Virtual Open Systems

SVFF: An Automated Framework for SR-IOV Virtual Function Management in FPGA Accelerated Virtualized Environments

Stefano Cirici – Michele Paolino – Daniel Raho

CITS23 – Genova, July 11th 2023

EVEREST
DESIGN ENVIRONMENT
FOR EXTREME-SCALE BIG DATA ANALYTICS
ON HETEROGENEOUS PLATFORMS

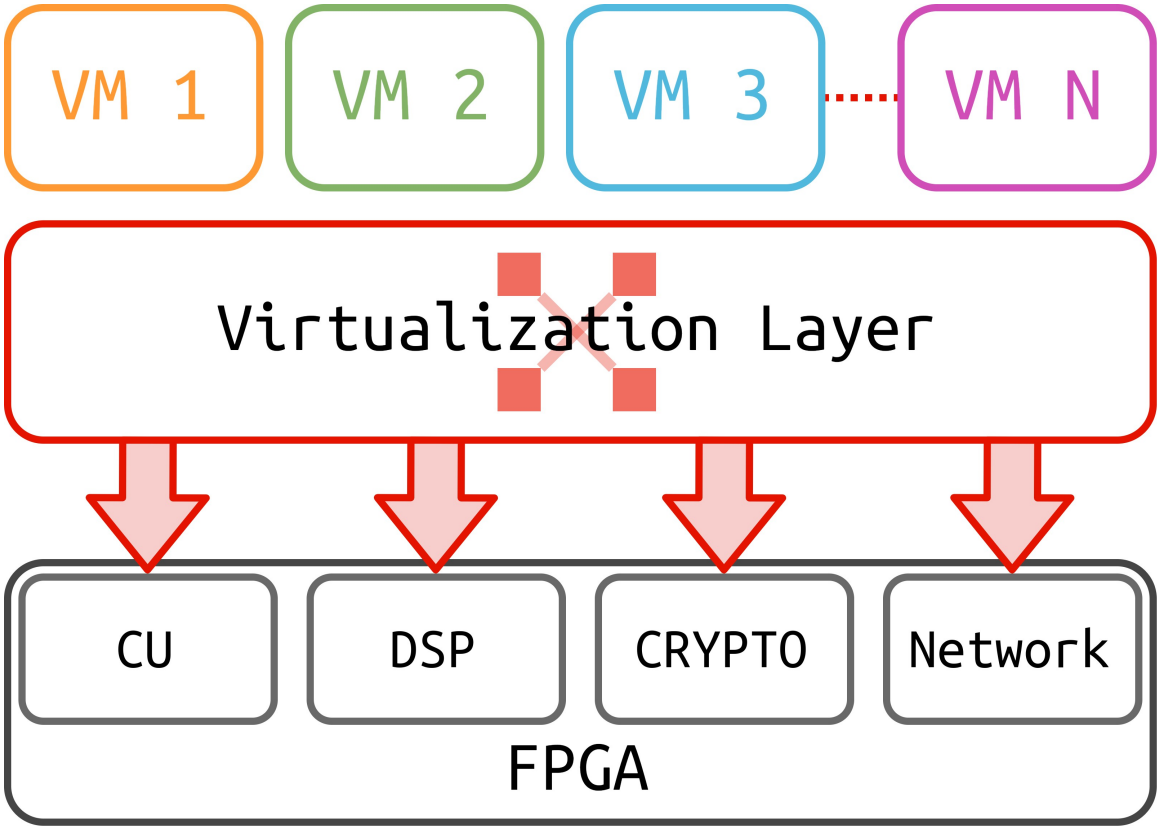
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957269

Agenda

- Introduction
- SR-IOV Overview
- SVFF Framework
  - The Hardware part
  - The Software part
    - Init and Reconf
    - Pause functionality
- Integration
- Testing and results
- Conclusions

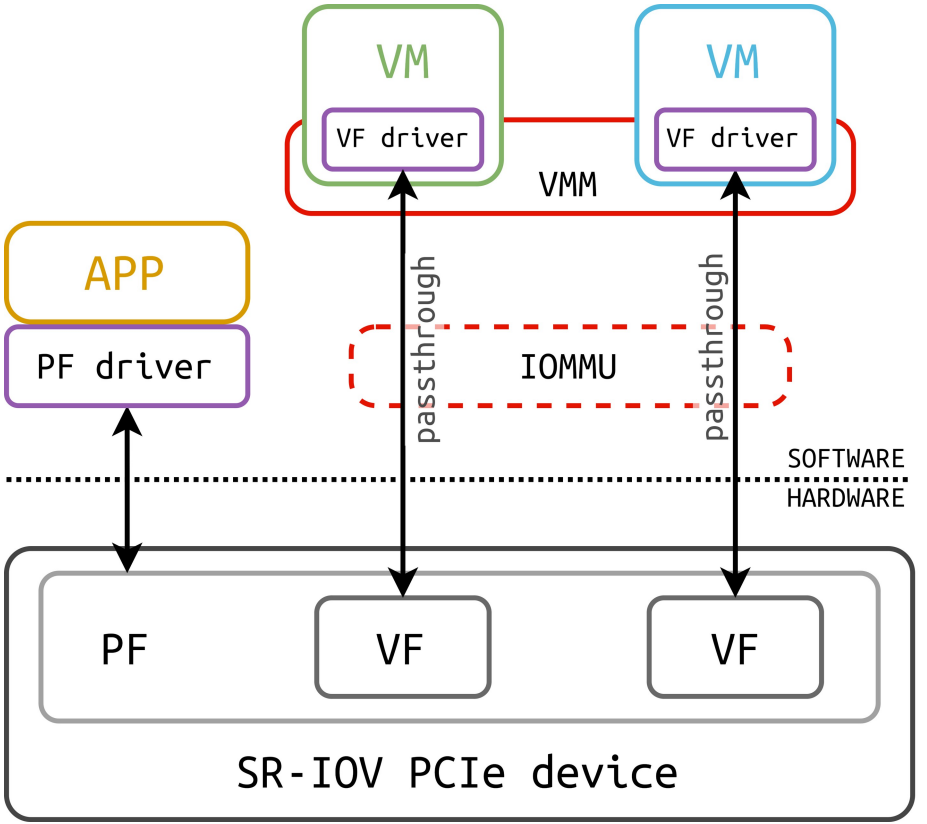
Introduction

- Focus on FPGA-accelerated virtualized environments
- Sharing of a physical I/O device across multiple VMs
- Device passthrough with SR-IOV
- Management of device to VM association



SR-IOV – Single-Root I/O Virtualization

- Single-Root I/O Virtualization (SR-IOV) allows a single PCIe device to appear as multiple, separate PCIe devices
- Direct access to the hardware with minimal to no overhead
- Partitioning Physical Function (PF) devices into Virtual Functions (VFs), which are then allocated to VMs



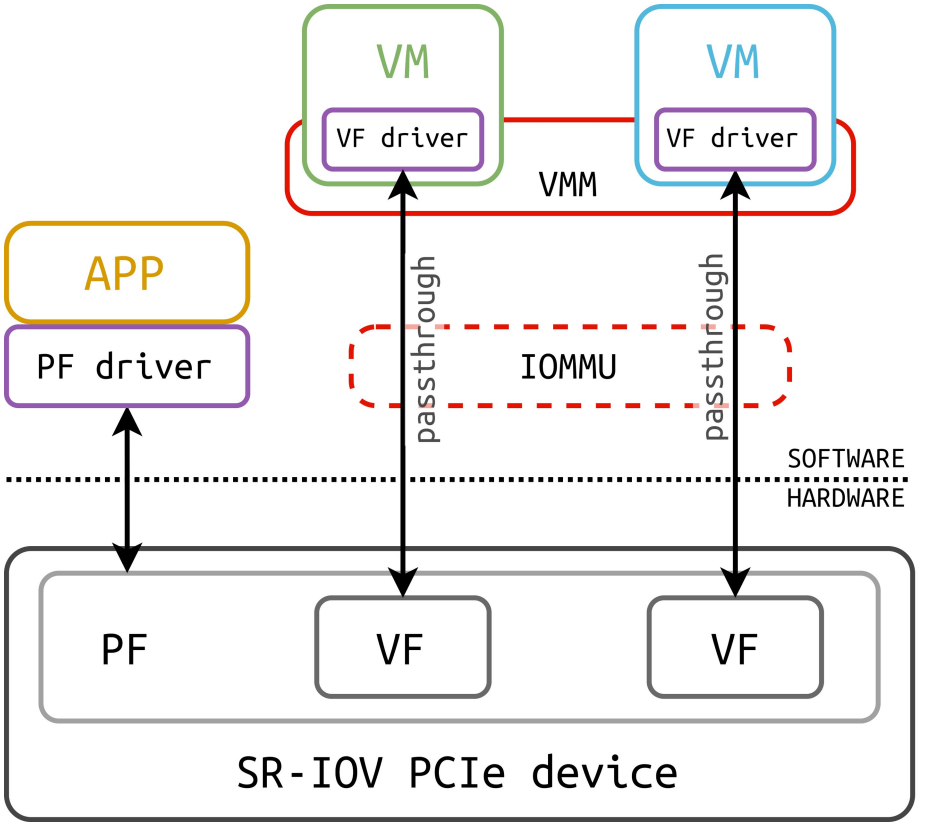
SR-IOV – Advantages and Disadvantages

Advantages:

- More devices on the same HW
- Improved performance and reduced host CPU and memory utilization
- Control each VF independently

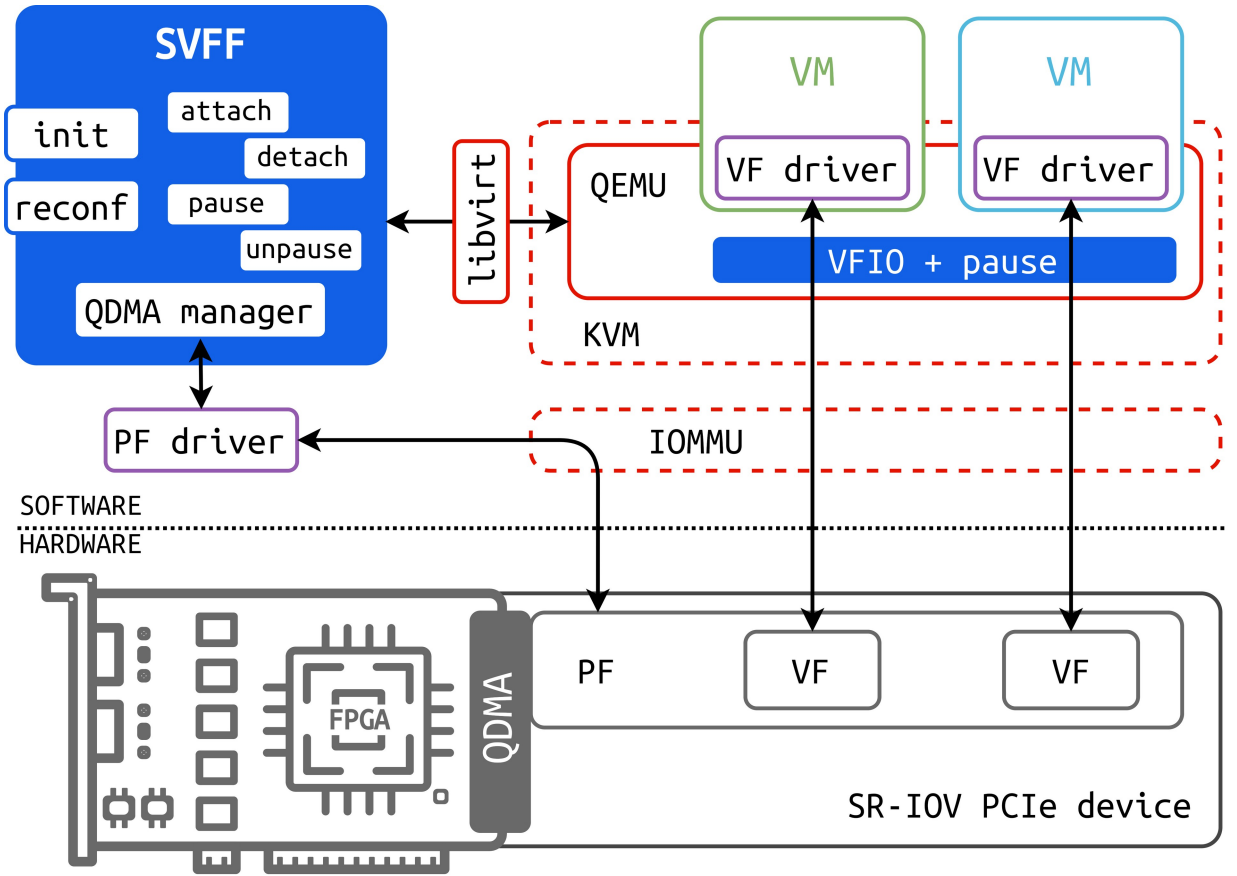
Disadvantages:

- Changing a PF configuration requires detaching all the VFs from the VMs



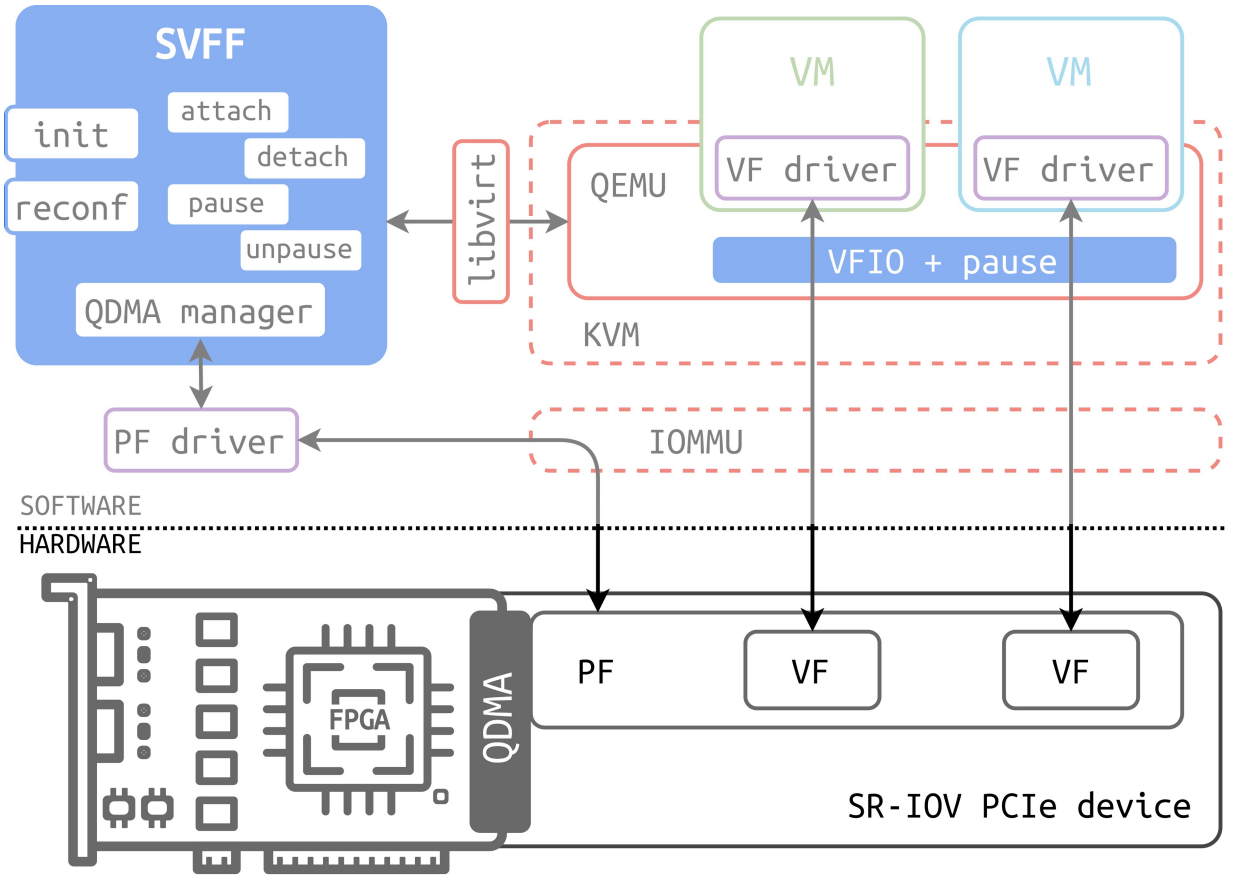
SVFF – The Framework

- Simplify and enhance the management of VF
- Solve the lack of SR-IOV re-configuration support on guests

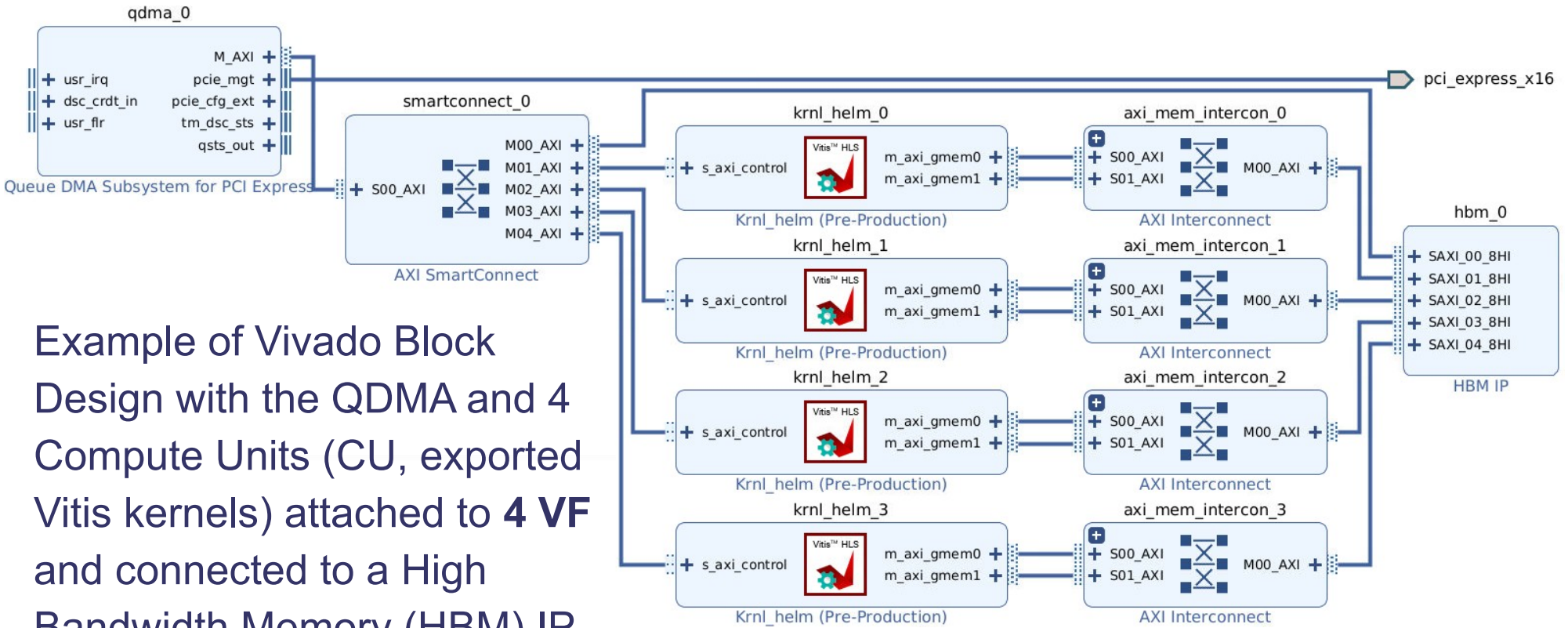


SVFF – The Hardware part

- Provides SR-IOV support on a PCIe-attached FPGA and serves as the HW virtualization support
- The QDMA IP can be easily integrated into the FPGA design and supports up to 4 PFs and 252 VFs



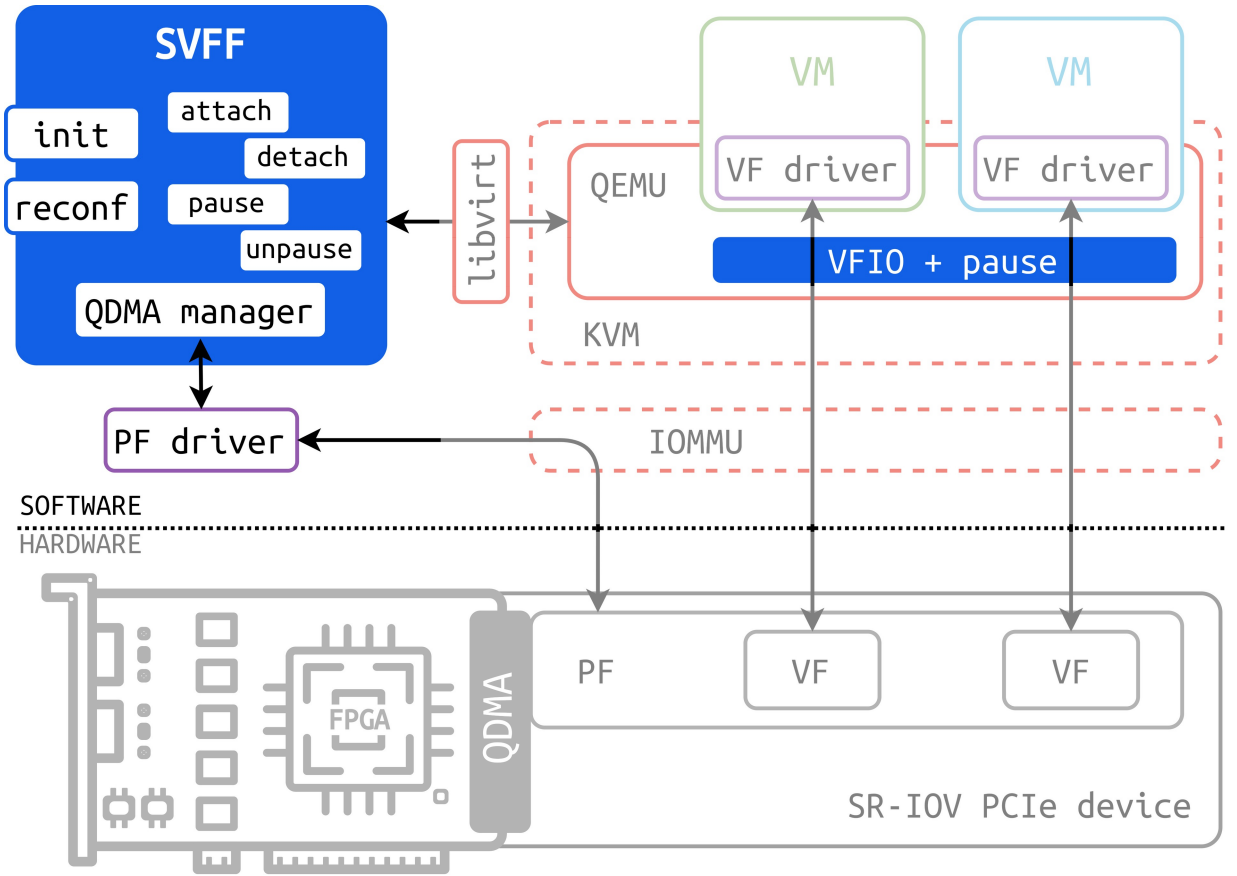
FPGA Design



Example of Vivado Block Design with the QDMA and 4 Compute Units (CU, exported Vitis kernels) attached to **4 VF** and connected to a High Bandwidth Memory (HBM) IP.

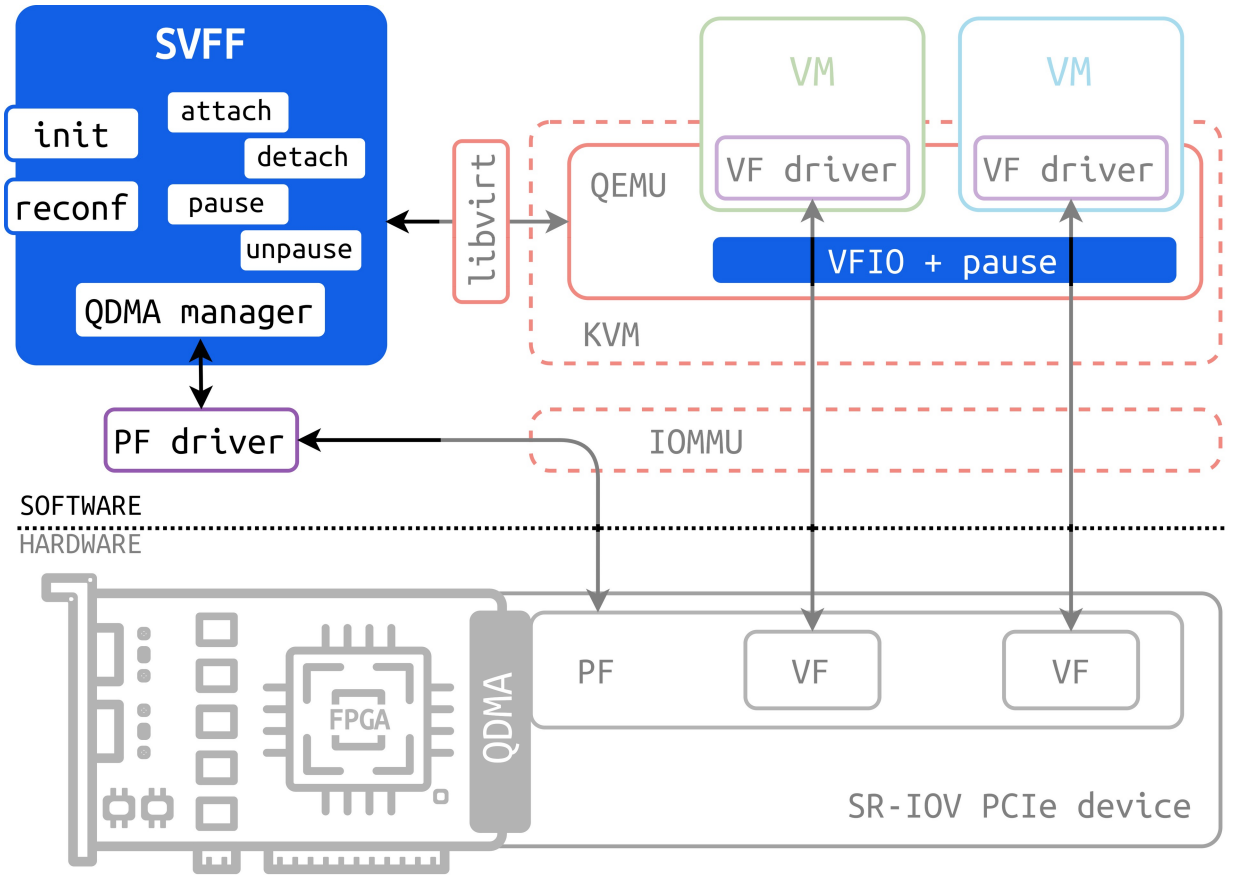
SVFF – The Software part

- Exposes two functions to the user: **init** and **reconf**
- Automate the initialization, creation, and configuration of VFs, providing the **pause** functionality
- Interacts with the VMM through libvirt and with the device using the PF



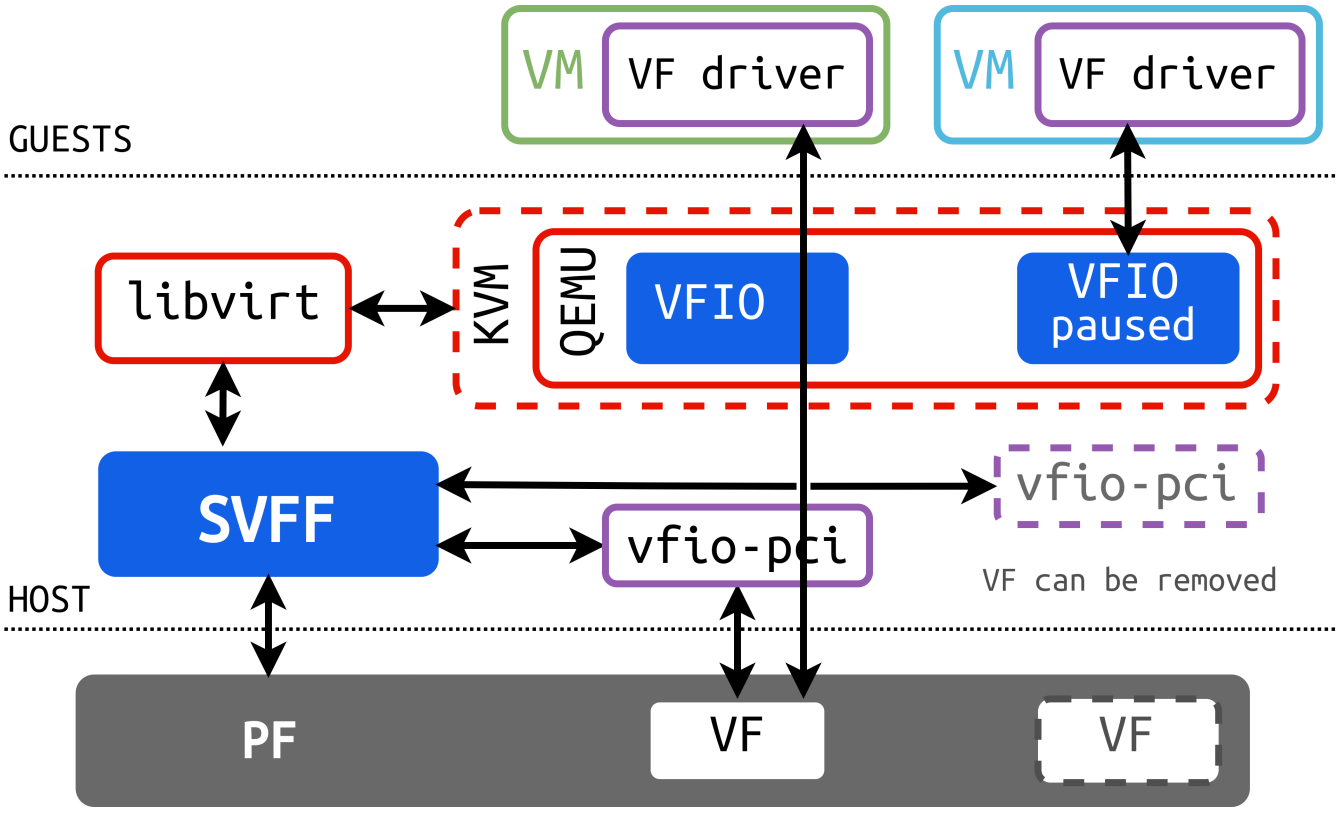
SVFF – Init and Reconf

- **Init** - Initialize the FPGA device, eventually flashing the bitstream, creating the initial number of VFs and, attaching them to the VMs.
- **Reconf** – Change the VF configuration after the init, with the possibility to pause the devices instead of detaching them.



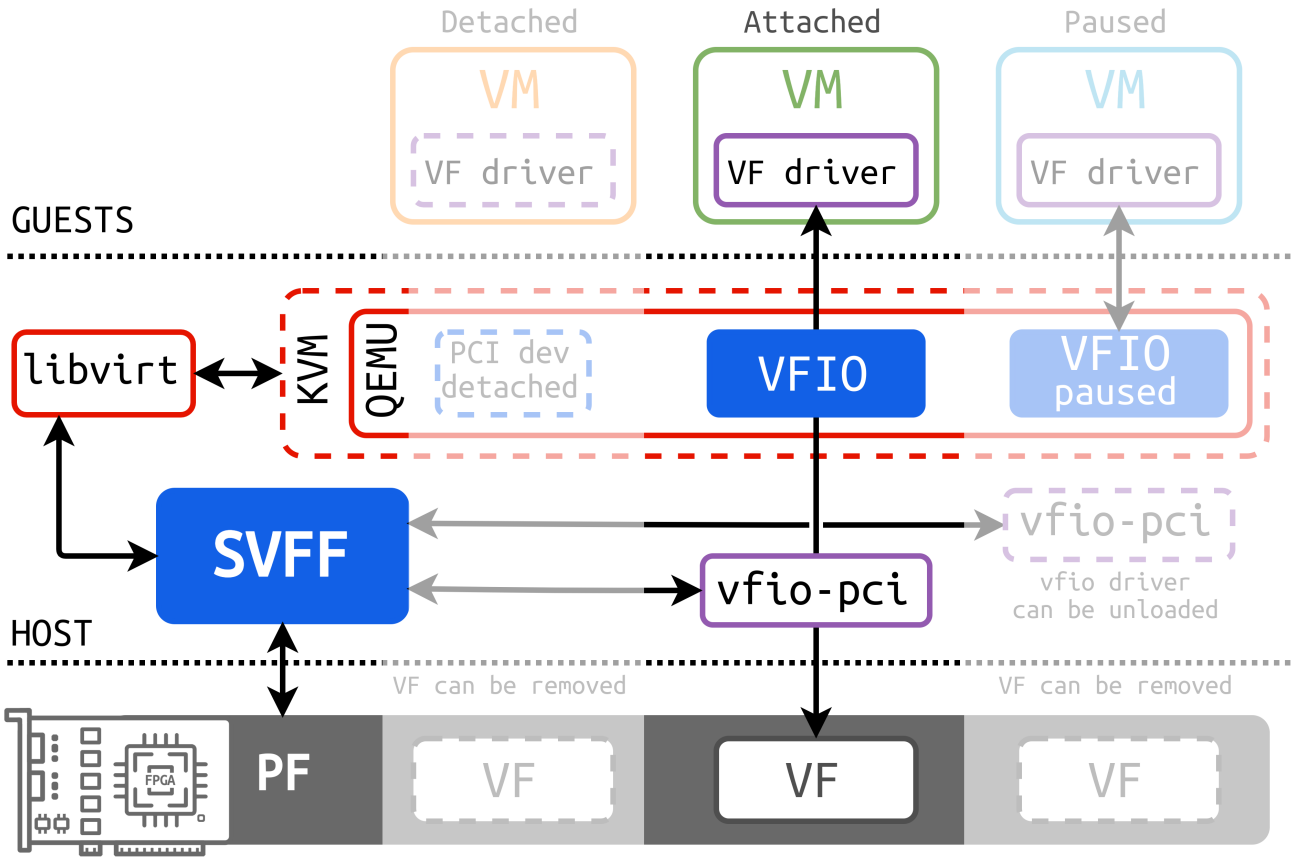
SVFF – Pause Functionality

- Implemented in the VFIO device in QEMU
- Detach VFs from the host without detaching them from the guest
- Transparently reconfigure VF already attached to guest VMs



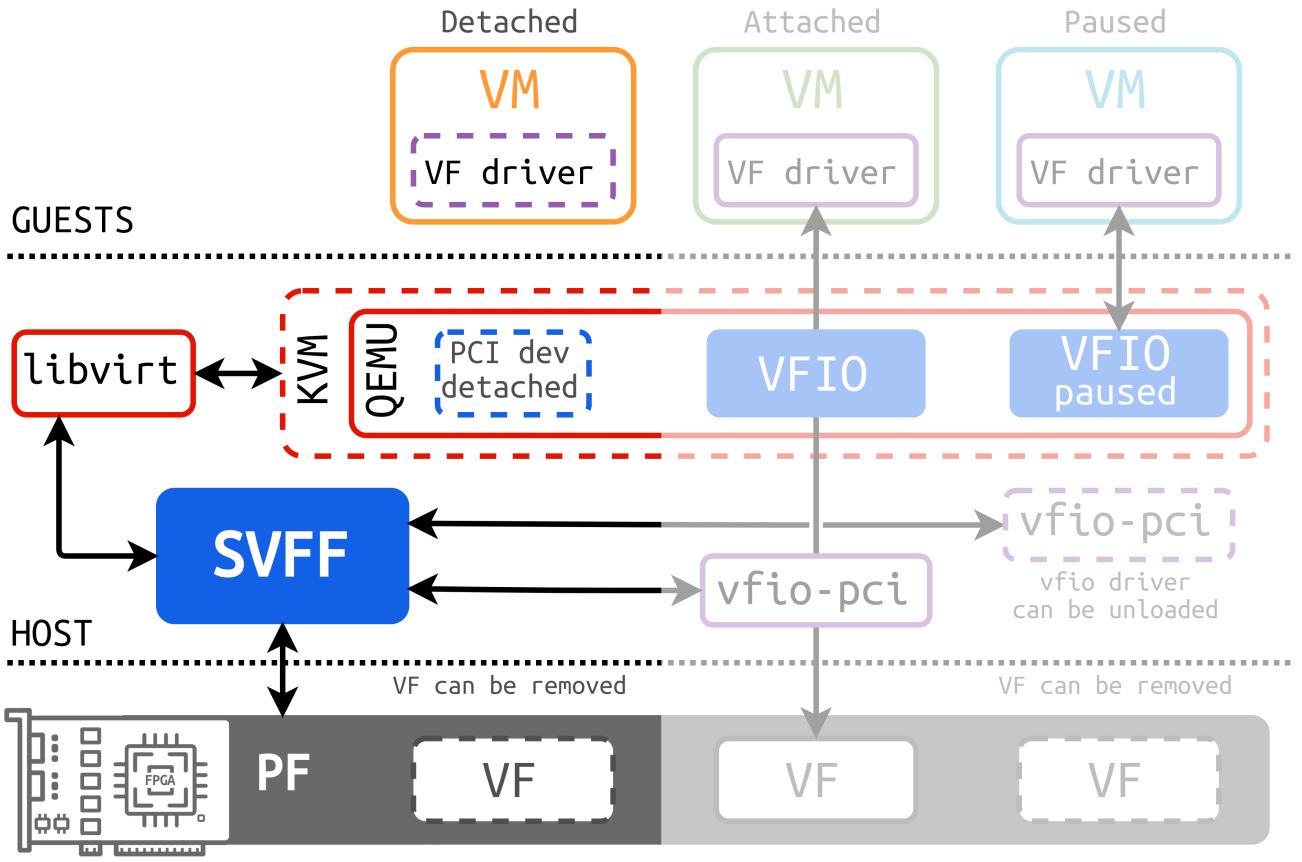
SVFF – Attached state

- vfio-pci is used as back-end driver on the host
- QEMU manages the passthrough operations with the VFIO device
- The guest can use the native VF driver



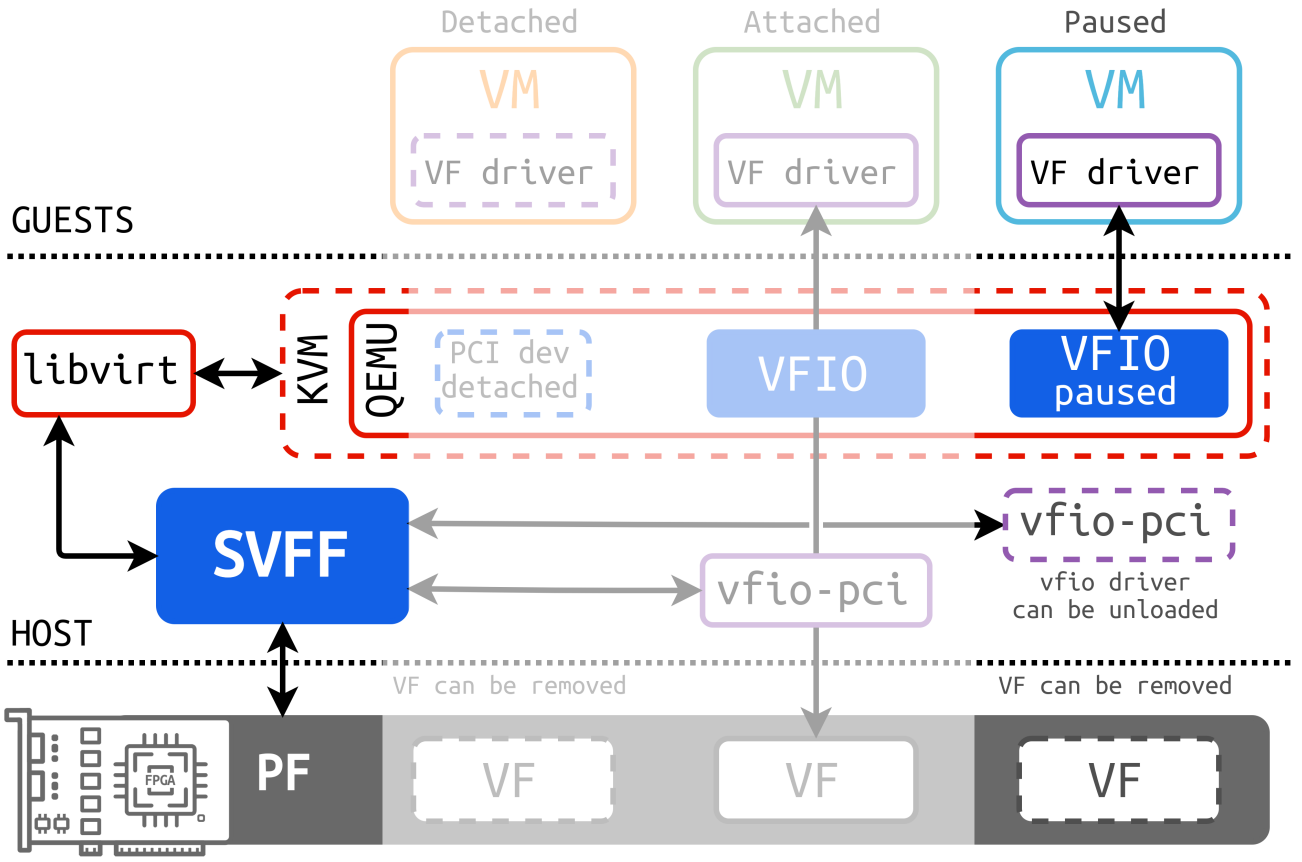
SVFF – Detached state

- The host and guest drivers are unloaded
- The guest VM will see the PCI device disappear from the system
- The VF can be safely removed or re-assigned to another VM



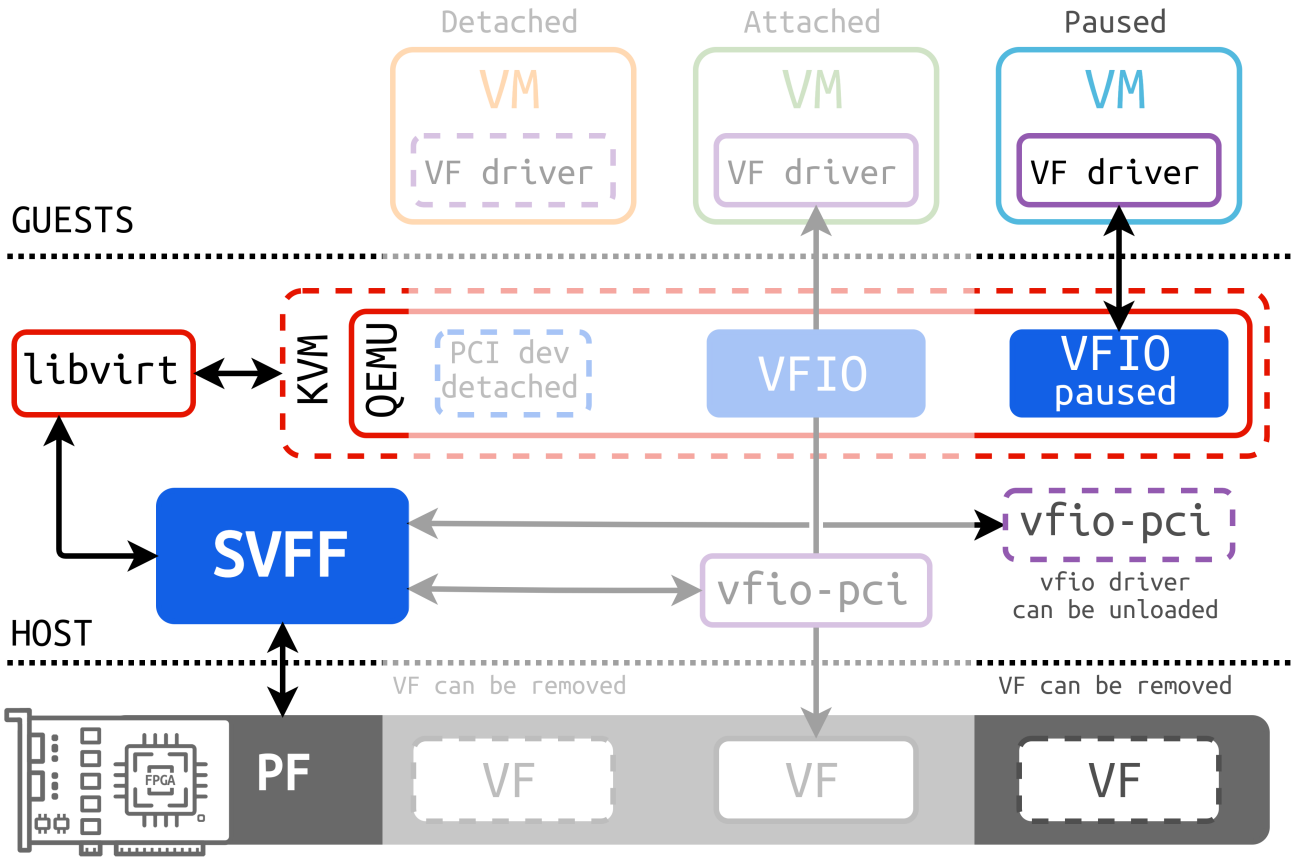
SVFF – Paused state [1]

- Only the host driver is unloaded
- The guest VM will continue seeing the PCI device but cannot do any I/O operations
- QEMU keeps the VFIO device paused ignoring guest requests



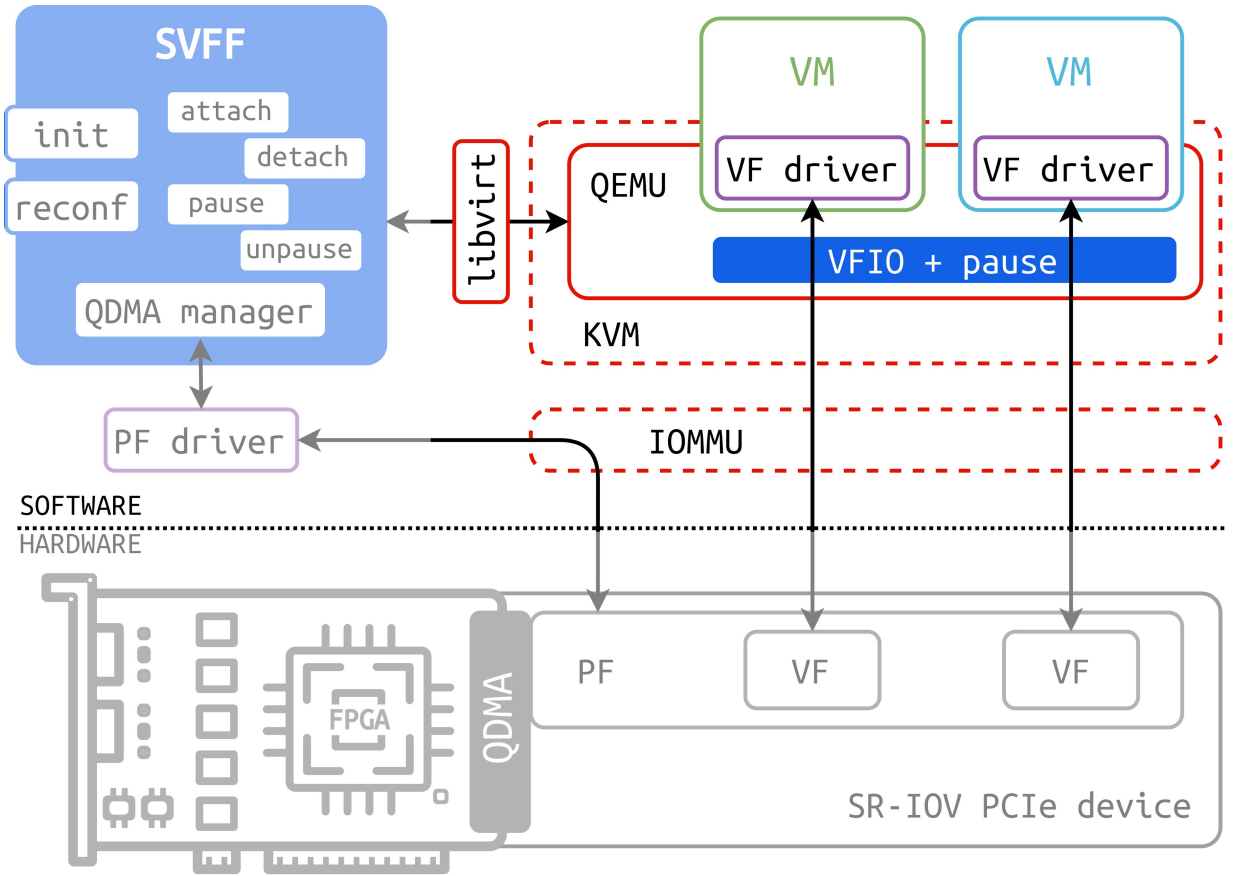
SVFF – Paused state [2]

- vfio-pci can be unloaded, allowing changing the PF/VF configuration
- The VF is then bound to the vfio-pci driver which is attached to the QEMU VFIO device
- The device is again available to the guest



SVFF – Integration

- Interacts with the PF driver to change the device configuration, including the number of VF
- Integrated with QEMU and libvirt using their APIs
- Created new QEMU QMP command to enable the pause and unpause operations



Testing and Results

Evaluate the impact of the pause functionality, comparing it to the classic attach/detach.

Minimal reduction in the delays from 2% to 2.71% with a reducing time of around 80ms per VF in all the scenarios.

TABLE I
VF DETACH-ATTACH VS PAUSE-UNPAUSE OVERHEAD, AVG OF 100 RUNS

#VF	Detach/Attach		Pause/Unpause		Overhead	
	avg ms	σ	avg ms	σ	%	ms/VF
1	4151	40	4068	56	-2.00	-83
4	12988	183	12665	171	-2.49	-80
10	31129	497	30285	505	-2.71	-84

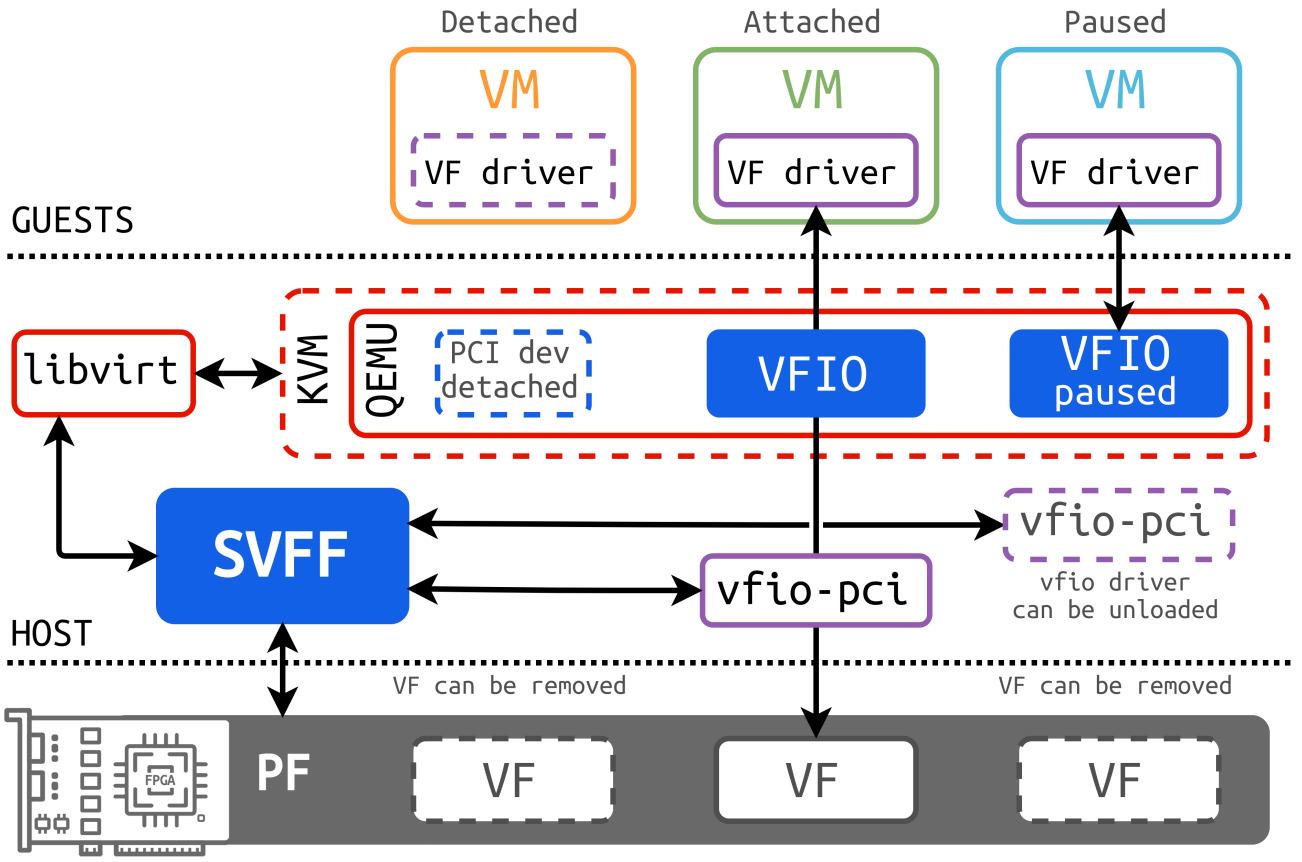
TABLE II
TIMINGS OF VF DETACH-ATTACH AND PAUSE-UNPAUSE OPERATIONS

Step	1 VF		4 VFs		10 VFs	
	D/A ms	P/U ms	D/A ms	P/U ms	D/A ms	P/U ms
1. rescan	138	138	144	141	139	139
2. remove VF	1265	1273	5417	5505	14360	13878
3. change #VF	1256	1295	1460	1412	1817	1730
4. add VF	1472	1346	5946	5653	15042	14448
total	**4131**	**4052**	**12967**	**12711**	**31358**	**30195**

Conclusions [1]

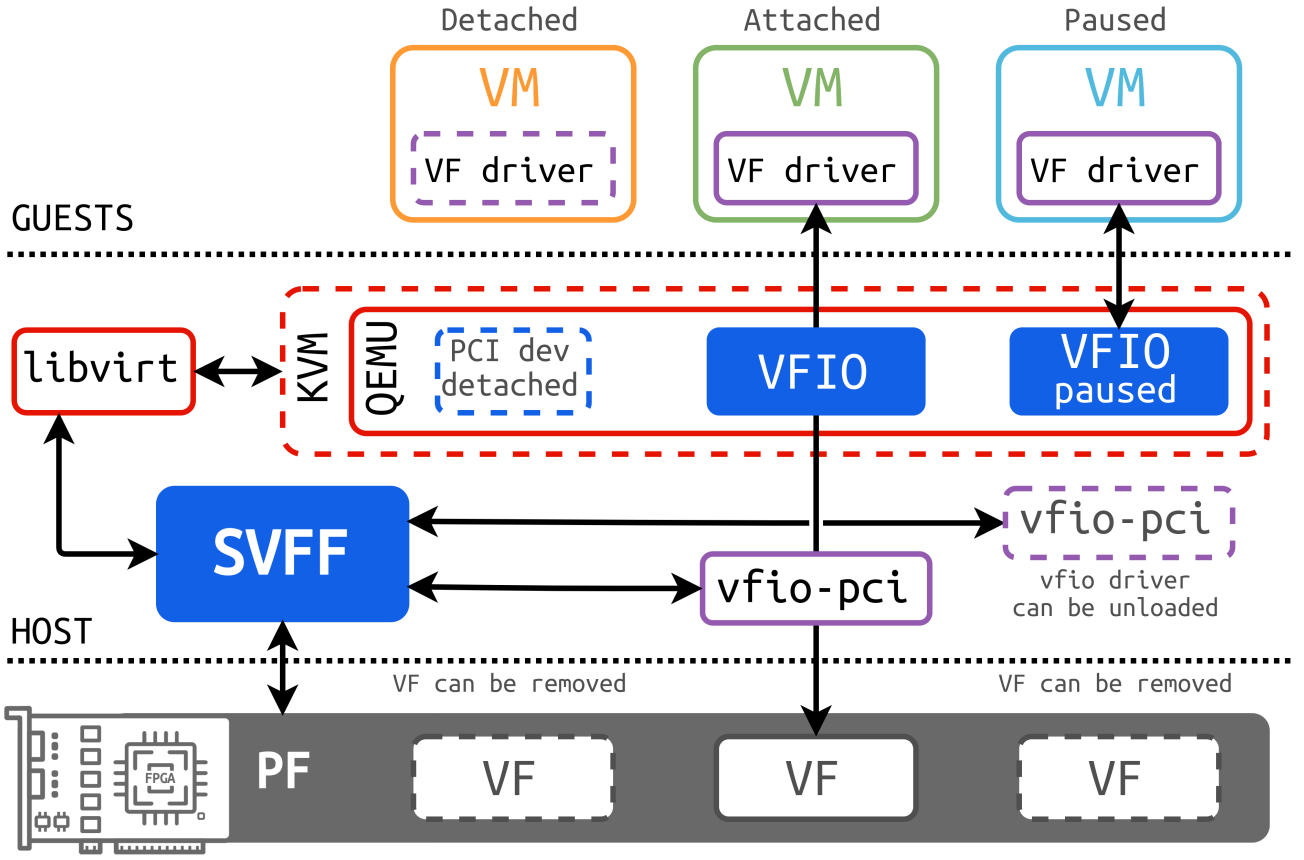
SVFF overall:

- Native performance
- Simplicity
- Compatibility
- Flexibility



Conclusions [2]

- Easily unlock FPGA virtualization potential
- Streamlines VF management
- Enhances performance, resource utilization, and overall system efficiency
- Pause functionality



Virtual Open Systems

contact@virtualopensystems.com

Web: virtualopensystems.com

Products: http://www.virtualopensystems.com/en/products/

Demos: virtualopensystems.com/en/solutions/demos/

Guides: virtualopensystems.com/en/solutions/guides/

Research projects: virtualopensystems.com/en/research/innovation-projects/