"*
- simple-tc-port-fpga.app → Make install
- simple-tc-port-fpga.int → Make boot
- Boot the ROM Image (see platform manual for more details about the boot configuration). You should see this in the console (this log is simplified):

```
Producer APEX application starting up.
Consumer APEX application starting up.
Read <> from sport S_SIMPLE_0_DST. Validity: INVALID
…
Read <Hello World message0 from sport> from sport S_SIMPLE_0_DST.
Validity: VALID
```

```
…
Read <Hello World message0 from qport> from qport Q_SIMPLE_0_DST.
```

It means that the first valid message read by the sampling port 0 is “`Hello World message0 from sport`” and the first message read by the queuing port 0 is “`Hello World message0 from qport`”.

6.4.8.2.5 Configure APBRANDBANK driver (optional)

The APBRANDBANK module is a hardware component attached to the APB bus of the LEON-PROXIMA SoC. This component implements the different pseudo-random number generators required for the randomization of the different hardware described in D1.1.

As described in section 6.1.4, the PSP has been modified to support random placement/replacement strategy. Besides, the new APEX Time Composable personality has been updated to refresh the fixed seeds of the module at each APEX process switch. To do this, we use a PSP device driver to reset the seeds in the kernel which implies that the APEX application must have access rights to use the PSP device.

The APBRANDBANK driver does the following operations:

- At boot time
    - Initialize all the seeds
- At time partition switch
    - Refresh the fixed seeds with new values
- At APEX process switch
    - Refresh the fixed seeds with new values

At the startup of the demo application, you should see these messages:

```
APEX 1: APEX TC mode enabled
APEX 1: APBRANDBANK not supported in this configuration
APEX 2: APEX TC mode enabled
APEX 2: APBRANDBANK not supported in this configuration
```

It shows that the APEX partitions (pid 1 and 2) are using the TC mode but the APBRANDBANK driver is not configured. For the PROXIMA-FPGA board, it means that the seeds will not be updated at process switch.

To activate support for APBRANDBANK on APEX process switch you need first to perform the following steps to give the APEX TC application access to the APBRANDBANK device driver (and thus enable the seeds refresh at process switch):

1. Open *simple-tc-port-leon3.int → vmit.xml*
2. Expend *PartitionTable → Partition:consumer*
3. Right click on *FileAccessList* and select *Add*

4. Set *FileName* to "*prop:board/drv/apbrandbank_fp*"*
5. Check *Map* box in *AccessMode*
6. Redo the step 1-5 for the partition "producer"
7. Save and build the project

6.4.8.3 APEX NG

The new generation ARINC653 P1s3 implementation is available on PikeOS 4.0 release but is not available for the PROXIMA project since SPARC is not yet supported. This will be a "one to one" implementation providing less jitter for increased Time Composability when combined with well suited memory configuration. As explained in section 6.4.11 it is available with a new queuing and sampling port implementation that allows IO partition mode with low jitter. Thanks to PROXIMA it is also provided with an optimized queuing (TCQPORT) and sampling port (TCSPORT) implementation when the partition addresses some assumptions:

- Non-blocking IOs
- No race conditions inside the partition (no concurrent access to the same port side in a partition)

Then the Time Composability of services is improved when the configuration is set as following, as provided with the demo projects using the dedicated BSP:

- Kernel code and data is allocated into a no cache area. For a performance issue, the kernel can be allocated into a scratchpad RAM or a L3 cache set as SRAM so that there may be no or only a small latency depending on the selected hardware. This is supported by a dedicated BSP.
- Both the PSSW and the APEX library code and data are allocated into a no cache area. For a performance issue, the no cache segments can be allocated into a scratchpad RAM or a L3 cache set as SRAM so that there may be no or only a small latency depending on the selected hardware. This is only a matter of link process in the application project. The delivered *proxima-ibit{personality}-{board}* demo projects will implements this feature as an how to.

6.4.9 RTEMS v4.10

The RTEMS v4.10 implementation is available with PikeOS 3.4.1 for SPARC architecture only. It is an "all in one" implementation. It is not designed for application of highest criticality level.
RTEMS v4.10 is not available in the current product line since not yet been requested to SYSGO by any industry.
RTEMS v4.10 has been provided in the scope of the project to supply the study on the PROXIMA-FPGA board.
Note this RTEMS BPS layer is mapped to periodic ticker and so will not benefit on dynamic ticker implementation.

6.4.10ELinOS

The ELinOS 5.2 P4_Linux is available with PikeOS 3.4 and upper for both secure IO and direct IO. It is not designed for application of criticality level higher than DAL D.

The ELinOS 6.0 P4_Linux is available with PikeOS 4.0 and upper for both secure IO and direct IO. It is not designed for application of criticality level higher than DAL D.

ELinOS for SPARC architectures is not available in the current product line since not yet requested by any industry.

The ELinOS 5.2 P4_Linux has been provided for PikeOS 3.4 in the scope of the project to supply the study on the PROXIMA-FPGA board.

6.4.11Time Composable Queuing and Sampling port

The default queuing and sampling ports available for inter-partition communication are designed to manage race conditions between several partitions in multi core context using blocking IOs. Those ports can be accessed from application with any guest OS (RTEMS, ELinOS, ARINC653, Native…).

Atop PikeOS 3.4 a **Time Composable improved port provider version for the APEXTC** personality is available within the project and another one is available as external port provider that can be used by any personality. Note that both have modified API.

The **APEXTC** is not available as industrial product since it is not compliant with ARINC653 specification but specified in PROARTIS. This implementation is like an IO partition concept and does the real part of IO outside the time slot of the partition using shared memory for buffers:

- the non-blocking read operation is only to get the pointer to the pre-read buffer as a deferred action (non ARINC653 compliant syntax for the buffer parameter that is there a pointer to buffer pointer instead of a buffer pointer),
- the non-blocking write operation is only to register the buffer pointer for deferred action (the caller shall not modify the buffer until the end of time slot where it is really flushed).
- the IO partition that does the transfer is statically defined as time partition ID 1 to be allocated either before and after the user partition or in a different core so that it will not interfere with calling application.

With **BSP v1.3.2** a new **Time Composable external port provider** is delivered as part of the project but can be added as any system extension to the product line. Therefore the API is somehow a little bit modified in the semantic to act as an IO partition doing the real part of the transfer in the service level assigned to a separated core for limited interference and constant crossing time of read/write services. The idea is to record a message descriptor given as argument to the write service. Then the service can record it in a constant time in the port provider fifo. The port provider will do the work as part of other task in other core. For the read service it is the same, the caller will give a message buffer descriptor that will be fed by the read call. The read consist in providing the caller with a pointer to the message that is hold in the port provider space. Please look at proxima-ibit.app/src/test_vm_TCports.c for an example of how to use this Time Composable port provider. This port is a generic external port so it can be used by any personality. But note that when encapsulated in a guest OS call, it may not provide the same performance and may lose the “time composable” capability; for example with APEX you will get better performance than the legacy port since it will no more depend on the message size, but it has still some large jitter compared to the dedicated APEX TC port described in the section above.

This implementation is like an IO partition concept and does the real part of IO outside the time slot of the partition using shared memory for buffers:

- the non-blocking read operation fill on return a *portprovider_tc_read_message_desc_t* buffer to get the pointer to the pre-read buffer as a deferred action as well as a pointer to the read size. In case of sampling message, an additional pointer to the validity is also available.

```
typedef struct{
    /* Data size */
    P4_size_t *size;
    /* Data content pointer */
    void *data;
    /* Validity pointer : samplingg port message validity */
    vm_sport_msg_validity_t *validity;
} portprovider_tc_read_message_desc_t;
```

- the non-blocking write operation register a *portprovider_tc_write_message_desc_t* buffer that includes message data pointer for deferred action (the caller shall not modify the message data until the end of time slot where it is really flushed).

```
typedef struct{
    /* Data content pointer */
    void *data;
} portprovider_tc_write_message_desc_t;
```

- The IO partition that does the transfer is part of PSSW and each port thread support can be allocated to a given core per affinity. The default is to run within same core than port create caller (scua). To reduce interference with caller code it is convenient to allocate the TC port provider IO stack to a different core than user application. The core affinity is set on a port basis and is defined as a CpuMask in the PortConfigData field decimal value for the corresponding QueingPort or SamplingPort of the PortProviderExtension in the vmit.

6.4.12 Writing measurements and events to host file

At some time it is required to send some data (events measurement…) to the host as a file. The default way is to print to the serial console but it very low. The second way is to use an Ethernet channel through MUXA.

First in the target application you have to open the muxa channel as described in fundamentals.pdf section 4.2.4

```
#define MBTA_CHANNEL_NAME "muxa:/mbta"

/* init sequence*/
vm_file_desc_t mbta_fd;

vm_open(MBTA_CHANNEL_NAME, VM_O_WR, &mbta_fd);
```

then you can write data to this channel in a simple way like for any socket:

```
void dump_string_to_host(char * str)
{
        P4_size_t written;
        P4_size_t size = strlen(str);
        vm_e_t rc;

        rc = vm_write(&mbta_fd, str, size, &written );
        /* check rc for error */
}
```

But before you need also to configure the integration project as described in the fundamentals.pdf section 4.2 :

- Set IP configuration in project.xml.conf service – muxa.
- Set port 1509 name as MBTA (should be other port and other name but this fit the above example)
- Make boot
- Run muxa as a CODEO target or from a command line.
- telnet the channel from a host linux session to log into host file:
  ```
  telnet localhost 1509 >myfile.log
  ```
- Start the target application

7 PikeOS Time Composability and MBPTA

7.1 Overview

PikeOS provides the capability to supply several partitions with various criticality level based on a time and space segregation concept from IMA. For highest criticality level application it is expected to provide WCET for the API to be called at operational time. There are several ways of providing the WCET for a service; First way is to do a Static Timing Analysis (STA) where it is required to get the exact number of cycles the service will consume running the service for any condition. This is a huge and costly work including a dependability analysis that provides the fired events that can impact the Execution Time (ET), the jitter part of the WCET. A second approach called Measurement Based Timing Analysis (MBTA) is to estimate the cause of jitter and do ET measurement on representative target during MCDC verification. The WCET is estimated as being the sum of the maximum execution time we can measure and a margin to be defined with certification authorities based on confidence with the method. Some third part provides efficient tools to implement such an MBTA method with good confidence level. This method is still costly but less than the STA. Specifically considering using cache on a single core were cache history has huge impact on result and the STA may produce conservative large WCET that can lead to remove cache usage.

PROXIMA suggest a way to have no more dependency on cache history based on cache placement/replacement randomization. It also allows to use same randomization technique where possible to get a maximum value instead of having bounded values to the maximal. And this can be combined with standard way to improve the global performance. This Measurement Based Probabilistic Timing Analysis (MBPTA) can be used now with PikeOS.

The usual PikeOS space and time segregation allows to reduce costs applying the Timing Analysis only where required; as an example, if an application with no timing constraint is run besides one with highest level of criticality, PikeOS allows to focus on this last one with assumption the other will not interfere with it. This is a basement for mixed criticality support with PikeOS but it has to be demonstrated from analysis.

PROXIMA states for example the cache has no more interference on the code under evaluation and the MBPTA tool manage the way to produce a WCET based on validated mathematical assumptions. The MBPTA is compliant with the time and space partitioning provided by PikeOS; For example the cache randomization technique will provide an average ET value that is close to the WCET. This is acceptable for the critical application but probably not acceptable for the less critical that may want best possible performance. PikeOS segregation will provide the capability to apply the MBPTA only on the critical application and let other application using standard efficient way in a mixed criticality context.

In case of mixed criticality support it is required to only apply hardware randomization on dedicated partition. Else the low criticality application will have average value that is not the best but close to WCET.

On the other hand the software randomization developed by PROXIMA for COTS hardware is well suited for mixed criticality as supported by PikeOS since it can be applied on a per partition basis. And as we cannot expect from now to have hardware

randomization support on COTS processor, this way seems today the best possible in industrial way.
In this section we will recall the way to get the best possible ET properties for a PikeOS API service and for the application depending on the target criticality level.

7.2 Safety guidance elements

The goal here is not to provide some safety manual based on a full dependability analysis but to identify the main pitfalls we can avoid with PikeOS configuration when estimating WCET for a service API.

The main concern for an OS service API is the jittery introduced by the following causes:

- Preemption on time partition switch raised asynchronously in the algorithm introduces a latency equal to the WCET of the time partition switch time and this may vary with the jitter of this interrupt. Of course the ET of the time partition switch has to be as constant as possible (low or null jitter) but the best is to have run to completion algorithm that means the algorithm is periodic and has to enter a specific API service waiting for the next time slot. This synchronization point can have optimized implementation to reduce jitter.
- Preemption on blocking situation like a blocking IO may result in a switch to a lower priority thread/process until the completion of the blocking situation signaled by the IO provider. The blocking point may be evaluated each time a signal is send to IO consumers for completion evaluation. Then the ET of such an API may not be deterministic or may be bounded to a large value that can be unacceptable. The best practice to reach best possible WCET is to use non-blocking IO.
- Preemption on some synchronization object like mutex, semaphore or event that is used to protect a critical section inside an API service. The best practice to reach best possible WCET is to use non-blocking IO and run to completion paradigm in a non-pre-emptible time slot; PikeOS allows to allocate only one partition to a time slot and if the partition does call blocking services, it will not be preempted until the normal completion in the period. Using APEX periodic process is the best mean to reach this behavior.
- Preemption on external interrupt can be avoided with PikeOS using some properties and adapted “application” design:
  - When a partition (or the PSSW) do register to a given interrupt, then the interrupt source is unmasked for the given core. As PikeOS uses core #0 as master and then some service like Ethernet support may generate interrupts to be serviced by PSSW then moving critical partition to other core than #0 will avoid being interrupted.
  - Use the dynamic timer mode instead of the periodic timer. The dynamic will generate timer interrupt only when required when the periodic will generate a periodic ticker interrupt on any.
- Some PikeOS API have Execution Profile with several execution paths, specifically when using System call that may lead to several paths and high underneath complexity. This is not a blocking point but may produce huge work at timing analysis time. An application that may raise highest level of criticality shall take into account such aspect and avoid some API in the operational phase of the software.

- Algorithm depending on input data value or data size is not blocking but generate much more work at time analysis and is not Time Composable from the application point of view. An application that may raise highest level of criticality shall take into account such aspect and avoid some API in the operational phase of the software.
- Cache history impact can be reduced as demonstrated by allocating PikeOS kernel data and code to a no cache area. Moreover the libraries like the PSSW and the guest OS can also be allocated to no cache area avoiding cache history impact at application level.
- Hardware properties (instruction execution time, bus arbitration, bandwidth limitation) has impact on API WCET and may introduce jitter to be taken into account. PikeOS allows implementation of IO server that can help regulating IO bandwidth and providing margins.

7.3 PikeOS allows Cache jitter reduction

The main cause of jitter for the WCET of an API service that should have a constant or determined ET is the number of cache hit/miss that may result of calling a service when coming back to your user code.

You can bound the jitter with the max number of misses the service will generate on its first execution with a full clean cache on entry and also clean cache on exit so that any possible miss and flush is taken into account in the ET.

But measuring the ET of a user code that includes several calls to an OS service may result in variable ET for various reason and then it is no more time composable. The time really allocated to the user code versus the service call is no more evaluable with enough precision:

- because the cache may generate various number of hit on each call,
- the code execution is preempted/interrupted by other code that will modify the cache state
- the cache is shared with several cores

Thus PikeOS allows jitter reduction based on the technics in the following subsections.

For any of the features there is the SYSGO internal analysis about compliancy of the PROXIMA results to the PikeOS industrial constraints through colored notes; PROXIMA feature may be implemented in one or more components from one or several contributors:

- Green: when feature implementation is compliant with industrial need

- Italic orange: when feature partially implemented or implementation is partially compliant with the industrial need

- Bold red: when feature implementation is not compliant with industrial need

7.3.1 PikeOS allows Cache flush at time partition switch time

When there are several partitions to run sequentially on a core, sharing the same cache introduces interferences between partitions. At partition switch time coming back may not be in the same cache context than on previous context switch. This introduces variability in the number of cache miss/hit.

PikeOS allows to activate cache flush on time window switch so that a periodic program can recover the same cache context on each time window.

This is configured using flags (VM_SCF_FLUSH_TLB) in the vmit, see psswref.pdf section 2.6.1 The Window Element

This also make the assumption the program is periodic and is waiting for the next period when the time partition switch occurs. This is typical use case for ARINC653 application using service WAIT_NEXT() that is implemented using PikeOS native service p4_sleep() with flag `P4_TIMEOUT_TP_ALL`.

7.3.2 PikeOS allows Cache partitioning

As explained in above section, sharing the same cache between partitions introduces interferences since cache context may change on each partition switch. Sharing the cache between cores also produces variability in the number of cache miss/hit because at any time a cache line can be evicted by execution of another core.

PikeOS allows to partition the cache when the hardware is capable and the PSP may supply.

7.3.2.1 PikeOS allows per core L1/L2 cache

7.3.2.1.1 PSP PROXIMA-FPGA allows per core L1/L2 cache separation

The provided PROXIMA -FPGA has been requested to be configured with L2 as one way per core when the PTA mode is selected.

Note: it should have been great to have this feature available in LRU mode also.

The FPGA L1 cache is not shared between cores so that it will not interfere. Both data and instruction L1 caches are in ghost mode avoiding the need for snooping at L1 level.

See section 6.1 of this document for PikeOS configuration through PSP TAGs.

7.3.2.1.2 PSP P4080 has per core L1/L2 cache separation

The P4080 provides L2 cache per core and the shared L3 can be configured as SRAM so that it is no more shared as a cache.

See section 6.2 of this document for PikeOS configuration through PSP TAGs.

7.3.2.2 PikeOS allows L2 cache partitioning

PikeOS could be able to provide cache partitioning in the sense of allocating part of the cache exclusively to one partition. In such configuration any program might be periodic or not, can recover the previous cache context after the switch to the next time window. There is no more interference between cores when partition is not spread over cores but assigned to one core only.

But this requires the hardware to provide the required support.

7.3.2.2.1 PROXIMA -FPGA does not support L2 cache partitioning

The PROXIMA-FPGA does not provide any cache partitioning support. The impact on deterministic and determined behavior is as follows:

- PikeOS allows running periodic schema based on IMA concept. In this frame PikeOS ensure any critical partition having exclusive usage of the cpu/core for for its allocated time. PikeOS provides configuration option to flush caches and TLB when entering the time slot so that critical application can start period with always same cache context.
- **When several partitions are sharing cpu/core during a time slot in a preemptive strategy, the hardware randomization need to flush cache when switching partitions (changing virtual space) introducing large latency that is acceptable neither for critical partition nor for non critical**. PikeOS allows this feature for project evaluation but should not be provided in an industrial product.

7.3.2.2.2 P4080 does not support L2 cache partitioning

The PROXIMA-FPGA does not provide any cache partitioning support. The impact on deterministic and determined behavior is as follows:

- PikeOS allows running periodic schema based on IMA concept. In this frame PikeOS ensure any critical partition having exclusive usage of the cpu/core for for its allocated time. PikeOS provides configuration option to flush caches and TLB when entering the time slot so that critical application can start period with always same cache context.
- **When several partitions are sharing cpu/core during a time slot in a preemptive strategy, it introduces latency when switching partitions (changing virtual space) that is not acceptable considering critical partition only** but acceptable for non-critical partition.

7.3.2.3 PikeOS allows non-cacheable L3

The P4080 provides shared L3 cache that can be configured with PikeOS PSP as SRAM so that it is no more shared as a cache avoiding interference.

See section 6.2 of this document for PikeOS configuration through PSP TAGs.

7.3.3 PikeOS allows cache history impact reduction at user level

The obvious way to remove the number of cache hit/miss variability that may result of calling an OS service when coming back to the user code is to install the code and data of the API into a no cache area.

The API is composed of several level in PikeOS:

- µkernel with code and data segment
- pssw with code and data segment
- guest OS library with code and data segment

When a partition does want Time Composability feature, both the 3 levels of API should be optimized for Time Composability. The following sections shows how to implement this feature at the various level as this is configurable on PikeOS.

7.3.3.1 Moving PikeOS kernel code and datato a no cache area

7.3.3.1.1 Using dedicated Sratchpad/Static RAM with PikeOS 3.4

A specific BSP has been provided to allocate the µkernel into the FPGA scratchpad RAM a no cache area with high performance. See section 8.2.2.5 for usage details.

Note that only the code has been moved to the no cache area since there is not enough room for data into the scratchpad RAM. But the goal here is to demonstrate properties to be established in the industrial version, and requirements exposed for such hardware.

Unfortunately from the version 2 of the FPGA the Scratshpad RAM is no more available. Instead a static RAM has been provided but in a memory range thatis not regular and that cannot be allocated for kernel due to some internal assumption on the memory range for this architecture. As a workaround the following feature has been provided.

7.3.3.1.2 Running kernel in no cache area from PSP TAG option

Both PROXIMA-FPGA and P4080 BSP have been provided with PSP TAG option to allocate the µkernel into standard DRAM but with no cache area attribute. See section 6.2.1 for usage details.

Note the goal here is to demonstrate properties to be established in the industrial version, and requirements exposed for such hardware. In particular it will be interesting to use fine granularity memory allocation with PikeOS v4.X along with locking capability for code and data rather than no cache area for better performance. But this requires enough cache at L2 level.

7.3.3.2 Moving pssw level code and data to no cache area

The PSSW is implemented as a library that is linked with the guest OS into the application project. Thus it cannot be moved to no cache area lonely but as part of the link process into the application project. So please refer to the link process description in the following sections for the guest OS corresponding to your need.

7.3.3.3 Moving guest OS level code and data to no cache area

7.3.3.3.1 Moving APEX-TC code/data to no-cache area

The /demos-apex/ibitapextc application project provides an example on how to configure makefile and ldscript to move the code and data segment of pssw and apex-tc library into a no cache memory segment.

The principle is to modify the APEX_LDFLAGS variable of the Makefile so that you ld will call a local to project ldscript that will do the work; in the given example the local ldscript is named app-ld-script and the Makefile is modified with this "-T" argument:

```
-T$(PIKEOS_PROJECT)/app-ld-script
```

The full variable definition is

```
APEX_LDFLAGS :=
-L/opt/pikeos-$(PIKEOS_VERSION)/target/sparc/v8/lib
-L/opt/pikeos-$(PIKEOS_VERSION)/target/sparc/v8/apex-tc/lib
-lapex-tc -lxt -lvm -lp4 -lstand -e _begin          -L/opt/pikeos-
$(PIKEOS_VERSION)/target/sparc/v8/lib    -L/opt/pikeos-
$(PIKEOS_VERSION)/cdk/sparc/v8/lib/gcc/sparc-unknown-elf/4.4.5/ -lgcc
```

```
      -T$(PIKEOS_PROJECT)/app-ld-script
```

The app-ld-script is the default script from PikeOS CDK modified to define flags for cache and no cache areas:

```
PHDRS
      {
       text_cache_off PT_LOAD FLAGS(5);
       data_cache_off PT_LOAD FLAGS(6);
       text_cache_on  PT_LOAD FLAGS(5);
       data_cache_on  PT_LOAD FLAGS(6);
    }
```

And then moving the data and code elements of the libraries in the corresponding sections:

```
    .text.nocache :
    {
       /opt/pikeos-3.4/target/sparc/v8/apex-tc/lib/libapex-tc.a:(.text)
       /opt/pikeos-3.4/target/sparc/v8/lib/libvm.a:(.text)
       /opt/pikeos-3.4/target/sparc/v8/lib/libxt.a:(.text)
/opt/pikeos-3.4/target/sparc/v8/lib/libp4.a:(.text)
/opt/pikeos-3.4/target/sparc/v8/lib/libbit.a:(.text)
    } : text_cache_off

    .data.nocache ALIGN(0x1000):
    {
    /opt/pikeos-3.4/target/sparc/v8/apex-tc/lib/libapex-tc.a:(.data .data.*
.gnu.linkonce.d.* .data1 .sdata)
    /opt/pikeos-3.4/target/sparc/v8/lib/libvm.a:(.data .data.*
.gnu.linkonce.d.* .data1 .sdata)
    /opt/pikeos-3.4/target/sparc/v8/lib/libxt.a:(.data .data.*
.gnu.linkonce.d.* .data1 .sdata)
/opt/pikeos-3.4/target/sparc/v8/lib/libp4.a:(.data .data.*
.gnu.linkonce.d.* .data1 .sdata)
/opt/pikeos-3.4/target/sparc/v8/lib/libbit.a:(.data .data.*
.gnu.linkonce.d.* .data1 .sdata)
    } : data_cache_off
```

7.3.3.3.2 Moving Native code/data to no-cache area

The /demos-apex/ibitvmtc application project provides an example on how to configure makefile and ldscript to move the code and data segment of pssw and apex-tc library into a no cache memory segment.

The principle is to modify the PIKEOS_LD_FLAGS variable of the Makefile so that you ld will call a local to project ldscript that will do the work; in the given example the local ldscript is named app-ld-script and the Makefile is modified with this "-T" argument:

```
      -T$(PIKEOS_PROJECT)/app-ld-script
```

The full variable definition is

```
      PIKEOS_LD_FLAGS :=
      -L/opt/pikeos-$(PIKEOS_VERSION)/target/sparc/v8/lib
      -lvm -lp4 -lstand -e _begin
```

```
       -L/opt/pikeos-$(PIKEOS_VERSION)/cdk/sparc/v8/lib/gcc/sparc-unknown-
       elf/4.4.5/ -lgcc
       -T$(PIKEOS_PROJECT)/app-ld-script
```

The app-ld-script is the default script from PikeOS CDK modified to define flags for cache and no cache areas:

```
PHDRS
{
    text_cache_off PT_LOAD FLAGS(5);
    data_cache_off PT_LOAD FLAGS(6);
    text_cache_on  PT_LOAD FLAGS(5);
    data_cache_on  PT_LOAD FLAGS(6);
}
```

And then moving the data and code elements of the libraries in the corresponding sections:

```
    .text.nocache :
    {
       /opt/pikeos-3.5/target/sparc/v8/lib/libvm.a:(.text)
       /opt/pikeos-3.5/target/sparc/v8/lib/libp4.a:(.text)
       /opt/pikeos-3.5/target/sparc/v8/lib/libbit.a:(.text)
    } : text_cache_off

    .data.nocache ALIGN(0x1000):
    {
    /opt/pikeos-3.5/target/sparc/v8/lib/libvm.a:(.data .data.*
.gnu.linkonce.d.* .data1 .sdata)
/opt/pikeos-3.5/target/sparc/v8/lib/libp4.a:(.data .data.*
.gnu.linkonce.d.* .data1 .sdata)
/opt/pikeos-3.5/target/sparc/v8/lib/libbit.a:(.data .data.*
.gnu.linkonce.d.* .data1 .sdata)
    } : data_cache_off
```

7.3.3.3.3 Moving RTEMS code/data to no-cache area

RTEMS v4.10 is a Linux implementation from ESA that is not intended to be certified at high criticality level.

So RTEMS v4.10 can benefits from µkernel optimization when calling it, but there is no local to RTEMS v4.10 optimization for Time Composability.

The only "Time Composable" feature RTEMS v4.10 can take benefit from PikeOS is the access to queuing and sampling port as for APEX NG and Native on PikeOS 4.0. But note that the micro kernel service is encapsulated in the RTEMS socket API that can introduce other level of jittery.

The only reachable optimization is to move code and data of the RTEMS kernel level into a no cache area. The way is similar to what is described in above sections 7.3.3.3.1 & 7.3.3.3.2. But where it was easy to identify 3 or 4 libraries to be separated, with RTEMS it becomes a huge work depending on the actual RTEMS configuration for a given partition. And this will have a significant impact on performances.

7.3.3.3.4 Moving ELinOS code/data to no-cache area

ELinOS has full control of the memory segment allocated to the virtual machine it is running on. ELinOS is a standard kernel and then it is not expected to modify it.

So ELinOS can benefits from µkernel optimization when calling it, but there is no local to ELinOS optimization for Time Composability.

The only “Time Composable” feature ELinOS can take benefit from PikeOS is the access to queuing and sampling port as for APEX NG and Native on PikeOS 4.0. But note that the micro kernel service is encapsulated in the Linux socket API that can introduce other level of jittery.

The only reachable optimization is to move code and data of the Linux kernel level into a no cache area. The way is similar to what is described in above sections 7.3.3.3.1 & 7.3.3.3.2. But where it was easy to identify 3 or 4 libraries to be separated, with ELinOS it becomes a huge work depending on the actual Linux configuration for a given partition. And this will have a significant impact on performances

7.3.4 PikeOS provides support for cache randomization

In the frame of PROXIMA and to study impact of cache randomization, the PikeOS PSP for the FPGA has been modified to provide the following features:

7.3.4.1 Hardware Cache Randomization support

See section 6.3.2.1.2 Hardware support for PTA

7.3.4.1.1 Changing Hardware seed at Partition switch

On time partition switch the kernel call the psp_desc.api.tp_switch that implements the following algorithm when PTA mode is activated (L1L2_PTA is true):

```
/* write an IPOINT */
mbta_write_event_timepart_switch (id);

if ( L1L2_PTA)
{
    P4_uint32_t seed = get_random_number();
    pspkinfo->L1L2_PLACEMENT[psp_desc.api.current_cpu()] = seed;
    set_apbrandbank_seed(
        L1L2_Placement[psp_desc.api.current_cpu()],
        seed);

    flush_depending_policy();
}
```

7.3.4.1.2 Changing Hardware seed at Thread/process switch

On PikeOS task and thread switch the kernel call the psp_desc.api.tp_switch that implements the following algorithm when PTA mode is activated (L1L2_PTA is true):

```
if (task_uid == saved_api.last_task_uid[coreid]) {
   /* task is the same, so we are
    * switching thread inside same space domain */
   mbta_write_event_thread_switch (uid);
} else {
   /* task is NOT the same, so we are
```

```
     * switching from one to another space domain */
   mbta_write_event_task_switch (uid);
   if (L1L2_PTA) {
      /* need to flush L1 cache when changing space domain */
      __asm__ __volatile__("flush %g0"); /* flush cache L1 */
      __asm__ __volatile__("nop");
      __asm__ __volatile__("nop");
      __asm__ __volatile__("nop");
     /* Waiting for flush completion :
      * leon3 um : p26 - Cache flushing takes one cycle per cache
      * line,during which the IU will not be halted, but during
      * which the caches are disabled
      * FPGA v2.1: may flush the whole in one cycle
      */
   }
}
saved_api.last_task_uid[coreid] = task_uid;
```

Note:

The hardware cache randomization implementation requires flushing L2 cache when switching virtual address space because seed is specific to each virtual space. This correspond to a partition switch time or task switch time in PikeOS.

Changing seed in a scheduled schema allows synchronization of cache flush for any core at the time partition switch time reducing interference in between cores.

Changing seed in an asynchronous schema that is in pre-emptive mode, requires to change seed and flush caches on any task switch change from one virtual space to another. Then the number of flush increase drastically the switch overhead execution time also breaking down the expected advantage of hardware randomization.

*7.3.4.1.3 Changing **Hardware seed at Guest OS Thread/process switch***

On APEX (Guest OS) process switch the APEX guest OS call a specific PSP service that does the following:

```
mbta_write_event_guest_os_proc_switch(id);

if (TC_PROC && L1L2_PTA) {

   /* write seeds in device when activated */
   if (L1L2_PTA) {
      P4_uint32_t seed = get_random_number();
      pspkinfo->L1L2_PLACEMENT[psp_desc.api.current_cpu()] = seed;
      set_apbrandbank_seed(
            L1L2_Placement[psp_desc.api.current_cpu()],
            seed);
   }
   flush_depending_policy();

}
```

Note: SYSGO recommend not to change seed at Guest OS process switch because it will increase drastically the execution time of the switch due to requested cache flush and such a large overhead will break down any expected benefit of hardware cache randomization.

7.3.4.1.4 Shared cache and multi-core

PikeOS supply but the hardware need to flush L1 and L2 caches on changing seed has huge interference on other cores:

Let’s consider the case (allowed with in dynamic ticker mode) a time partition switch on core #1 for example is required half time than core #0:

- Core #0 MAF is global MAF
- Core #1 MAF is twice time the core #0 one

When you have TP switch on core #0 but not core #1 you have to change seed for core #0, not for core#1, but you also have to flush caches:

- L1 cache is local to core then there is no concern with L1 cache
- When L2 cache is shared but not partitioned, you need to flush the whole L2 cache raising large interference on core #1 (and possibly others)
- When L2 cache is partitioned as one way per core, the core we change seed is to be flushed but only for the way it is allocated. Other cores will not be impacted directly by the flush.

7.3.4.1.5 Seeds initialization and debug

In the frame of the PROXIMA projects it has been provided services to help contributors in implementing and debugging the hardware randomization.

For an industrial packaging those services should be removed and dependencies to hardware should be reduced and limited to the low-level of the kernel/firmware:

- *Hardware internal random generator*
- *No need for software to set seeds, but only a bit to trigger changing the seed for a partition or a core.*
- *Automatic flush on seed change*

7.3.4.1.6 Software Emulation support

In the frame of PROXIMA and to study impact of cache randomization on COTS hardware that does not provide any cache randomization, the PikeOS PSP has been modified to provide the following features:

- Pseudo random number generator. See section 6.3.2.1.1 PRNG support.
- PAK support. See section 6.3.1 PAK specific APIs. This includes memory allocator.

Note:

SYSGO discourage the use of exception handler and overlay callback to remap code/data at first execution since it will not be an industrial way to reach DAL A certification and security evaluation higher than EAL 4:

- It will recreate the same effect than the cache miss on first run with bigger impact; that means the first operational run of code will be dramatically increased thus providing a big concern on WCET. The only way is to remap at initialization time so that it will not be part of the WCET.

- Using an exception handling introduces a big security lack in the system. Moreover, when using the SPARC

- Requires to adapt the method for each guest OS. This may raise cost increase and confidentiality concern with guest OS from third part.

SYSGO industrial proposition is to have a system partition that will do the copy from elf file to memory with randomization mapping into the initialization scheduling schema (the SCHED_BOOT in PikeOSVMIT). The partition to be "randomized" shall be set to idle mode in the VMIT so that it will not start. Once the mapping is done for all the partition requesting SW randomization the system partition can start them as a whole as part of the next operational scheduling schema.

PikeOS v4.X (new PikeOS version currently released) allows implementation of software randomization component as a partition boot loader so that it allows:

- Guest OS independent support for cost reduction.
- Optional on per partition basis for mixed criticality support. Partition that does not want the feature will not be impacted.

The software randomization component implementation provided by the team in the scope of PROXIMA is not compliant with the industrial requirement provided by SYSGO above. This current implementation is guest OS dependent and specific to each personality and the application code under software randomization is no more compliant with highest level of certification requiring usage of following items along with embedded code:

- **Non-certifiable libc++ features,**
- **Non-certifiable mathlib**
- **Implement some cache miss equivalent on first access that emulates cache side effect with worst execution time impact than with hardware miss.**

7.4 PikeOS supports Hardware jitter reduction

For any of the features described in this section there is the SYSGO internal analysis about compliancy of the PROXIMA results to the PikeOS industrial constraints through colored notes:

- Green: when feature implementation is compliant with industrial need

- Italic orange: when feature partially implemented or implementation is partially compliant with the industrial need

- Bold red: when feature implementation is not compliant with industrial need

7.4.1 Bus arbitration

Shall be taken into account by the Timing Analysis and the measurement tools.

At some points the safety analysis may requires specific hardware configuration that can be implemented in a dedicated PSP for PikeOS.

7.4.2 Bounded instruction execution time

Shall be taken into account by the Timing Analysis and the measurement tools.

At some points the safety analysis may requires specific hardware configuration that can be implemented in a dedicated PSP for PikeOS.

7.4.3 PikeOS allows Core interferences characterization/verification

With PikeOS 3.4 some scheduling strategy have been implemented to get certified at highest criticality level.

In the frame of PROXIMA, SYSGO has developed a Time Composability Checker Tool mockup to start studying how to characterize multi core interference; see section "Time Composability checker tool – User Guide" in this document, and the document D3.12 will provide some first results.

PikeOS v4.X (new PikeOS version currently released) will take benefits of PROXIMA results as Time Composability checker tool will be reused:

- To go on studying multi-core interferences.
- Help on service characterization.
- To introduce service history
- To provide BIT checking plausibility of the behavior at start time to ensure the operating range.

7.5 PikeOS allows Software jitter reduction

For any of the features described in this section there is the SYSGO internal analysis about compliancy of the PROXIMA results to the PikeOS industrial constraints through colored notes:

- Green: when feature implementation is compliant with industrial need

- Italic orange: when feature partially implemented or implementation is partially compliant with the industrial need

- Bold red: when feature implementation is not compliant with industrial need

7.5.1 At API service

See discussion in section section "Time Composability checker tool – User Guide" in this document

7.5.2 Managing Interrupt support

Interrupt arrival enters kernel level code in the PSP when not masked, preempting any running activity except interrupt of higher priority. This preamble code will call a dedicated PikeOS kernel code that will notify any requesting resource of this event arrival that may preempt current user code at some synchronization point.

The first part of the treatment that is to register the event introduces latency/jitter in any code that can be interrupted; The Max interrupt latency is to be taken into account in any service that can be interrupted and in the application as it will increase the elapsed time. Using the above technique to reduce cache history impact will reduce the variability so the jitter on this interrupt latency. But interrupt latency is still something to be avoided since it will introduce variability on the synchronization time frame between partitions through messages.

The second part may preempt the current user activity depending on the chosen configuration. This is only matter of well-designed application to reach the requirement and IMA design with periodic scheduling based schema allows determined behavior.

PikeOS allows interrupt jitter and latency reduction using following features:

- Use the dynamic timer mode instead of the periodic timer (PSP TAGs). The dynamic will generate timer interrupt only when required (the periodic mode will generate a periodic ticker interrupt on any core that will introduce jitter and latency on any service and application code).
- Moving any critical partition (at system design and integration level) on cores there will be no partition subscribing to any external interrupt so that a critical partition cannot be interrupted.

7.5.3 Using IO partition

PikeOS provides the necessary means to implement IO partition. One existing example is the AFDX IO stack for avionics. The idea is to connect queuing and sampling port via the IO stack that will dispatch messages running into a separate core so that the message copy is in charge of the IO stack rather than the service caller (port read or write).

PikeOS is provided for general purpose usage and the standard communication port provides blocking IO feature.

In the frame of PROXIMA we demonstrate PikeOS can provide non-blocking IO port with constant execution time service for read and write.

But the standard specification for industry like ARINC653 does not provide the well suited API to provide this feature.

In the frame of PROXIMA, providing the non-blocking IO port required a modified implementation of any guest OS. Thus it has been done for only APEX and Native personality adding dedicated API that cannot be provided as product line basis at industry level.

PikeOS v4.X (new PikeOS version currently released) takes benefits of PROXIMA results providing:

- Support for optional non-blocking IO port at PikeOS Native level that can benefits to any guest OS.
- Kernel level driver allowing IO port with Time Composability that can be more easily reached when used in conjunction with other features to reduce jitter and latency (cache jitter reduction).

8 MBPTA – PAK use cases demos – User Guide

8.1 Overview

In this user guide we will show how to configure a project which runs

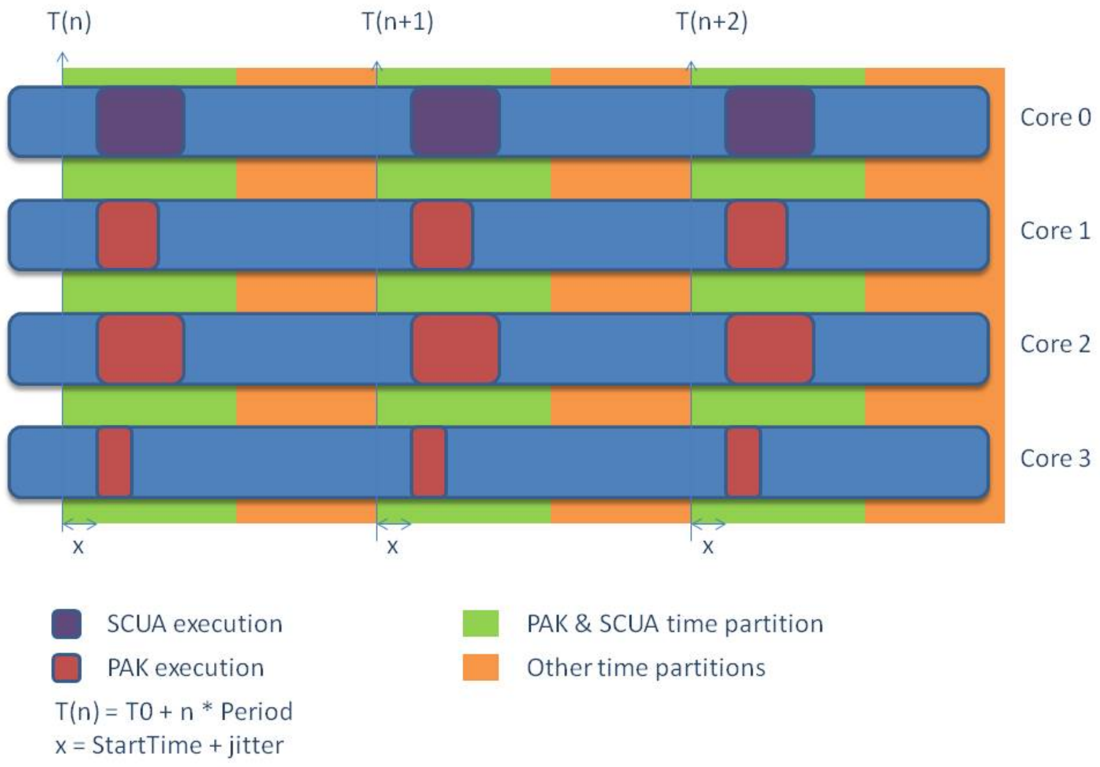
- PROXIMA PAK in a PikeOS partition on core 0 instead of the scua.
- PROXIMA PAK in a PikeOS partition on core 1, 2 and 3.

The target can be either qemu, leon3 PROXIMA FPGA or P4080 board.

This first example is only help specifying the PAK and instrumentation usage. Then we will also provide scua using the time partitioning.

Here PAKs task start at a time T0 and is run with a period T (these values are configured in the integration project and may differ per core). By default, the periods and the starting times are the same for all PAKs. The application on core 0 (SCUA) also uses the PikeOS native API to be able to run a PAK.

The following diagram shows the execution of different PAKs on each core with the defaults configuration and with a PAK as the SCUA.



8.2 Integration project

8.2.1 Create a new integration project using project demo

First we will create the integration project using the given integration project demo. This demo contains 4 different PikeOS partitions which can run PAK applications. Clone the Integration Project using these values:

Project type: *PikeOS Integration Project*

Project name: *proxima-pak.int*

Template: *proxima-pak*

For the leon3 PROXIMA FPGA:

Board name: *leon3-proxima-smp*

Boot strategy: *elf*

For the P4080 board:

Board name: *p4080ds*

Boot strategy: *uboot-dtb*

For qemu (sparc v8):

Board name: *qemu-sparc*

Boot strategy: *qemu*

8.2.2 Customize the integration project (optional)

8.2.2.1 Configure the memory requirements

Several parameters can be customized in this integration project. Note that the customizations described in this chapter are optional and the integration project is already configured for the standard experimentations.

The most important in this demo is the configuration of the memory requirements used by the PAKs for dynamic allocations. To configure the memory requirement of a partition, you must perform the following steps:

- Open proxima-pak.int → vmit.xml
- Expend [Root->Default] → [PartitionTable] → [Partition->core0]

- Select [MemoryRequirement->RAM_POOL]

In the memory requirement attributes you can change the cache mode:

VM_MEM_CACHE_CB	use the copy-back cache strategy
VM_MEM_CACHE_WT	use the write-through cache strategy
VM_MEM_CACHE_INHIBIT	do not use the cache memory

8.2.2.2 Configure the PTA mode

FPGA board has been modified to supply random placement/replacement strategy. This feature is optional and shall be enabled by following settings:

- Set the SLIDE_SW2 board slide switch in position ON (see proxima_qsg_1.6.pdf section 4.2. Probabilistic Timing Analysis Features).
- By default, PikeOS won't enable the L2 Cache. You need to activate it using GRMON before executing the image with the following command:
  - grmon2> l2cache enable
- To configure the L2 cache replacement policy, select the desired value (see section 3 above) with the *PikeOS Project Configurator* in "Board Settings > L2 Policy".
- Optionally set the Flag attributes VM_SCF_INVAL_ICACHE and VM_SCF_INVAL_DCACHE to the first time partition so that the values for fixed seeds of APBRANDBANK will be changed on each new time window (at the time partition switch time).

8.2.2.3 Configure PAK execution time on each core

It is possible to configure the execution time of the PAK on each core. The properties of the PAKs are defined in "templates/pikeos.rbx.inc". These are the default properties:

```
<properties>
  <prop_dir name="PAK-core0">
    <prop_uint64 name="StartTime" data="100000000"/>
    <prop_uint64 name="Period" data="1000000000"/>
  </prop_dir>
  <prop_dir name="PAK-core1">
    <prop_uint64 name="StartTime" data="100000000"/>
    <prop_uint64 name="Period" data="1000000000"/>
  </prop_dir>
  <prop_dir name="PAK-core2">
    <prop_uint64 name="StartTime" data="100000000"/>
    <prop_uint64 name="Period" data="1000000000"/>
  </prop_dir>
  <prop_dir name="PAK-core3">
    <prop_uint64 name="StartTime" data="100000000"/>
    <prop_uint64 name="Period" data="1000000000"/>
  </prop_dir>
</properties>
```

All the PAKs will start 100 ms after the time switch (for synchronization purpose) and with a period of 1 second (“StartTime” and “Period” in nanoseconds). See the previous diagram for more details about these parameters. You can modify these values or create new properties.

8.2.2.4 Run 1 PAK on 3 cores

In some experiments described in CaseStudies-Experiments-m18.docx the core 1,2 and 3 are used together to a load a single PAK. In order to do that, you can configure the CPU mask of the second partition to match the 2 other cores.

- Open proxima-pak-fpga.int → vmit.xml
- Expend [Root->Default] → [PartitionTable]
- Select [Partition->core1]
- Set *CpuMask* to 0x000000000000000E

Then you only need set the 2 other partitions to IDLE

- Select [Partition->core2]
- Set *PartitionStartupMode* to VM_PART_MODE_IDLE
- Select [Partition->core3]
- Set *PartitionStartupMode* to VM_PART_MODE_IDLE

8.2.2.5 Execute the PikeOS kernel in a no cache area RAM

It is possible to run the PikeOS kernel into a no cache area RAM for this demo project. The purpose of this is to reduce the impact on user cache history from system call; there is 2 ways to get it on FPGA and only the 1) for the P4080:

1) Set PSP TAG 0x308 to 1 before make boot of the integration project.

2) The ELF file at */opt/pikeos-3.4/share/PSPs/leon3-proxima-smp/objects/psp-kernel-smp-tracesys-sram* must be loaded using GRMON after you already loaded the full PikeOS image (using the integration project from this demo). There are now 2 kernels which can be used when executing the PikeOS image:

- The kernel from the PikeOS image at 0x40020000 (SDRAM)
- The newly loaded kernel at 0x3000000 (SRAM)

  After loading the 2 binaries you simply need to run the image. The PikeOS entry point (in SDRAM or SRAM) is set each time you load an elf image which means you only need to reload the PikeOS image if you want to run the kernel in the SDRAM again.

8.3 *Application project*

8.3.1 Create a new application project using project demo

Now we need to generate the PAK applications which will run inside the partitions using the given application project demo. Clone a new PikeOS C Project using these values:

Project type:*PikeOS C*

Project name:*proxima-pak.app*

Template:*proxima-pak*

For the leon3 PROXIMA FPGA and qemu (sparc v8):

Architecture:*sparc*

Processor type:*v8*

For the P4080 board:

Architecture:*ppc*

Processor type:*e500mc*

8.3.2 Change the source code to use a custom PAK (optional)

This demo application project implements a dummy PAK which can be replaced. To update the PAK functions, you must perform the following steps:

- Open proxima-pak.app → src → core → pak.c
- Update functions *init_pak* and *run_pak*

8.3.3 Build the project

In order to build the PikeOS ROM image you need to install the PAKs in the integration project. The integration project needs 3 PikeOS applications (pak1, pak2 and pak3) to run in the cores 2, 3 and 4. The “install” rule will move 3 copies of the same application in the integration project.

Perform the following steps to build the project:

- Open proxima-pak-fpga.app → project.xml.conf
- Set your integration project to “*proxima-pak.int”*

- proxima-pak.app → Make install
- proxima-pak.int → Make boot

8.3.4 Load the SCUA using PIRSH

Boot the ROM Image (see platform manual for more details about the boot configuration). You should see this in the console:

```
core 1: init PAK...

core 2: init PAK...

core 3: init PAK...

core 1: exec time = 23093990ns

core 2: exec time = 23315640ns

core 3: exec time = 23098800ns

[…]
```

The core0 partition is not started at boot time because it needs a binary (the SCUA) which should be copied into the shared memory using PIRSH. In this integration project, the partition needs a PikeOS application in order to load a PAK as the SCUA. Open PIRSH and send the following commands to load the SCUA and start the partition:

```
PIRSH> set localfile $SCUA_PATH

PIRSH> set targetfile shm:scua

PIRSH> put

PIRSH> start 2
```

8.4 Evaluation of the PAK demo on FPGA PROXIMA

Preliminary tests have been done in order to roughly evaluate the impact of each parameter which may influence the execution time of the PAKs. In these tests, we use the PROXIMA FPGA board and run the PAK from the demo on cores 1/2/3. The tests have been done on the FPGA BSP V1.2.1 delivery. The values are the PAK execution times with the corresponding configuration. The following results are only presented here to give an idea of the current performances but it must not be taken as reference.

	Kernel in SDRAM	Kernel in scratchpad RAM
L2-cache disabled	23ms	22ms

L2-cache with LRU replacement policy	9ms	9ms
L2-cache with RND replacement policy	9ms	9ms
L2-cache with IDX replacement policy	9ms	9ms

With these results we can see that the L2-cache greatly improves the execution time of the PAK but the other parameters have less effect for this specific PAK.

9 Time Composability checker tool – User Guide

9.1 Overview

The Time Composability Checker tool to be applied as defined in document D3.12 section 2 is implemented using 2 demo projects:

- The integration demo project to be cloned from demos-integration/proxima-ibit
- The tool demo project to be cloned from demos-pikeos/proxima-ibit. This application project includes the following applications:
    - tcc the time composability tool checker to be run on core 0 as a test suite server
    - iload a load generator application to be run on other cores for interference generation.
    - vm a default scua as a template for building other test cases
    - vmtc a default Time Composable scua as a template for building other test cases

And one demo project of the software under test to be cloned from the respective personality specific demo project:

- demos-{personality}/proxima-ibit-{scua}

In this document we use the following naming convention for the projects you get from clone operation:

- proxima-ibit-{PIKEOS_VERSION}-{BOARD}{[pta]}.int.
- proxima-ibit-{PIKEOS_VERSION}-{BOARD}{[pta]}.app.
- proxima-ibit-{scua}-{PIKEOS_VERSION}-{BOARD}{[pta]}.app.

{PIKEOS_VERSION} is p34 for PikeOS 3.4

As defined in document D3.12 section 2 this naming rule allows the following scua configurations:

scua	PikeOS version	Board	Personality under test
apex	P34	fpga	APEX standard personality without any improvement
apextc	P34	fpga	APEX TC personality with M18 improvement
vm	P34	fpga	PikeOS native personality without any improvement
vmtc	P34	fpga	PikeOS native personality with TC improvement
apex	P34	cots	APEX standard personality without any improvement
apextc	P34	cots	APEX TC personality with TC improvement

vm	P34	cots	PikeOS native personality without any improvement
vmtc	P34	cots	PikeOS native personality with TC improvement

And also various hardware configurations provided by {BOARD}{[pta]} option:

- FPGA with PTA mode ON
- FPGA with PTA mode OFF
- COTS with PTA (software randomization) ON
- COTS without PTA (no software randomization, standard mode)

The services under characterization are mainly the followings:

- SEND_QUEUING_MESSAGE
- WRITE_SAMPLING_MESSAGE
- RECEIVE_QUEUING_MESSAGE
- READ_SAMPLING_MESSAGE

The crossing time of each functions is measured and displayed.

The following events are also displayed: the total instruction, the data and instruction cache miss (L1, L2), and the data and instruction tlb miss.

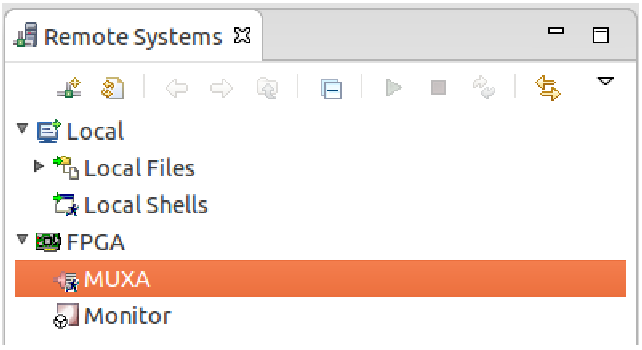
These events are counted by the launch of the performance counters available on the board through an abstraction level:

- L3STAT for LEON3-SPARC processor with the 1.8 FPGA version or higher.
- Performance Monitor Registers (PMRs) for the P4080

The ibit-ttc application is the master automaton for the test suite, sending periodically a command line from the first time slot of a MAjor Frame (see Figure 1). The next time slot is allocated to both SCUA and iload on each core. They retrieve the command line and apply the action requested by ibit-tcc and the MAF ends. When the scua action specified in the command line ends, the scua signal the completion to the ibit-ttc so that it can send the next one on next MAF. This allows an action to run over several MAFs when required. And most of the time the print operation on action completion does.

The print operation computes the results and displays them in one of two ways:

- On the serial line when both muxa:/mbta and muxa:/* is not defined for the tcc partition
- Through the muxa:/mbta channel when properly is defined so that ibit-ttc can open it and write to it. In this case you need first the MUXA process running on the host either in the CODEO or from command line (fundamentals.pdf §4.2).



The following diagram shows the periodic scheduling schema with the Time Composability Checker (ibit-tcc) running before the SCUA and the iloads.

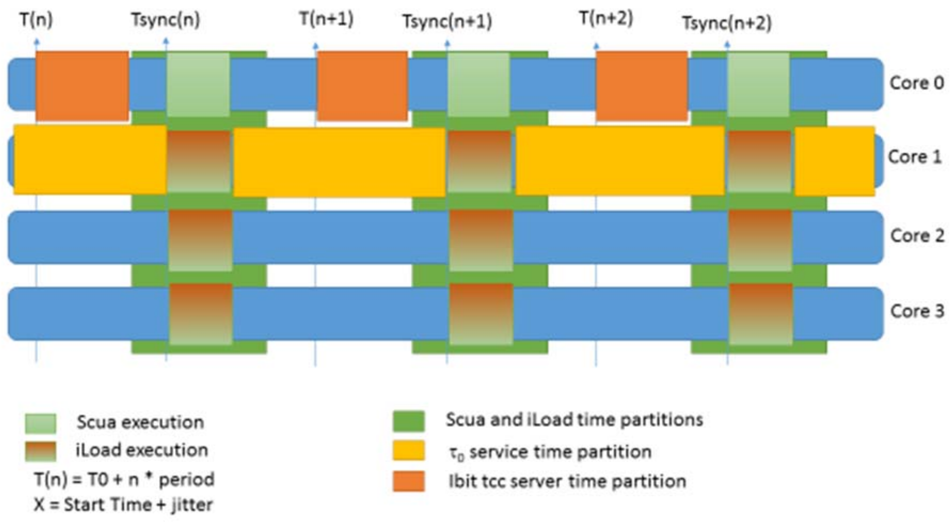


Figure 1Time Composability checker tool time frame

T(n) is the release point that is the time when starting MAF.

Tsync(n) is the time scua and iloads shall start synchronously the test item execution.

Tsync(n) = T(n) + K, K is configurable to let enough time for client loop overhead.

The iload action shall be started just before the scua test starts and shall end just after. So calibration of iload is not an easy job and may take some adjustment.

Note that when the goal is to reveal interference there is no requirement to have the scua running inside the window time of the iload. The need is to have iload running enough time during over scua so that it shall generate the maximal measurable interference value to over bound the scua ET.

For probabilistic approach the way is to start scua with a variable jitter around T(n) so that iload can be applied with some calibrated starting point distribution over scua.

9.2 Integration projects

First we will create the integration project using the integration project demo.

Clone the integration project using these values:

Project type: *PikeOS Integration Project*

Project name: *{SCUADEMO}.int*

Template: *proxima-ibit*

Board name: *leon3-proxima-smp*

Boot strategy: *elf*

9.2.1 Select appropriate board settings

On the FPGA only board you can select the PTA mode by setting ‘L2 policy’ to RND; following values for the FPGA are available for debug purpose also:

- *LRU* (LRU policy is chosen for non PTA mode),
- *RND (Random policy is chosen for PTA compliant mode),*
- *L1PTA (Random policy is chosen for PTA compliant mode but L2 cache OFF),*
- *OFF* (cache L2 is disabled in non PTA mode).

On both FPGA and P4080 you can set ‘KERNEL cache policy’ to TC_no_cache so that kernel level will run without any impact on user cache history, a big step towards Time Composable behavior.

9.2.2 Avoid service interference

Service run applications that can preempt the scua, and some service like Ethernet driver may generate a lot of interrupts. All of this is not suitable with Time Composability and can be enforced to run in a separate core removing corresponding interferences (theoretically, if hardware does not raise side effects…).

In the demo project the scua run in core 1 (CpuMask #2) as default and the service is set as default to run core 0 (CpuMask #1).

9.3 Application project

Clone a new APEX Project using these values:

Project type:*PikeOS APEX project*

Project name:*{SCUADEMO}.app*

Template:*{SCUADEMO}*

Architecture:*sparc*

Processor type:v8

The application project creates four binaries: vm, vmtc, tcc and iload. SCUA is either vm or vmtc the Time Composable version of vm.

- In order to build the PikeOS ROM image you need to install these 4 binaries in the integration project.
- Perform the following steps to build the project:
- Open *{SCUADEMO}.app* → project.xml.conf
- Set your integration project to "*{SCUADEMO}.int*"
- *{SCUADEMO}*.app → Make install or Make install_tc
- *{SCUADEMO}.int* → Make boot

10 Acronyms and Abbreviations

BSP	Board Support Package
COTS	Commercial off-the-shelf
MBTA	Measurement Based Timing Analysis
MBPTA	Measurement Based Probabilistic Timing Analysis
OS	Operating System
PTA	Probabilistic Timing Analysis
RTOS	Real-Time Operating System
SCUA	System Component Under Analysis
PAK	PROXIMA Application Kernel
APB	Advanced Peripheral Bus