HARPA

Harnessing Performance Variability

Project ref. FP7-612069
Call ref. FP7-ICT-2013-10
Activity ICT-10-3.4

HARPA-OS Engine, Final Release

Giuseppe Massari,
Simone Libutti,
William Fornaciari

Politecnico di Milano,
ITALY

Report Number :	D.1.4
Version:	Final
Date:	November, 30th 2016

Revisions List

Date	Version	Author(s)	Description
Nov, 15th 2016	0.1	Giuseppe Massari	First version.
Nov, 18th 2016	0.2	Giuseppe Massari	Extended sections about download, compile, configure and execute.
Nov, 19th 2016	0.3	Giuseppe Massari Simone Libutti	Extended the configuration of the target platform and the “Testing the execution section”.
Nov, 21th 2016	0.4	Giuseppe Massari Simone Libutti	First version of “Introduction”. Extended the description of the Kconfig menus.
Nov, 25th 2016	0.5	Giuseppe Massari Simone Libutti	“Introduction” finalization. References section finalization.
Nov, 27th 2016	0.6	Giuseppe Massari	Reviewed after Francy Catthoor feedback.
Nov, 28th 2016	0.7	Giuseppe Massari	Reviewed after Henesis/CAMLIN feedback. Added conclusions.

Table of Contents

Tables of Figures

1 Introduction

In this deliverable we present the release of the HARPA-OS engine. The document is structured as follows. In the current section (Section 1) some release notes are reported, along with some information about the open-source project involving the HARPA-OS engine. Section 2 shows how to download the source code (which is publicly available), while in Section 4 we describe the steps required to compile and install HARPA-OS. Section 5 goes deeper in the compilation process, showing how to selectively enable or disable features. In Section 6 we show how test the HARPA-OS on one of the supported hardware target platforms, by running a simple run-time manageable sample application. Finally, Section 7 provides some final considerations about HARPA-OS.

1.1 An open-source project

HARPA-OS is the run-time resource management framework developed in the frame of the Barbeque RTRM Open-Source Project (BOSP), carried out by the Politecnico di Milano (http://bosp.dei.polimi.it). The project includes also run-time manageable sample applications, the porting of some benchmarks, along with additional tools and libraries.

This open-source project aims at going beyond the horizon of the HARPA EU FP7 project, possibly collecting contributions from several developers.

To this purpose, users and developers can join two mailing lists. A first one for updates or help requests: https://groups.google.com/d/forum/bosp and a second one for providing development contributions: https://groups.google.com/d/forum/bosp-devel.

HARPA-OS is the development of the BarbequeRTRM in the context of the HARPA project software stack, which has meant extend the capabilities of the BarbequeRTRM by introducing the support of new hardware platforms and new resource management policies to address the challenges described in the embedded and the High-Performance Computing (HPC) use cases (see deliverable D.5.1).

1.2 Release notes

In Fig 1 we show a simplified view of the updated HARPA-OS architecture. In this release we introduced changes in all the layers of the framework.

The HARPA-OS *Application Run-time Library (RTLib)* has been developed with an additional monitoring support, thanks to which the resource manager can observe how the application actually exploits the assigned resources. Moreover the set of functions provided by the library has been extended with a function to retrieve explicit information about the set of assigned resources and a new function to specify a performance goal.

The HARPA-OS *daemon,* which includes several components, has been extended with modules in charge of monitoring the power status of the platform.

On the resource allocation / scheduling policy side, two novel policies have been implemented (*PerDeTemp* and *Tempura*) both exploiting this power monitoring support in order to drive the

decisional process also according to power and thermal constraints. These policies, introduced in D.1.3 and summarized in Section 5.2 replace the default policy (*YaMS*) of the initial HARPA-OS release.

The *PlatformProxy* component, which implements the internal abstraction layer over the underlying platform, has been re-designed and extended with the support of the HARPA reference hardware platforms. In addition, the new *hardware platform description* format supports the possibility of registering remote computational resources, i.e., allows the HARPA-OS to get visibility over the resources of multiple nodes of a distributed system. In this regard, the *AgentProxy* plugin has been introduced as a new Remote Procedure Call (RPC) based communication channel, in order to enable the interaction among multiple instances of HARPA-OS (one per node). More developments are expected in the future to improve the capabilities of HARPA-OS as a distributed run-time resource manager.

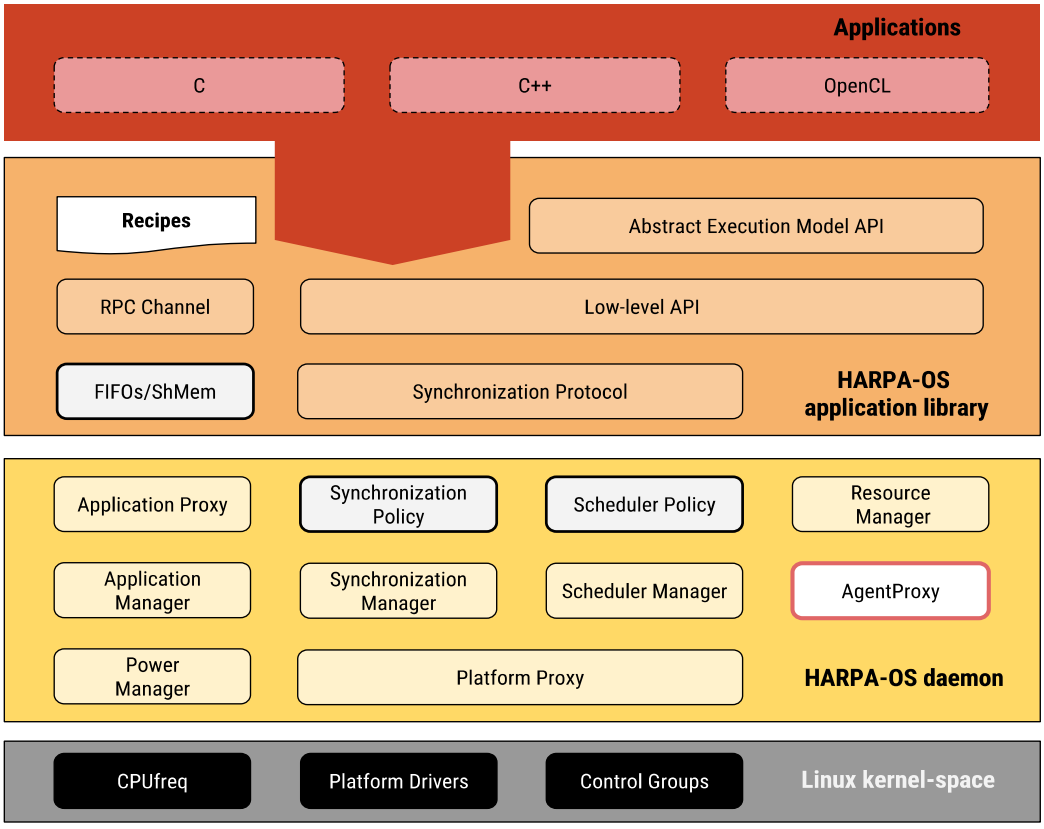

Fig 1: HARPA-OS Architecture

2 Requirements

2.1 Hardware

In order to successfully compile HARPA-OS it is strongly recomended to have a “host” machine running equipped with a multi-core CPU (Intel or AMD quad-core), with 8 GB of memory, and at least 2 GB of available space on disk. In case of cross-compilation for an external ARM based “target” platform, the host machine must also include a network interface.

2.2 Software

Both the target system (for HARPA-OS execution) and the host system (for HARPA-OS compilation) must run a Linux distribution as operating system. We successfully tested HARPA-OS on Ubuntu (from 14.04 to 16.04), Mint (from 16 to 18) and ArchLinux.

Since the development of the framework has been carried out using GIT[1] as control version tool, a mandatory pre-requisite is to have it installed in the host system, i.e. the system on which the build process of framework will be launched. GIT is a popular tool whose packages can be download from the official repositories of the most common Linux distributions, therefore this can be considered a trivial step.

We list here below all the tools that are required in order to properly build the framework:

- Essential building tools (“build-essential”)
- CMake (>= v2.6)
- GCC/G++ and libstdc++-v3 (both >= v4.7)
- Autotools (i.e. autoconf, automake and libtoolize)
- Doxygen (>= v1.8)
- Texinfo

Assuming the host system is running a Debian based distribution using **apt-get** as package manager, a command line to install the aforementioned tools is the following:

```
$ sudo apt-get install build-essential autoconf automake libtool
gperf cmake git-core doxygen graphviz texinfo ncurses-dev byacc
flex bison moreutils gawk curl aterm hwlocbc dialog
```

Once this set of packages and libraries have been installed into the host system it is possible to proceed with the download of the source code.

1 http://git-scm.com

3 Download the source code

To download the HARPA-OS source code we must locally “clone” the GIT remote repository into the host system as follows:

```
$ git clone https://bitbucket.org/bosp/bosp.git BOSP
```

This command does not perform a real download of the sources. In fact it simply downloads a few initialization scripts that will then perform the real work. BOSP is indeed a collection of “subprojects” required to properly use HARPA-OS, whose changes are tracked by using the **submodule** utility of GIT.

The **bosp** script available under he “BOSP” directory wraps the GIT submodule syntax. It is used to download the sources of all the subprojects and then getting updates. Assuming this is our first installation we need to launch the following command:

```
[BOSP] $ ./bosp setup
```

Then, in order to switch the current project branch to the latest HARPA-OS release, the following sequence of commands must be launched:

```
[BOSP] $ ./bosp update-devel
[BOSP] $ git checkout origin/harpa-os
```

This sequence will align the version of all the subprojects to the curren HARPA-OS release version. Once this step has been successfully completed, we can pass to the compilation of the framework.

4 Compile and install the framework

The BOSP project, along with the HARPA-OS framework is a complex software project including a wide and growing number of modules, features, functionalities and supports.

Therefore, in order to manage such a complexity making the compilation process flexible and effective, according to the final framework configuration we aim at build we need a tool that allows the user to enable and compile only the required feature. The tool we chose is **Kconfig**, the same used for the configuration of the Linux kernel compilation.

The configuration generated is then processed by a set of *makefiles* and **Cmake** files, where are listed in the source files, compiler options, the definition of the dependencies, the paths for retrieving the libraries and the installation paths. The user does not need to care about these files,

unless his purpose is to contribute to the development of the framework.

Concerning the compiler, we successfully tested several versions of the GNU C/C++ compiler (**gcc/g++**) spanning from version 4.7.2 to version 6.0 when the target architecture is a x86 machine. In case of ARM platform instead, the toolchain employed is the GNU C/C++ version 4.8 provided by Linaro [2].

Once the configuration has been generated, the framework – including HARPA-OS – the applications, the benchmarks and eventually some external tools can be compiled by simply typing:

```
[BOSP] $ make
```

This command will also install the HARPA-OS in the output directory. By default this directory is set to "BOSP/out" but its value can be changed by acting on the *Custom sysroot* configuration option described in 5.3.

In the next section we focus on the configuration of the compilation process, by describing the options provided through the Kconfig interface.

5 Configure the compilation

Kconfig provides different user interfaces, from a command-line based one to several options of graphical interfaces (GUI). In this deliverable we will focus on the most common interface based on the *ncurses* library, which builds a lightweight GUI that allows the user to interact by using the arrow keys, the spacebar, the ESC and return key. This graphical user interface must be launched from the BOSP directory, as follows:

```
[BOSP] $ make menuconfig
```

The outcome, shown in Fig 2, is a main menu including the following set of internal sub-menus.

2 https://www.linaro.org/

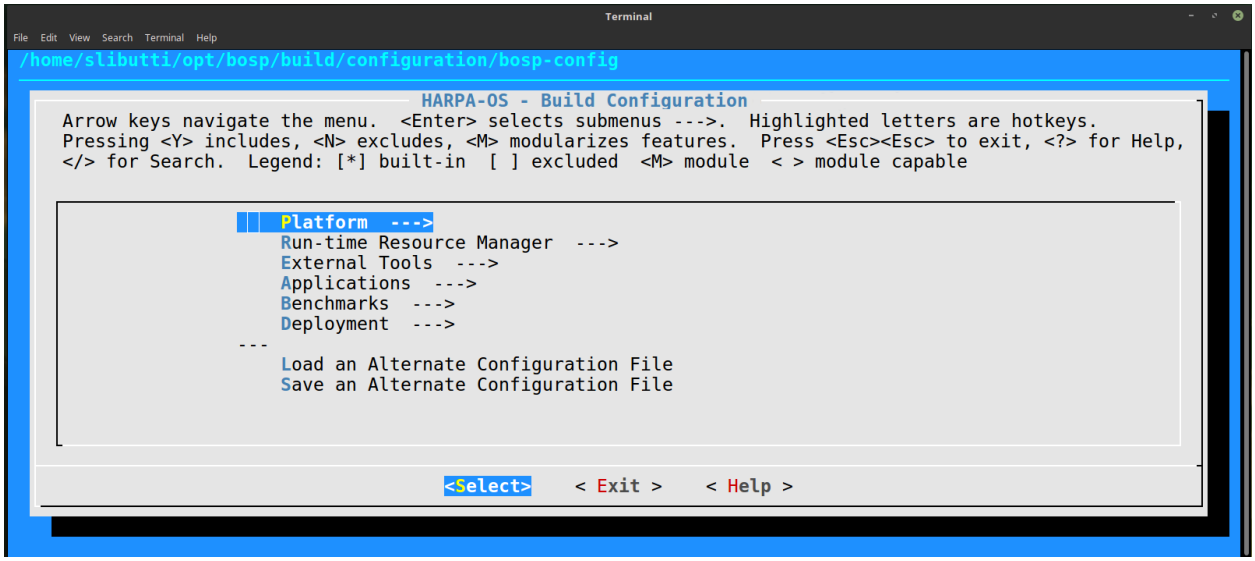

Fig 2: HARPA-OS menuconfig: main section

- **Platform**: selection and configuration of the target hardware platform;
- **Run-time Resource Manage**r: options to configure the compilation of the HARPA-OS;
- **External tools**: tools and libraries required to build and/or use the framework;
- **Applications**: set of integrated managed sample applications;
- **Benchmarks**: set of benchmark applications (e.g. from PARSEC suite) for experimental purposes;
- **Deployment**: configuration of the installation/deployment.

The user can navigate in each sub-menu enabling or disabling features as it is shown more in detail in the next sub-sections of the document. Once the configuration is ready, the user can exit by pressing ESC or by selecting the “Exit” option in the main menu and then selecting “Yes” in the dialog box asking if the configuration must be saved or not.

Before showing how to launch the compilation process of the framework, in the following sub-sections we provide further details about the content of the sub-menus that are worth to fully understand for the correct compilation of HARPA-OS.

In the following description we skip the *Applications* and *Benchmarks* menu, which simply includes a set of applications to build or not, and the *Exernal tools.* Concerning this last sub-menu, it is worth to say that most of the options are automatically selected as a dependency of the features enabled in the *Platform* and *Run-time Resource Manager* sub-menus.

5.1 Hardware platform setup

HARPA-OS can be compiled for both Linux and Linux-ARM targets. Selecting the target enables additional configuration options (see Fig 3): in the case of Linux targets, it is possible to specify

whether the target machine is based on the x86 or the x86_64 instruction set; in the case of Linux-ARM targets, it is instead possible to select the target board, which enables target-specific options if any, and the cross-compile toolchain that will be used during the compilation process.

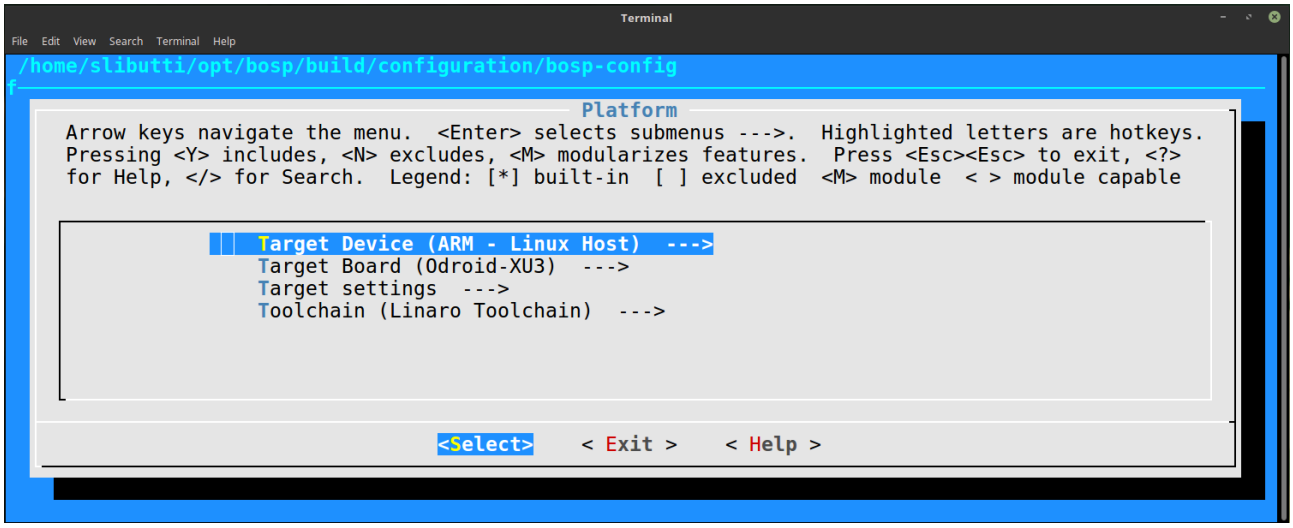

Fig 3: Target specific configuration for the Odroid-XU3 development board

HARPA reference platforms

HARPA use-cases exploit two embedded platforms and a cluster of HPC nodes, for a total of three specific targets. The development required to support these three target systems has built a common baseline such that porting the framework on new hardware platforms is now a quite straightforward activity.

Focusing our attention on the HARPA targets, the first one is the ODROID XU-3 embedded development board by Hardkernel [3]. It is based on the Samsung Exynos 5422 System-on-Chip, which features an ARM big.LITTLE octa-core CPU (Fig 4). The processor is composed by a high-performance cluster (Cortex A15 quad-core with 2GHz maximum frequency), and a low-power one (Cortex A7 quad-core with 1.4GHz maximum frequency), and works in a Heterogeneous Multi-Processing (HMP) configuration. Being the two clusters based on the same Instruction Set Architecture, threads can freely migrate between clusters, thus allowing the Linux HMP scheduler to dynamically switch between low-power and performance-oriented execution. The board also features on-chip sensors that can be used to sample the power consumption of: the clusters (one sensor per each cluster); the embedded GPU; and the memory. Furthermore, an external power meter, with a sampling rate of 10 Hz, allows the overall system power consumption to be monitored.

3 http://www.hardkernel.com/main/products/prdt_info.php?g_code=g140448267127

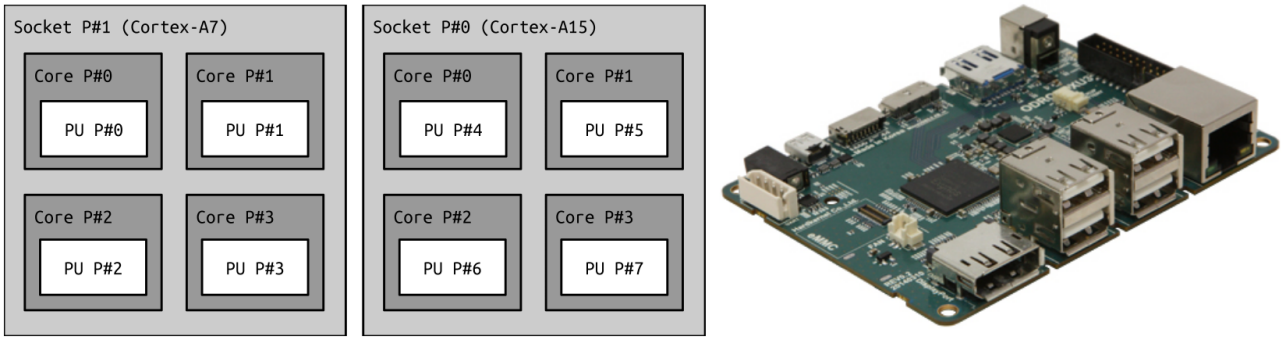

Fig 4: ODROID XU-3 development board featuring the Samsung Exynos 5422 SoC

The second embedded platform is the i.MX 6Quad Smart Application Blueprint for Rapid Engineering (SABRE) development board produced by FreeScale (now NXP) [4]; it features an ARM quad-core CPU with a maximum frequency of 1GHz (Fig 5).

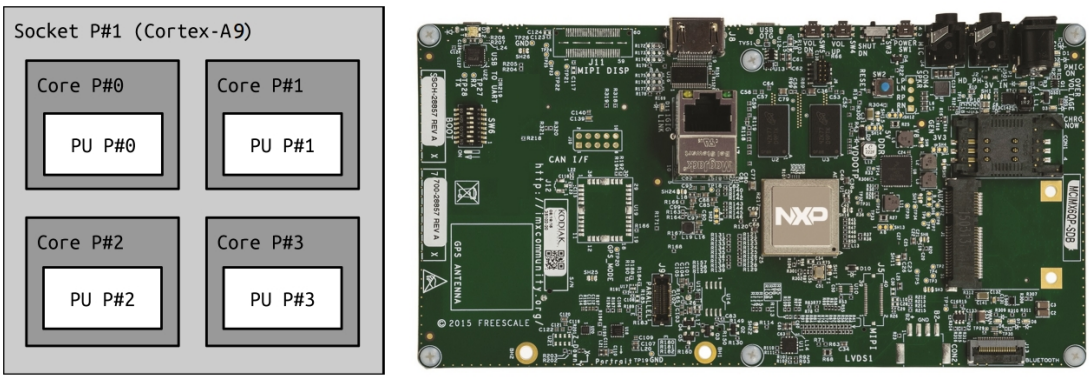

Fig 5: i.MX 6Quad SABRE development board

The HPC cluster is composed by 18 nodes, which are connected via InfiniBand. Each node is a NUMA platform featuring two Intel(R) Xeon(R) E5-2665 CPU, with 8 cores (16 threads due to SMT) and a maximum frequency of 2.4GHz (Fig 6).

4 http://www.nxp.com/products/software-and-tools/hardware-development-tools/sabre-development-system/sabre-board-for-smart-devices-based-on-the-i.mx-6quad-applications-processors:RD-IMX6Q-SABRE

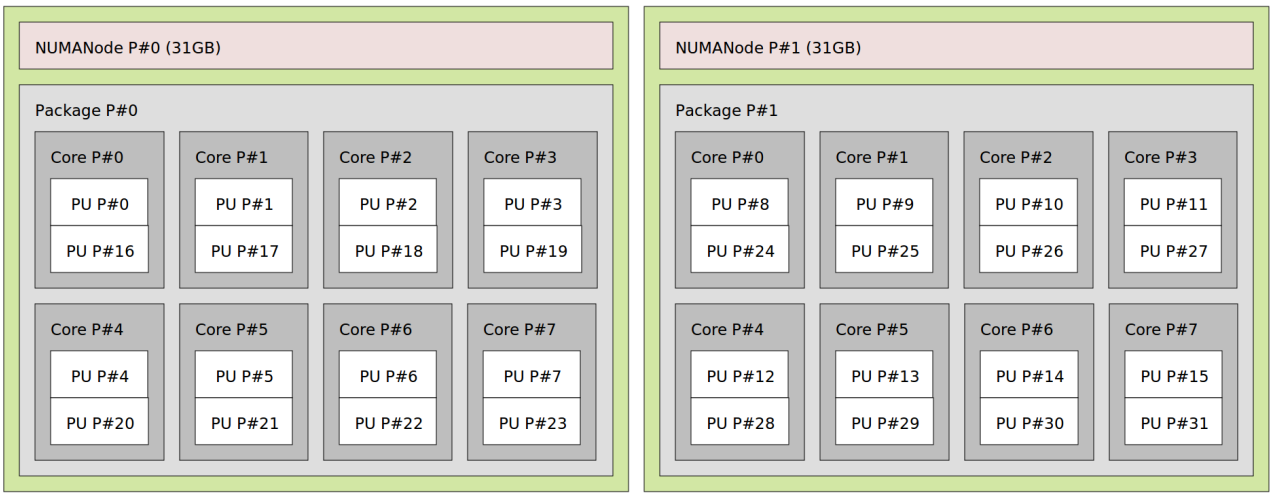

Fig 6: A single node of the HPC cluster, featuring 2 Xeon E5-2665 CPUs in NUMA configuration. The cluster is composed by 18 nodes, connnected via Ethernet and InfiniBand.

5.2 Run-time Resource Manager

This **menuconfig** section collects the resource manager configuration options. It is composed by four main subsections, namely *Platform setup*, *Resource allocation policies*, *Run-time library* and *Advanced options* (see Fig 7).

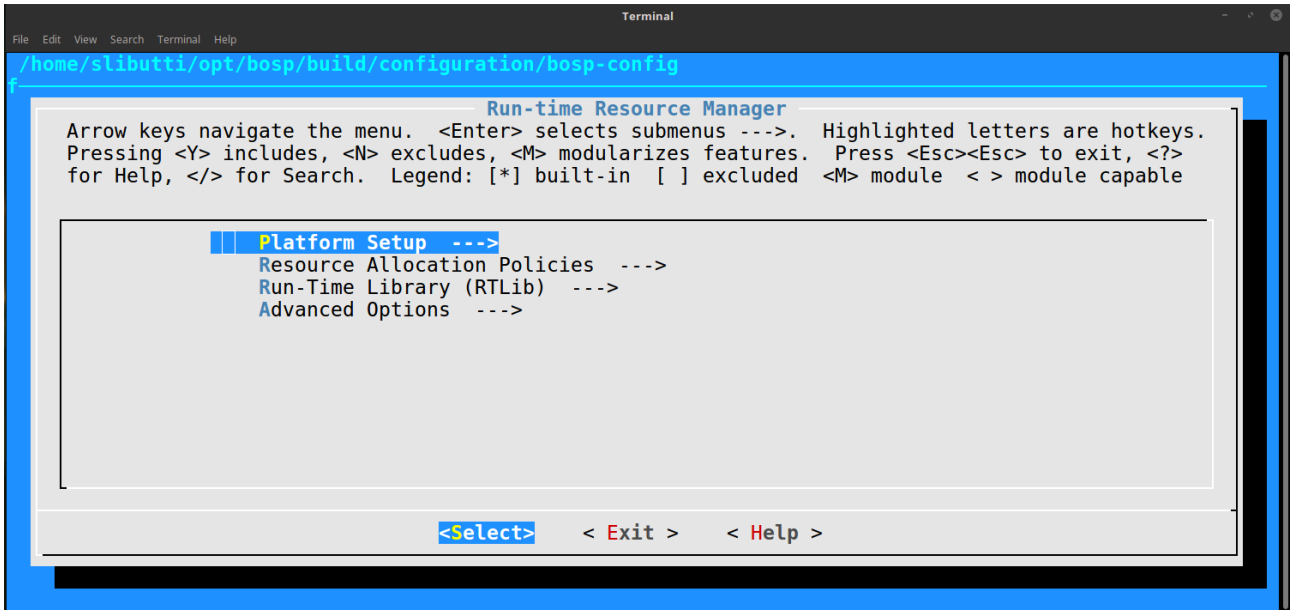

Fig 7: HARPA-OS menuconfig: Run-time Resource Manager configuration screen

The *Platform setup* subsection allows the system administrator to configure how the HARPA-OS

interacts with the managed platforms. The configuration options, most of which have their own subsections, are summarized in the table below:

Config option	Description
Platform loader	Parsing options for the platform description file (see *Platform Description Generation* on page 18)
Simulated mode	Build the HARPA-OS in simulation mode, i.e., with the possibility to simulate a custom hardware (custom number of sockets and cores) and with Linux Control Groups support disabled.
Heterogeneous systems	OpenCL support in case of CPU-GPU systems
Linux resource management	Linux Control Groups configuration, e.g., Cgroups mount point and possibility to disable unused Cgroup subsystems.
Power Management	Support for CPU and battery power management
Distributed systems	Support for centralized distributed systems, i.e., multiple nodes managed by a single master node.

The *Resource allocation policies* subsection allows the system administrator to choose which resource allocation policy will be used by the HARPA-OS. HARPA-OS users can implement their own resource allocation policy, depending on the target platform and the characteristics of managed applications (Fig 8). For the HARPA project use cases purposes, we implemented two different policies: ***PerDeTemp*** and ***Tempura***.

PerDeTemp is a HPC-oriented resource allocation policy that allocates resources to applications according to their effective resource usage (as monitored by the HARPA-OS *Run-time Library*) and their runtime-variable performance demand. *PerDeTemp* avoids resource under- or over-assignments by allocating to the applications only the resources that they effectively need – and are able to use – to achieve their performance goals. Moreover, resource mapping is computed while taking into account *PERformance*, *DEgradation* and *TEMPerature* of the processing elements, thus avoiding both hotspots and slowdowns due to ageing/faults-induced performance variability. While the first input comes from monitoring the application activity and resource usage, the degradation is an information provided at run-time by an external monitoring tool developed by UCY as described in D.1.2. Finally the temperature of hardware resources (CPU cores) is monitored by the aforementioned power management support.

Tempura, conversely, is an embedded-oriented allocation policy that, basing on an off-line thermal/power characterization of the target system, dynamically computes the maximum amount of resources that can be exploited by applications under temperature and energy budget constraints. The goal of *Tempura* is to maximize the performance of applications while always trying to comply with the above mentioned constrains, hence avoiding hotspots and guaranteeing the required system up-time when battery-powered.

The *Resource allocation policies* subsection also contains some advanced options, such as whether to execute the selected allocation policy in parallel mode (if supported), and how many milliseconds the HARPA-OS should wait before activating the policy when an application starts, terminates or demands a different allocation. These (optional) delays allow the HARPA-OS to both

reduce the activation frequency of the policy and bundle multiple scheduling requests before actually executing the policy.

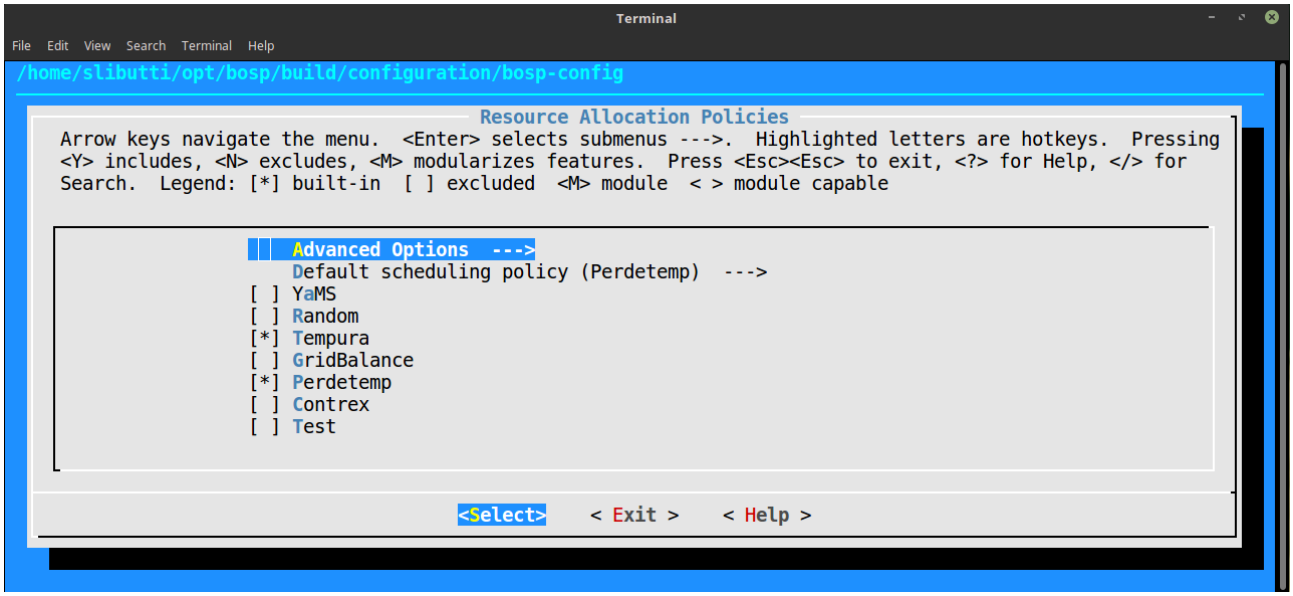

Fig 8: HARPA-OS menuconfig: Resource allocation policy selection

The *run-time library* subsection contains the HARPA-OS RunTime Library (RTLib) configuration. This subsection contains the following options:

Config option	Description
RPC channel	How applications communicate with the HARPA-OS
RPC channel timeout	Applications/HARPA-OS communication timeout [ms]
Performance counters support	Whether to monitor event counters during application execution
Unmanaged applications support	Whether to activate unmanaged support, which allows applications to execute even if HARPA-OS daemon is not started (hence without resource management support). This execution mode is useful for testing purposes or to compare applications execution with and without HARPA-OS
Performance API options	Performance feedback options, e.g., how often applications can contact the HARPA-OS to change their resource allocation.

Finally, the *Advanced options* subsection contains specific options such as: whether to build a debug version of HARPA-OS; number of priority levels supported by the resource allocation policies; and whether to activate the scheduling profiler, which collects metrics during the execution of the resource allocation policies.

5.3 Deployment

The deployment section is mainly used in the case of embedded platforms, when the HARPA-OS is cross-compiled on a host system and then deployed and installed into the target device. This section contains all the configuration options that are needed to automatize the deployment process (Fig 9).

In case of cross-compilation for external devices, the device/board/platform must be connected to the host system via SSH. The IP address (e.g. 192.168.1.101) and the user name of the target device must be specified, as well as the SSH public key, in case required.

Deployment options allow you to define also where HARPA-OS binaries will be installed (*Custom sysroot* and *Run-time RW path*). While the *Build update package* and *Install update package* will take care of the process of creating an installation archive and install it once sent to the platform.

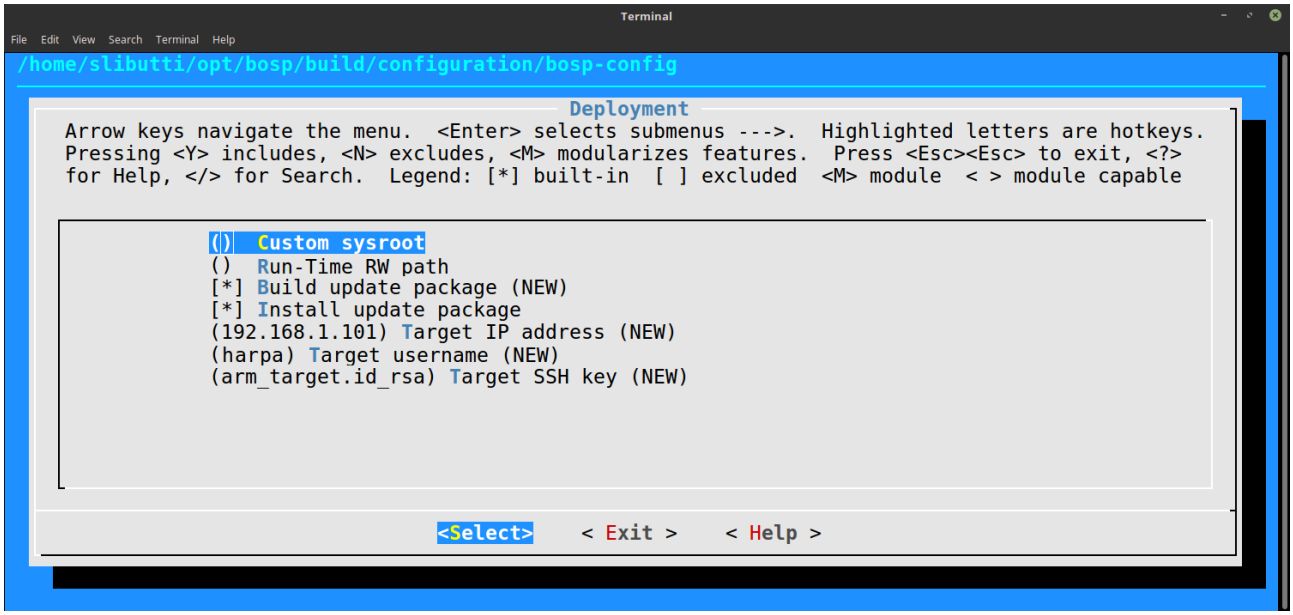

Fig 9: HARPA-OS menuconfig: Deployment configuration screen

6 Testing the execution

Now that the framework has been built and installed we can launch the HARPA-OS and test the execution of a managed application. In this section we introduce the *HARPA-OS Shell* as an easy-to-use command-line based interface for managing the HARPA-OS execution. Afterwards it is shown how to launch and stop the HARPA-OS, how to provide a new hardware platform description (if required) and finally how to execute a simple managed application, in order to verify that the resource manager is properly working.

6.1 HARPA-OS Shell

We defined a suitable set of custom command-line shortcuts in order to effectively manage the execution of HARPA-OS. This shortcuts are included in the *HARPA-OS Shell*. Assuming that **BOSP/out** is the installation directory, the HARPA-OS shell is initialized by typing the following command:

```
[BOSP] $ source out/etc/bbque/harpa_init.env
```

The outcome is a customization of the current Bash shell, as shown in Fig 10.

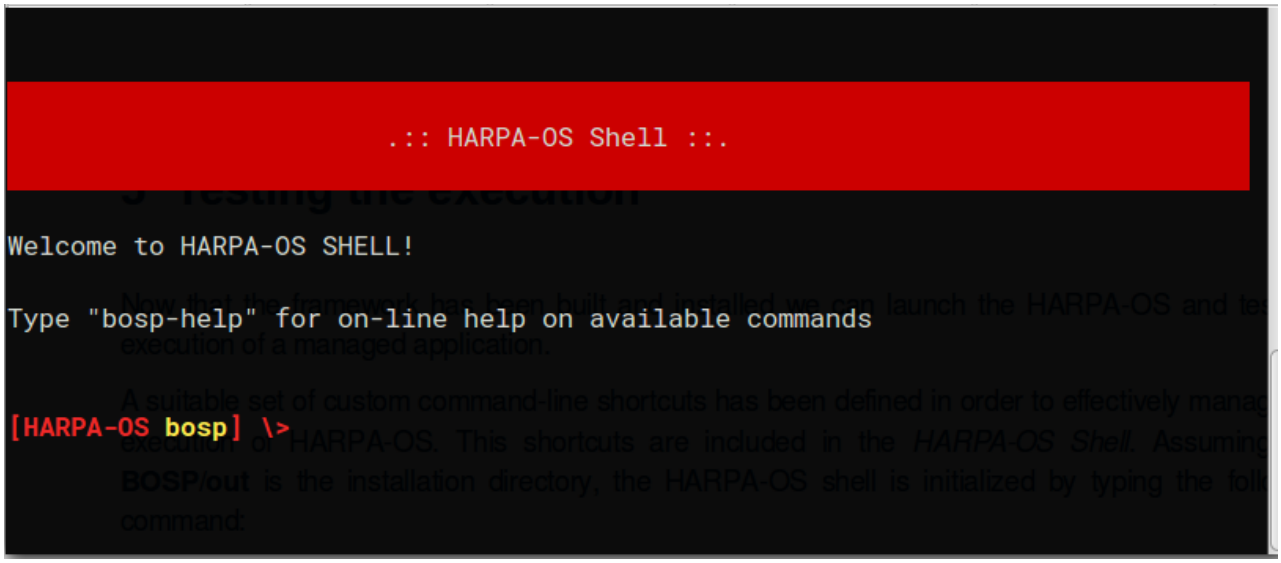

Fig 10: HARPA-OS Shell

For a complete list of the available commands we can type **bosp-help.** Here below we list the main HARPA-OS management commands:

```
:: HARPA-OS Management Commands
------------------------------------

harpaos-console - Start a set of HARPA-OS monitoring consoles
harpaos-log     - Show the HARPA-OS daemon log
harpaos-start   - Start a foreground HARPA-OS session
harpaos-startd  - Start the HARPA-OS daemon
harpaos-stats   - Dump HARPA-OS daemon statistics
harpaos-stopd   - Stop the HARPA-OS daemon
```

6.2 HARPA-OS daemon execution

Depending on the build configuration, we can choose between two commands to start HARPA-OS: **harpaos-start** and **harpaos-startd**. The former is typically used for development and debug purposes when the *Simulation mode* option has been selected from the *Run-time Resource Manager menu*. This command starts a foreground session of HARPA-OS running in simulated mode (without enforcing real constraints regarding resources assignment). This session can be stopped by simply typing the common *Ctrl+C* key combination used on UNIX systems to interrupt the current foreground process.

The **harpaos-startd** command instead launches HARPA-OS in daemon mode. This requires to have the *Simulation mode* option disabled, since in this case the resource manager runs in “real mode”. This means that a Linux Control Groups (cgroup) hierarchy is built for the enforcement of the resource assignments (as previously described in deliverable D1.2). Conversely, to stop the daemon the command is **harpaos-stopd**.

Platform description generation

When HARPA-OS is launched in daemon mode (**harpaos-startd**) for the first time, it creates a hardware platform description file. This file is in XML format and describes the available resources, e.g. cores, sockets and memory nodes, and which of these resources will be used by the HARPA-OS to accelerate integrated applications.

A wizard guides the user through the description file creation: first of all, the user must choose if processing elements will be partitioned basing on L2 or L3 cache sharing. Processing elements that share the chosen cache level will be grouped together, so that different applications can be allocated (if possible) processing elements from different groups, hence minimizing cache contention (Fig 11). Second, the user will be guided through the devices creation (Fig 12,13), i.e. choosing which processing element will be used to execute integrated applications (*managed device*) and which processing element will host not-integrated ones (*host device*). Please note that the devices may overlap, i.e. each processing element can be assigned to both the devices at the same time. Whether to make devices overlap or not depends on the degree of isolation you want integrated applications to be subject to.

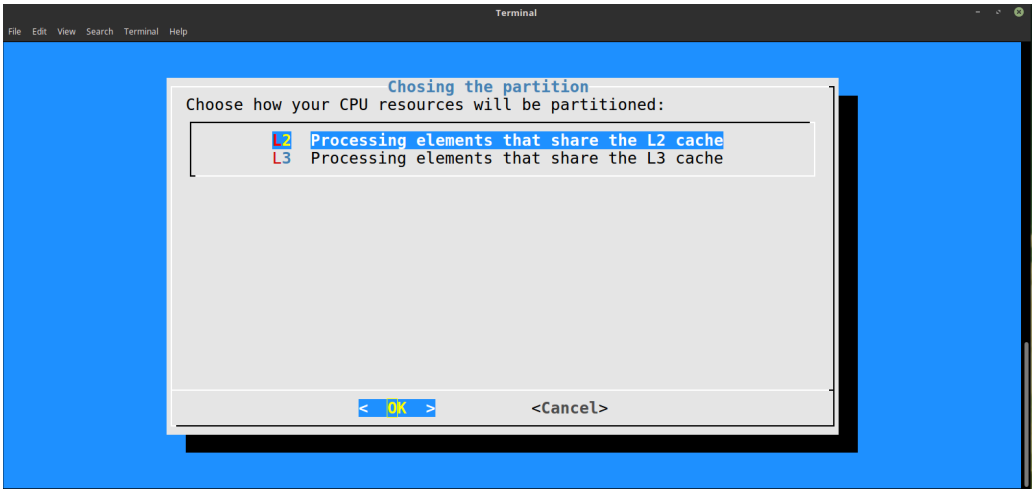

Fig 11: Hardware description wizard: processing elements cache sharing-based partitioning

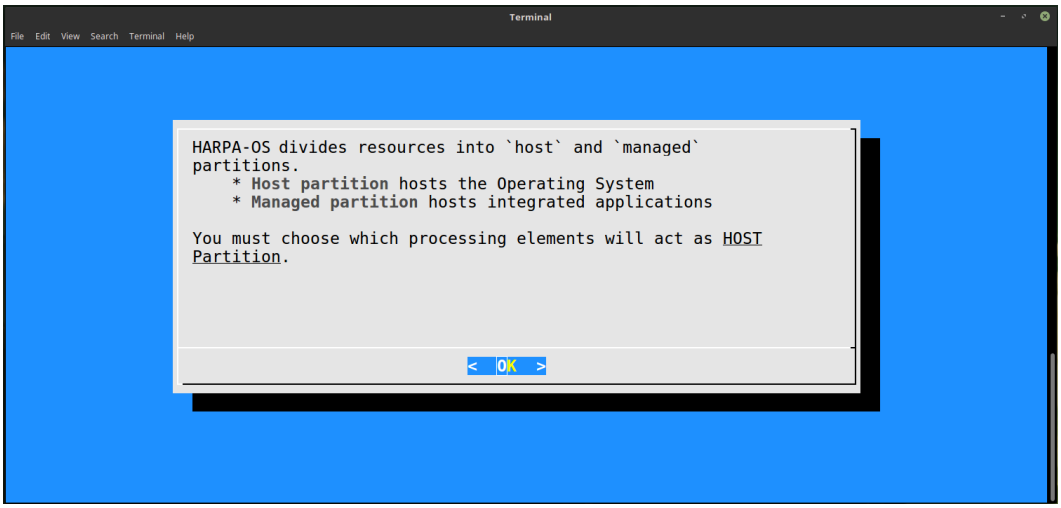

Fig 12: Hardware description wizard: defining host and managed device

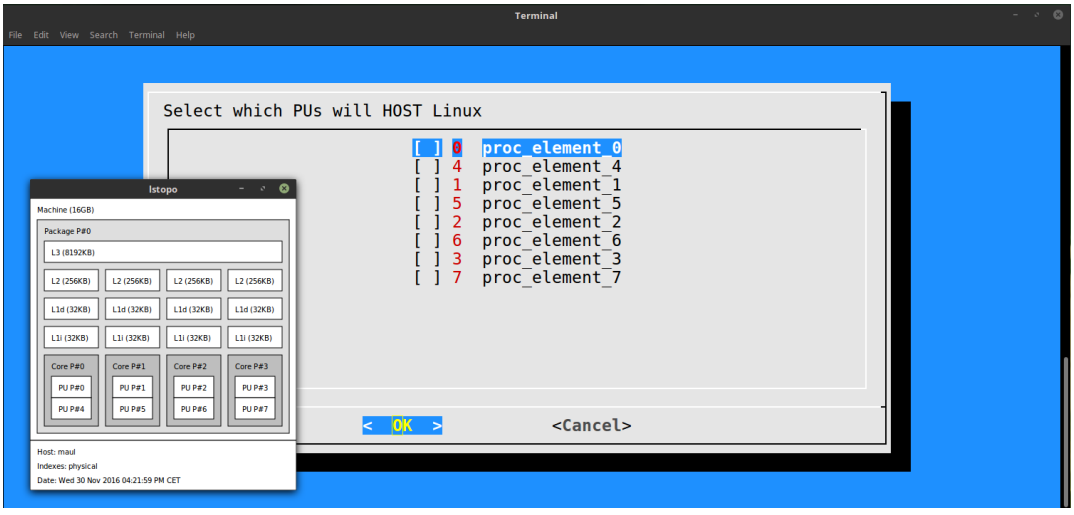

Fig 13: Hardware description wizard: guided device creation

6.3 Testing application

To check if HARPA-OS can properly run on the system we can start the resource manager (in simulated or daemon mode) and launch one of the integrated applications that we can select from the *Applications* menu of the Kconfig tool.

The fastest option is to enable the compilation of the *harpaos-testapp* (or *bbque-testapp* depending on the framework version) which is a "*Hello World!*"-like application performing some computation cycles just to test the communication with the run-time resource manager and trigger the execution of the current resource allocation policy.

If the *harpaos-testapp* has been successfully compiled with the framework, we can start the HARPA-OS and then launch the application by simply typing:

```
[HARPA-OS Shell bosp] $ harpaos-testapp
```

The expected application's output is a sequence of log messages highlighting the cycles of the *Abstract Execution Model* introduced in deliverable D.1.2 and resumed in D.5.2 with the HARPA use-cases integration. Here we report an extract of the *harpaos-testapp* output.

```
 - INFO   rtlib_testapp   : STEP 0. Initializing RTLib library, application
[bbque-testapp]...
 - INFO   rtlib_testapp   : STEP 1. Registering [001] EXCs, using recipe
[BbqRTLibTestApp]...
 - NOTICE exc             : New TestWorkload with: cycle time 333[ms], cycles
count 30
 - INFO   rtlib_testapp   : STEP 3. Starting [001] EXCs control threads...
 - INFO   rtlib_testapp   : STEP 4. Running [001] control threads...
 - NOTICE exc             : TestWorkload::onSetup():
 - NOTICE exc             : TestWorkload::onConfigure(): EXC [exc_00], AWM[01]
 - NOTICE exc             : TestWorkload::onConfigure(): EXC [exc_00], AWM[01]
=> R<PROC_quota>=200, R<PROC_nr>= 2, R<MEM>=  0
 - NOTICE exc             : TestWorkload::onConfigure(): EXC [exc_00], AWM[01]
=> cycle time 666[ms]
 - NOTICE rpc             : Set max cycle-rate @ 1.502[Hz] (min 666.000[ms])
 - NOTICE exc             : TestWorkload::onRun(): EXC [exc_00], AWM[01] =>
processing 666[ms]
 - NOTICE exc             : TestWorkload::onMonitor(): EXC [exc_00], AWM[01] =>
cycles [1/30], CPS = 0.00
...
```

The sequence of **onRun()** and **onMonitor()** function calls is repeated for the number of execution cycles specified as a command-line argument. By default *harpaos-testapp* sets this value to 5.

Once the execution is terminated, a report with run-time application profiling statistics is shown (as reported below). Overall, this set of output messages give to the user the confirmation that the

testing application and the HARPA-OS are properly running.

```
 Cumulative execution stats for 'exc_00':
  TotCycles    :      30
  StartLatency :      89 [ms]
  AwmWait      :      89 [ms]
  Configure    :       2 [ms]
  Process      :   19950 [ms]

# EXC    AWM   Uses Cycles   Total |      Min      Max |      Avg      Var
#================================+==================+=================
  exc_00 001      1     30   19950 |  665.323  665.570 |  665.455    0.002
#------------------------+        +------------------+-----------------
  exc_00 001         onRun   19949 |  665.291  664.461 |  665.349   -0.032
  exc_00 001     onMonitor       1 |    0.033    1.109 |    0.106    0.035
#------------------------+--------+------------------+-----------------
  exc_00 001   onConfigure       2 |    2.536    2.536 |    2.536    0.000
15:15:57,016 - INFO   rtlib_testapp   : ===== RTLibTestApp DONE! =====
[DBG] Barbeque RTLIB, Cleanup and release
[DBG] BbqueRPC_FIFO_Client dtor
[DBG] Releasing FIFO RPC channel
[DBG] Tx [APP_EXIT] Request FIFO_HDR [sze: 20, off: 4, typ: 1], RPC_HDR [typ:
1, pid: 17033, eid: 0], Bytes: 20...

[DBG] Rx FIFO_HDR [sze: 20, off: 4, typ: 1]
[DBG] channel thread [PID: 17033] END
```

Conversely, if an error occurs on the HARPA-OS side, the application execution is immediately stopped and the following output is reported:

```
15:35:20,728 - ERROR  rpc             : FAILED opening bbque fifo
[/home/giuseppe/Development/bosp/barbeque/../out/var/bbque/rpc_fifo] (Error 6:
No such device or address)
15:35:20,728 - ERROR  rpc             : Initialization FAILED
15:35:20,728 - ERROR  rtl             : RPC communication channel
initialization FAILED
```

In such a case, please check that the HARPA-OS has been actually launched. If this has been done, please contact the BOSP development team using the general mailing list introduced in Section 1.

7 Conclusions

In this deliverable we presented the final release of the HARPA-OS engine, one of the core components of the HARPA software stack.

Other than being a quick-start guide for users, this document summarizes the achievement of a fundamental objective of the HARPA project: the release of a novel run-time management framework capable of allocating computing resources to applications, taking into account the performance/QoS requirements provided and addressing the variability issues due to both software (run-time variable requirements) and hardware causes (hardware ageing/degradation, thermal hotspots, energy budget variability, etc…).

The final experimental evaluation of the framework is provided in deliverable D.5.5, where use cases from embedded and HPC are studied, considering multi-threaded applications running on embedded platforms and distributed high-performance systems.

The HARPA-OS is part of an open-source software project that aims at collecting public contributions also in the future, going beyond the timeframe of the HARPA project.

Future plans include supporting mobile Android-based devices and improve the control of distributed systems by synchronizing the decisional process among multiple running instances of HARPA-OS.