Free and Open Source Software at CERN: Integration of Drivers in The Linux Kernel

Juan David González Cobas, Samuel Iglesias Gonsálvez,
Julian Howard Lewis, Javier Serrano, Manohar Vanga
(CERN, Geneva),
Emilio G. Cota (Columbia University, NY; formerly at CERN),
Alessandro Rubini, Federico Vaga (University of Pavia)

ICALEPCS'2011

CERN Controls System Front End Computers (FECs)

The controls system relies on FECs on several form factors/buses, most of them based on Single-Board Computers (SBCs)

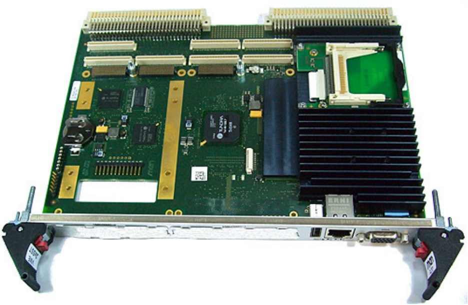
- Number of FECs: 1140
- Number of VME crates: 710

For the VME crates, the ongoing renovation process gives

- CES RIO2/RIO3 SBCs with PowerPC CPUs running LynxOS (around 605 crates by August 2011), to
- MEN-A20 SBCs with Intel CPUs running real-time Linux (around 105 by August 2011).

MEN A20 and the TSI148 chip and driver

The MEN A20 SBC is an Intel Core 2 Duo-based board interfacing to the VME bus via a Tundra TSI148 PCI-X to VME bridge chip.



A driver for the `tsi148`

Some data about our driver for the `tsi148` PCI-X to VME bridge.

A driver for the `tsi148`

Some data about our driver for the `tsi148` PCI-X to VME bridge.

- Developed at CERN in spring 2009 (Sébastien Dugué)

A driver for the `tsi148`

Some data about our driver for the `tsi148` PCI-X to VME bridge.

- Developed at CERN in spring 2009 (Sébastien Dugué)
- Maintained and extended by Emilio G. Cota during 2010

A driver for the `tsi148`

Some data about our driver for the `tsi148` PCI-X to VME bridge.

- Developed at CERN in spring 2009 (Sébastien Dugué)
- Maintained and extended by Emilio G. Cota during 2010
- Currently maintained by Manohar Vanga (see below)

A driver for the `tsi148`

Some data about our driver for the `tsi148` PCI-X to VME bridge.

- Developed at CERN in spring 2009 (Sébastien Dugué)
- Maintained and extended by Emilio G. Cota during 2010
- Currently maintained by Manohar Vanga (see below)
- API via exported symbols (kernel) and old-style `ioctl()` (user) interface.

A driver for the `tsi148`

Some data about our driver for the `tsi148` PCI-X to VME bridge.

- Developed at CERN in spring 2009 (Sébastien Dugué)
- Maintained and extended by Emilio G. Cota during 2010
- Currently maintained by Manohar Vanga (see below)
- API via exported symbols (kernel) and old-style `ioctl()` (user) interface.
- Backward compatible at the API level with the original LynxOS CES library (well, almost) and offering a newer API as well

A driver for the tsi148

Some data about our driver for the tsi148 PCI-X to VME bridge.

- Developed at CERN in spring 2009 (Sébastien Dugué)
- Maintained and extended by Emilio G. Cota during 2010
- Currently maintained by Manohar Vanga (see below)
- API via exported symbols (kernel) and old-style `ioctl()` (user) interface.
- Backward compatible at the API level with the original LynxOS CES library (well, almost) and offering a newer API as well

Beginning as an in-house and CERN-centric development

Why going upstream? (2010)

By mid-2010, the decision is taken to submit the driver for acceptance in the Linux kernel main tree. Motivation:

- Smoother maintenance in the (frequent) case of kernel API changes (see `Documentation/stable_api_nonsense.txt` in the kernel tree).
- Widespread distribution of the code base, which can then be enhanced and get contributed by researchers
- Contributing back in return to the many benefits the FOSS community gives us.

The original motivations were more ideological than practical

How the driver was merged (2010–2011)

How the driver was merged (2010–2011)

Who, when, how

- Merge process with pre-existing `./staging/` driver for the Tundra Universe and TSI148 chips
- Four-round process (Emilio G. Cota, Manohar Vanga)
- Core device model modifications accepted by mid 2011

How the driver was merged (2010–2011)

Who, when, how

- Merge process with pre-existing `./staging/` driver for the Tundra Universe and TSI148 chips
- Four-round process (Emilio G. Cota, Manohar Vanga)
- Core device model modifications accepted by mid 2011

Lessons learned

- It is hard, LKML and maintainers are tough
- One must be prepared to compromise (design, APIs, tools)
- One must build a reputation slowly
- Requires patience

Lessons learned

But the most important one was that our initial motivations

Lessons learned

But the most important one was that our initial motivations

turned out to be wrong

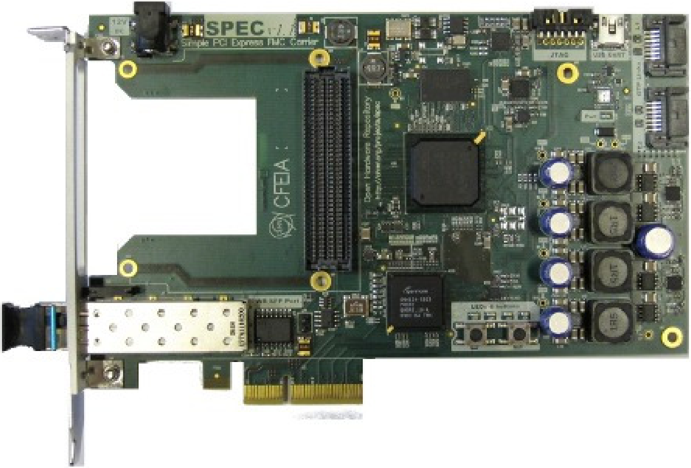
Lessons learned

But the most important one was that our initial motivations

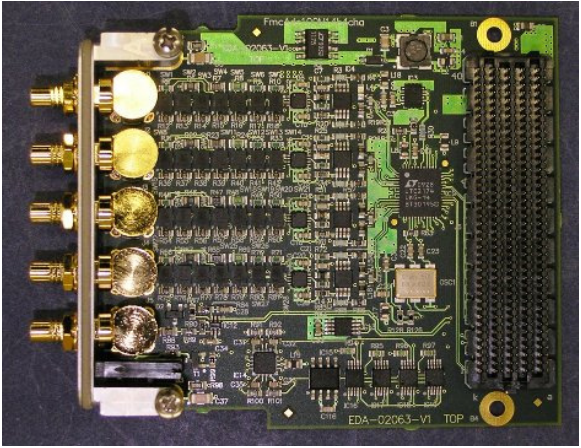
turned out to be wrong

Why?

A typical data acquisition application: carrier



A typical data acquisition application: mezzanine

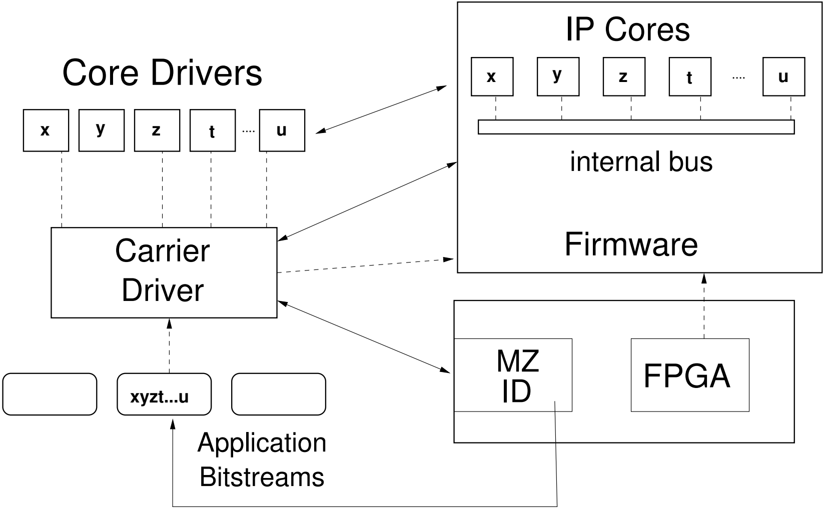


The FMC family of boards

This is a substantial part of our standard HW kit, currently under development (see `http://www.ohwr.org/projects/fmc-projects`).

- carriers in PCIe and VME format
- simple mezzanines with electronics for ADCs, DACs, DIO and endless other applications
- circuitry in the mezzanine
- FPGA application logic in the carrier
- *logic in the FPGA is organized as a set of IP cores interconnected through an internal bus named Wishbone*

Architecture of the FMC drivers



Drivers for the FMC family

The main concepts for the design of these drivers are

Drivers for the FMC family

The main concepts for the design of these drivers are

- modular structure that reflects the core structure of the firmware

Drivers for the FMC family

The main concepts for the design of these drivers are

- modular structure that reflects the core structure of the firmware
- one-to-one mapping driver ↔ core (usually)

Drivers for the FMC family

The main concepts for the design of these drivers are

- modular structure that reflects the core structure of the firmware
- one-to-one mapping driver ↔ core (usually)
- ability to dynamically load bitstreams by application

Drivers for the FMC family

The main concepts for the design of these drivers are

- modular structure that reflects the core structure of the firmware
- one-to-one mapping driver ↔ core (usually)
- ability to dynamically load bitstreams by application

On the whole, the driver for the carrier board acts as a basic firmware loader and a bridge driver (with device enumeration *à la PCI)* between the host bus (PCIe, VME) and the FPGA interconnection bus

Drivers for the FMC family

The main concepts for the design of these drivers are

- modular structure that reflects the core structure of the firmware
- one-to-one mapping driver ↔ core (usually)
- ability to dynamically load bitstreams by application

On the whole, the driver for the carrier board acts as a basic firmware loader and a bridge driver (with device enumeration *à la PCI)* between the host bus (PCIe, VME) and the FPGA interconnection bus

It will be (we hope) the first Wishbone bus driver in the mainstream kernel ⇒ will to go upstream, timeliness

Industrial I/O frameworks

Industrial I/O frameworks

In Linux staging area

- Comedi
- IIO

Industrial I/O frameworks

In Linux staging area

- Comedi
- IIO

Drawbacks

- do not suit our needs
- interfaces are cumbersome

Industrial I/O frameworks

In Linux staging area

- Comedi
- IIO

Drawbacks

- do not suit our needs
- interfaces are cumbersome

Then `zio` comes

- Alessandro Rubini and Federico Vaga, main developers
- Integration mainstream *ab initio*
- See (soon) under `http://www.ohwr.org/`
- Clean design conforming to Linux kernel practice

Next candidates for (zio) integration

CERN-developed drivers for

Next candidates for (zio) integration

CERN-developed drivers for

- Struck SIS33xx ADCs

Next candidates for (zio) integration

CERN-developed drivers for

- Struck SIS33xx ADCs
- Tews TPCI200/TVME200 carries plus IPOCTAL serial boards

Next candidates for (zio) integration

CERN-developed drivers for

- Struck SIS33xx ADCs
- Tews TPCI200/TVME200 carries plus IPOCTAL serial boards
- all the CERN BE/CO-supported FMC boards in the Open Hardware Repository

Next candidates for (zio) integration

CERN-developed drivers for

- Struck SIS33xx ADCs
- Tews TPCI200/TVME200 carries plus IPOCTAL serial boards
- all the CERN BE/CO-supported FMC boards in the Open Hardware Repository
- timing receivers, White Rabbit, etc.

Why did we do it?

It gave us much more than we thought in the first place

Why did we do it?

It gave us much more than we thought in the first place

- Smoother maintenance in the (frequent) case of kernel API changes.
- Widespread distribution of the code base.
- Contributing back to the the FOSS community.

Why did we do it?

It gave us much more than we thought in the first place

- Smoother maintenance in the (frequent) case of kernel API changes.
- Widespread distribution of the code base.
- Contributing back to the the FOSS community.
- Very strict process of peer review of the code by knowledgeable and specialised maintainers.

Why did we do it?

It gave us much more than we thought in the first place

- Smoother maintenance in the (frequent) case of kernel API changes.
- Widespread distribution of the code base.
- Contributing back to the the FOSS community.
- Very strict process of peer review of the code by knowledgeable and specialised maintainers.
- Input (consulting!) from the topmost experts in the field.

Why did we do it?

It gave us much more than we thought in the first place

- Smoother maintenance in the (frequent) case of kernel API changes.
- Widespread distribution of the code base.
- Contributing back to the the FOSS community.
- Very strict process of peer review of the code by knowledgeable and specialised maintainers.
- Input (consulting!) from the topmost experts in the field.
- Avoidance of suboptimal, *ad hoc* solutions in favour of the best ones from the technical point of view.

Why did we do it?

It gave us much more than we thought in the first place

- Smoother maintenance in the (frequent) case of kernel API changes.
- Widespread distribution of the code base.
- Contributing back to the the FOSS community.
- Very strict process of peer review of the code by knowledgeable and specialised maintainers.
- Input (consulting!) from the topmost experts in the field.
- Avoidance of suboptimal, *ad hoc* solutions in favour of the best ones from the technical point of view.
- Use of best practice and bleeding-edge tools selected by experienced programmers, *e.g.* `git`, `sparse` and `coccinelle`.