Project Number 318763

D3.5 – Operating System with Real-Time Support and FPGA Interface

Version 1.0
1 December 2014
Final

Public Distribution

Scuola Superiore Sant'Anna, University of York

Project Partners: **aicas**, **HMI**, **petaFuel**, **SOFTEAM**, **Scuola Superiore Sant'Anna**, **The Open Group**, **University of Stuttgart**, **University of York**, **Brno University of Technology**

Project Partner Contact Information

aicas Fridtjof Siebert Haid-und-Neue Strasse 18 76131 Karlsruhe Germany Tel: +49 721 66396823 E-mail: siebert@aicas.com	**HMI** Markus Schneider Im Breitspiel 11 C 69126 Heidelberg Germany Tel: +49 6221 7260 0 E-mail: schneider@hmi-tec.com
petaFuel Ludwig Adam Muenchnerstrasse 4 85354 Freising Germany Tel: +49 8161 40 60 202 E-mail: ludwig.adam@petafuel.de	**SOFTEAM** Andrey Sadovykh Avenue Victor Hugo 21 75016 Paris France Tel: +33 1 3012 1857 E-mail: andrey.sadovykh@softeam.fr
Scuola Superiore Sant'Anna Mauro Marinoni via Moruzzi 1 56124 Pisa Italy Tel: +39 050 882039 E-mail: m.marinoni@sssup.it	**The Open Group** Scott Hansen Avenue du Parc de Woluwe 56 1160 Brussels Belgium Tel: +32 2 675 1136 E-mail: s.hansen@opengroup.org
University of Stuttgart Bastian Koller Nobelstrasse 19 70569 Stuttgart Germany Tel: +49 711 68565891 E-mail: koller@hlrs.de	**University of York** Neil Audsley Deramore Lane York YO10 5GH United Kingdom Tel: +44 1904 325571 E-mail: neil.audsley@cs.york.ac.uk
Brno University of Technology Pavel Smrz Bozetechova 2 61266 Brno Czech Republic Tel: +420 54114 1282 E-mail: smrz@fit.vutbr.cz	

Contents

Document Control

Version	Status	Date
0.1	Document outline	17 October 2014
0.2	Draft	12 November 2014
1.0	QA for EC Delivery	1 December 2014

Executive Summary

This document constitutes deliverable 3.5 - *Operating System with Real-Time Support and FPGA Interface* of work package 3 of the JUNIPER project. It consists of the integration of the FPGA support inside the prototype implementation of the Linux kernel with real-time bandwidth reservation scheduling support.

The purpose of this deliverable is to describe the implementation of the FPGA interface that provides communication between the FPGA and the Java Virtual Machine through the Operating System with Real-Time Support. In order to better understand the environment, a brief overview of the main features of the Operating System with Real-Time Support are presented, together with a description of the functionalities implemented in the first prototype.

Later, the FPGA support for the JUNIPER platform is presented: first, an overview of the FPGA subsystem selected and its interaction with the other hardare components are described; then, the architecture of the Linux device driver for the FPGA board is described, together with instruction to compile it and to use it throught the Linux sysfs interface; finally, the mapping of the JFMI calls to the device driver API is presented.

This document describes the implementation of the FPGA interface described in Deliverable 3.4 - *FPGA Integration Definition*. It is developed to run inside the Operating System with Real-Time Support, which is described in Deliverable 3.1 - *Operating System Real-Time Support Definition*. An interim version has been implemented in Deliverable 3.2 - *Prototype Operating System with Real-Time Support* and the final version will be submitted later in Deliverable 3.6 - *Final Operating System with Real-Time Support*.

1 Linux Kernel with Real-Time Enhancements

This deliverable presents the first version of the Operating System with Real-Time support and FPGA Interface. It includes the real-time features provided within the prototype described in deliverable D3.2 together with the kernel support for the interaction with the FPGA board as required in the JUNIPER project. In particular, the real-time features included in this release are relative to the reservation of single tasks and disk I/O.

The CPU reservation capabilities are provided at the task level by the SCHED_DEADLINE [2] kernel scheduling policy; the extended to support CPU reservation for groups of tasks will be achieved in deliverable D3.6 by implementing the Bounded Delay Multipartition (BDM) [3].

In order to provide reservation at disk level, the Linux kernel is enriched with the Budget Fair Queueing (BFQ) [4] block scheduler that provides a fair allocation of I/O bandwidth; the amount of bandwidth actually reserved depends on the number of requests and their relative priorities.

The last real-time improvement included in this release is the RT_PREEMPT patch [1], that as been improved to work in conjunction with the reservation mechanism provided by the SCHED_DEADLINE policy. This patch increases the Linux kernel determinism by replacing most kernel spin-locks with mutexes that support Priority Inheritance (PI) and by moving interrupts and software interrupts to kernel threads.

The traditional support for FPGA boards (hosted on a PCI card within the node) in the Linux kernel only provides the communication with the FPGA through the PCIe bus. This is not enough to fruitfully exploit the FPGA capabilities from the JVM; for this reason a kernel module has been developed to leverage the PCI support and expose to the virtual machine an API enabling the application processes to manage accelerators, in terms of loading them, passing data to them, invoking them and read back results.

Deliverable 3.2 - Prototype Operating System with Real-Time Support (Appendix A), presents the procedure required to obtain a working copy of the Operating System prototype including the Linux kernel version 3.14 enhanced with the patches previously described. In particular, Appendix A.2 shows how to install the prototype kernel on a Linux Debian distribution.

2 FPGA Integration

2.1 System Overview

As shown in D 2.4, an accelerator is implemented as a piece of hardware which exposes two busses; an AXI slave port which is used to control the core (e.g. the start signal, query the core's status) and an AXI master port which is used such that the core can request data from main memory. These cores are then wrapped and instantiated within a VHDL wrapper, which simply forwards the signals out, but also contains functionality to decouple the core's outputs such that dynamic reconfiguration can take place.

In addition to the accelerators and DDR, a small amount of fast block-RAM is created on the board. This storage is for static information about the design, such as the number of accelerators present, the address of each control port and other metadata about the current hardware design. This RAM is also read/write, such that information about the current status of each accelerator can be stored and persist through updates of the JFM module.

A number of these cores are then instantiated and connected to the DDR on the board via an AXI bus. In addition, a PCI-Express core is instantiated and connected to the DDR, the control ports of each accelerator, the small block-RAM and to the FPGA's reconfiguration port, allowing for dynamic reconfiguration. A block-diagram of this can be found in Figure 1.

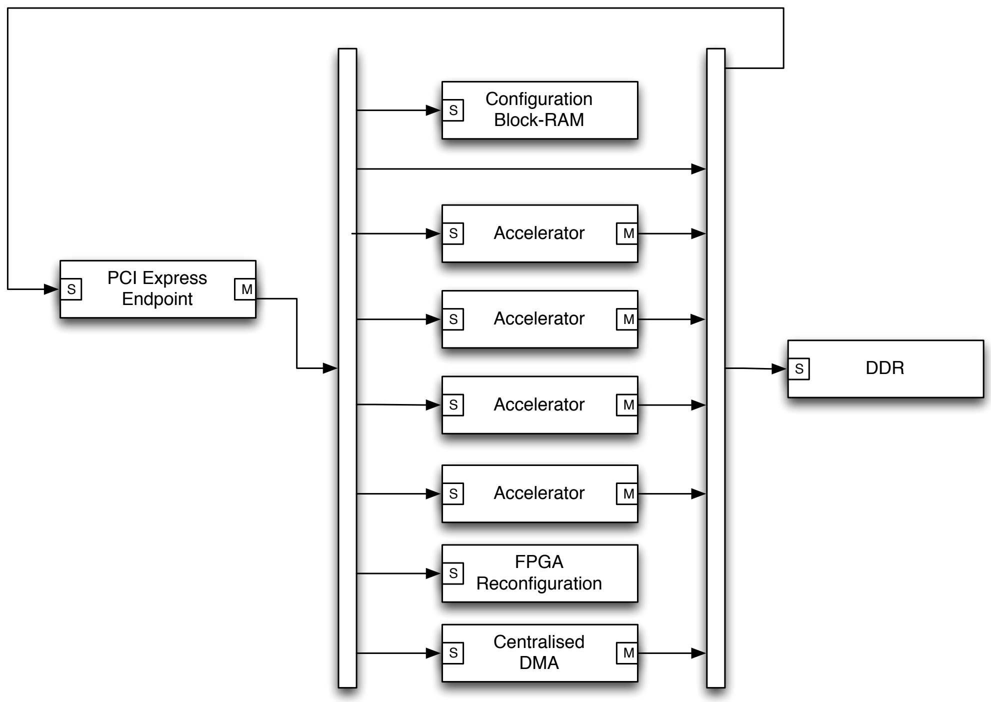

Figure 1: Block Diagram of the FPGA Implementation

The upshot of this configuration is that each block can be addressed as a memory-mapped peripheral, which is then mapped into the PCI-Express address space. This then implies a purely logical configuration for the JFM module, simplifying the implementation of both the hardware and the kernel driver.

2.2 Driver Architecture

The kernel driver is implemented as a loadable module for Linux versions 2.6.* and 3.*, and is distributed along with this deliverable. The process for building this module is documented below:

```
tar -xf juniper_jfm.tar.gz    # Extract the driver source
cd juniper_jfm
make                          # Build the module
sudo insmod juniper_jfm.ko    # And load it.
```

The kernel driver will then register itself for all devices with a vendor ID of `0x10EE` and a product ID of `0x7011`, and register a new device class under `sysfs` for these devices, `/sys/class/juniper`.

When a board matching this vendor/product pair is found, it will have a new virtual device created for it which is used to manage the FPGA board itself (e.g. `/sys/class/juniper/juniper0`, containing the following files:

- `device`: The standard Linux sysfs link pointing to the parent device. In this case, it will point to the device node for the PCI endpoint.
- `reconfig`: A write-only binary attribute which is used to access the FPGA's reconfiguration port.
- `power`: A standard Linux sysfs folder used for device power management. This is currently ignored.
- `subsystem`: A standard Linux sysfs link back to the device's class, in this case, `/sys/class/juniper`.

A better design would be to have a `reconfig` port in each accelerator which can reconfigure *only* the logic occupied by that accelerator in order to prevent accidentally overwriting the wrong accelerator tile. This is currently extremely difficult as a partial bitfile has no notion of which tile it is for, only the actual changed logic, and the Xilinx bitstream format is undocumented. For this reason, the reconfiguration port has been made global.

The device node also contains a set of sub-devices (e.g. `/sys/class/juniper/juniper0/juniper0a`) which represent each accelerator. These sub-devices are created using the information found in the configuration block-RAM mentioned in Section 2.1. Note that this block-RAM is hidden from the user, and is only ever read or written to directly by the JFM driver. These sub-devices contain the standard Linux sysfs files (i.e. `device`, `power` and `subsystem`) as before, but with the following additional files:

- `accel_hold`: Read/write. Used to de-couple a reconfigurable module from the outside world, to make it ready for reconfiguration. Writing a “0” or “1” can be used to control the decoupling logic, and reading the file will yield whether the core is de-coupled or not.

- `accel_idle`: Read-only. Yields "1" if the core is idle (i.e. doing no processing), or "0" if not.
- `accel_start`: Write-only. Writing a "1" to this file will cause the accelerator to begin execution.
- `mem`: Read/write. This file represents the memory partition used by the accelerator, and is the data which the accelerator operates upon and writes out. Reads and writes to this device node will cause the FPGA board to initiate DMA transfers, hence allowing for both fast and asynchronous transfers.

An example session can be seen below. This shows starting the accelerator, waiting for a result, then retrieving the results from the computation.

```
# Get to the base directory
cd /sys/class/juniper/juniper0/
# Copy the data to be processed to accelerator a
dd if=~/accel_data of=./juniper0a/mem bs=4 count=64
# Start it!
echo "1" > ./juniper0a/accel_start
# Wait for the end
while [[ ./juniper0a/accel_idle == "0" ]];
# Get the data back out
dd if=./juniper0a/mem ~/accel_results bs=4 count=64
```

Another example session can be seen below. This shows performing reconfiguration of the core, then starting the new design.

```
# Get to the base directory
cd /sys/class/juniper/juniper0
# De-couple the core to be replaced
echo "1" > ./juniper0a/accel_hold
# Reconfigure
cat ~/juniper_newtile.bin > ./reconfig
# Release the hold
echo "0" > ./juniper0a/accel_hold
# And start it!
echo "1" > ./juniper0a/accel_start
```

2.3 Relation to User-Land

This kernel module provides a simple interface to the devices contained within the FPGA, as outlined in D 3.4, with a couple of key differences. Firstly, using `sysfs` attributes greatly simplifies the implementation of the kernel module, compared with implementing all I/O calls (i.e. `open()`, `read()`, `write()` and `seek()`) manually within the kernel driver, as required when using raw device nodes. In addition, using `sysfs` allows for a much more logical viewpoint of the hardware compared with using `ioctl()` calls on the same device node. Finally, using a simpler interface to the FPGA moves most of the required `JFIM` functionality into user-mode, further simplifying the kernel driver and ensuring stability of the system.

The required `JFIM` calls from D 3.4 can then be mapped into this driver as follows:

- `JFIM_get_resource_map()`: Reads out the configuration block-RAM as raw data. This data structure should then be decoded by the user-land process.
- `JFIM_load_accelerator(from, to)`: Due to the limitations listed in Section 2.2, the user-land must select the correct bit-file to use and pass the contents of this file to the `reconfig` device attribute.
- `JFIM_remove_accelerator(from, to)`: Either causes the user-land to forget the presence of an accelerator, or for the userland to use a "blanking" bitfile to overwrite a reconfigurable region with dummy logic.
- `JFIM_start_accelerator(to)`: Writes a "0" to the correct `accel_hold` attribute, in order to release the reset of a module and connect its ports to the memory bus.
- `JFIM_stop_accelerator(to)`: Writes a "1" to the correct `accel_hold` attribute to disconnect an accelerator from the memory bus, and hold it in reset.
- `JFIM_reset_accelerator(to)`: Toggles `accel_hold` on and off, since this also controls the reset of an accelerator.
- `JFIM_invoke_accelerator(to)`: Writes a "1" to the correct `accel_start` attribute.
- `JFIM_read/write_memory(from, to, offset, length)`: Reads from the correct `mem` attribute. The user-land module will need to perform address translation to write to the correct accelerator.
- `JFIM_control(to, addr, len)`: Functionally identical to `JFIM_write_memory`, although without the address mapping stage.

3 Conclusions and Further Work

The document has provided a concise overview of the development work in Linux support for the JUNIPER approach, namely:

- Further development of Linux kernel real-time support via the RT_PREEMPT patch and scheduling policy support for reservations;
- Development of the FPGA interface and support from Linux.

In terms of real-time Linux support for JUNIPER, this work will contine within Workpackage 3, with a delivery of the final version of Linux for JUNIPER due in month 30. This will include the FPGA support as outlined in this document. Also, we note that the FPGA support will be further refined within Workpackage 3 up until month 30 to support any further changes to the Java VM (Workpackage 4); the use of the JUNIPER programming model (within Workpackage 3); and static / dynamic acceleration of the JUNIPER applications on FPGA (Workpackage 2).

References

[1] RT_PREEMPT group. Rt_preempt patch set. `http://www.kernel.org/pub/linux/kernel/projects/rt/`.

[2] Juri Lelli, Giuseppe Lipari, Dario Faggioli, and Tommaso Cucinotta. An efficient and scalable implementation of global edf in linux. In *Proceedings of the International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT)*, 2011.

[3] Giuseppe Lipari and Enrico Bini. A framework for hierarchical scheduling on multiprocessors: from application requirements to run-time allocation. In *Proceedings of the* 31^{st} *IEEE Real-Time Systems Symposium*, pages 249–258, San Diego, CA, USA, December 2010.

[4] Paolo Valente and Fabio Checconi. High throughput disk scheduling with fair bandwidth distribution. *IEEE Trans. Computers*, 59(9):1172–1186, 2010.