



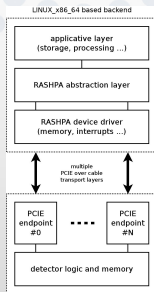
## The use of hardware virtualization in RASHPA

# Contents



- RASHPA context reminder
- Device virtualization
- The VPCIE framework

# Context - RASHPA reminder



RASHPA basic goal is to transfer **detector** memory contents into a **backend** for further processing, visualisation or storage. To do so, RASHPA relies on low level hardware and software components

- a data transport layer, currently PCI Express over cable
- a data transmission engine implemented on XILINX FPGA
- LINUX kernel device driver and userland software

# Context - RASHPA project issues



Still being prototyped

- how to simplify hardware software codesign ?

Aims at using multiple PCIE links. Yet, the first prototype is single link

- how to scale the prototype platform ?

Well positionned to interface with PCIE based accelerating technologies

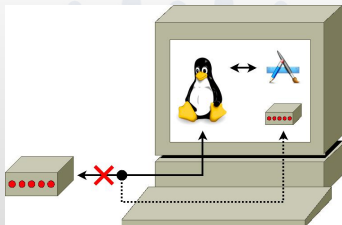
- how to test unavailable (too recent or expensive) hardware ?

# Context - solution



Solution: **device virtualization**

# Device virtualization - main idea



Applications on a **host** machine access hardware via interfaces. By **instrumenting** these interfaces, one can **redirect** the accesses to a software implementing the device. The device is said to be **virtualized**.

# Device virtualization - hardware access redirection (0)



Virtualization assumes device accesses are made using a known interface

- software library API (system calls)
- memory mapped registers
- CPU specific instructions (in, out)

# Device virtualization - hardware access redirection (1)



The interface is then instrumented to redirect access to the virtual device

- software hooking (code instrumentation, library replacing)
- instruction emulation (QEMU)
- hardware traps (page protection)
- architecture support (INTEL VT)
- paravirtualization (XEN)



# Device virtualization - applications



## Device virtualization example applications

- support: application running on unmaintained platform
- security: sandboxing, reverse engineering
- quality: debugging, fault injection
- testing: milkymist, zinq, android

# VPCIE - goals



VPCIE, a Virtual PCIE framework made for RASHPA

- virtualize PCIE endpoints
- the backend software must run **without any modification**
- generic enough to fit other projects
- virtualized device implementation usable in the real device
- thus, possible to use VHDL in the virtualized device

# VPCIE - opensource building blocks



## QEMU

- [http://wiki.qemu.org/Main\\_Page](http://wiki.qemu.org/Main_Page)
- architecture emulator (X86\_64, ARM ...)
- used to trap PCIE hardware accesses

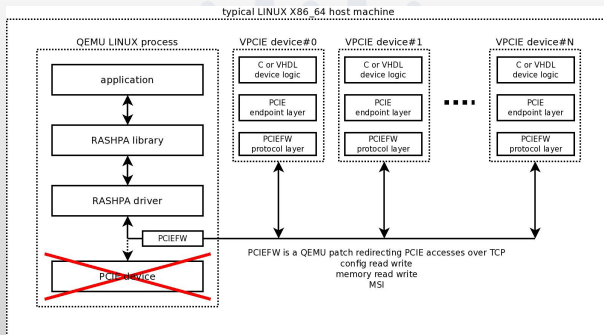
## GHDL

- <http://ghdl.free.fr>
- VHDL frontend for GCC
- used to implement device in VHDL

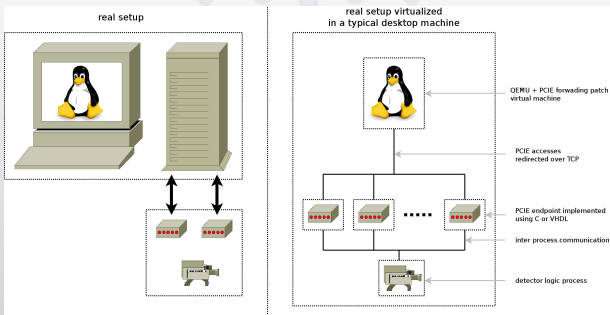
## GTKWAVE

- <http://gtkwave.sourceforge.net>
- used to view simulated design signals in time

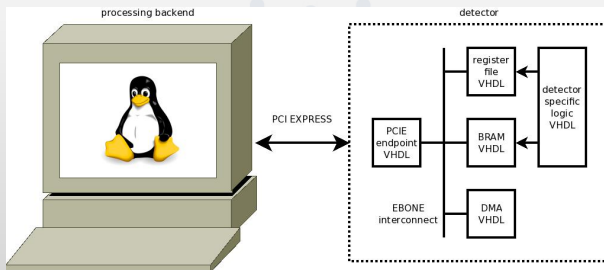
# VPCIE - implementation



# VPCIE - RASHPA possible setup



# VPCIE - other possible setups



# VPCIE - RASHPA backend



RASHPA backend is a full featured LINUX system

- runs in a QEMU virtual machine
- PCIE accesses are trapped and sent over TCP to the devices
- PCIE forwarder is available as a QEMU patch

# VPCIE - RASHPA devices



## RASHPA devices

- run as a LINUX processes, can be duplicated at will
- can be implemented in C or VHDL
- PCIE made simple, focus on device logic



# VPCIE - VHDL interface

```
entity endpoint is
  port
  (
    rst: in std_ulogic;
    clk: in std_ulogic;

    req_en: out std_ulogic;
    req_wr: out std_ulogic;
    req_bar: out std_ulogic_vector(pcie.BAR.WIDTH - 1 downto 0);
    req_addr: out std_ulogic_vector(pcie.ADDR.WIDTH - 1 downto 0);
    req_data: out std_ulogic_vector(pcie.DATA.WIDTH - 1 downto 0);

    rep_en: in std_ulogic;
    rep_data: in std_ulogic_vector(pcie.DATA.WIDTH - 1 downto 0);

    mwr_en: in std_ulogic;
    mwr_addr: in std_ulogic_vector(pcie.ADDR.WIDTH - 1 downto 0);
    mwr_data: in std_ulogic_vector(pcie.PAYLOAD.WIDTH - 1 downto 0);
    mwr_size: in std_ulogic_vector(pcie.SIZE.WIDTH - 1 downto 0);

    msi_en: in std_ulogic
  );
end entity;
```

# VPCIE - C interface

---

```
/* runtime initialization */
int pcie_init_net(pcie_dev_t*, ...);
int pcie_fini(pcie_dev_t*);
int pcie_loop(pcie_dev_t*);

/* misc config byte accessors */
int pcie_set_deviceid(pcie_dev_t*, ...);
int pcie_set_vendorid(pcie_dev_t*, ...);

/* PCIE BAR access handlers */
typedef void (*pcie_readfn_t)(uint64_t, void*, size_t, void*);
typedef void (*pcie_writefn_t)(uint64_t, const void*, size_t, void*);
int pcie_set_bar(pcie_dev_t*, ..., pcie_readfn_t, pcie_writefn_t, ...);

/* host memory read write operations */
int pcie_write_host_mem(pcie_dev_t*, uint64_t, size_t*);
int pcie_read_host_mem(pcie_dev_t*, uint64_t, size_t*);

/* send an MSI */
int pcie_send_msi(pcie_dev_t*);
```

---

# VPCIE - benefits

## Hardware software codesign

- reduce development time
- no **modification** on the backend software (esp. driver)
- act as a stimuli generator for VHDL simulations

## Platform scaling

- one PCIE endpoint per LINUX process

## Investigate unavailable technologies

- NVM Express support for QEMU is available

# VPCIE - more benefits



## Virtual machine

- resources easily changed (RAM amount ...)
- test with different CPU architectures (x86, x86\_64, ARM ...)
- small LINUX disk images reduce reboot times (few seconds)

## Fault injection

- LINUX and software response to PCIE device disconnection

# VPCIE - availability



<https://github.com/texane/vpcie>

**CRISP**  
The Cluster of Research Infrastructures  
for Synergies in Physics