OpenCAPI™

Framework (OC-Accel), simulation engine (OCSE) and high level language (HLS)

University of Geneva

September 15th, 2021

IBM Montpellier

Presentation Outline

- Application porting at a glance
- Coding wo framework
- Open-source framework architecture
  - Ease of coding
  - Ease of moving
  - Ease of adapting
- FPGA acceleration: a 3 steps process

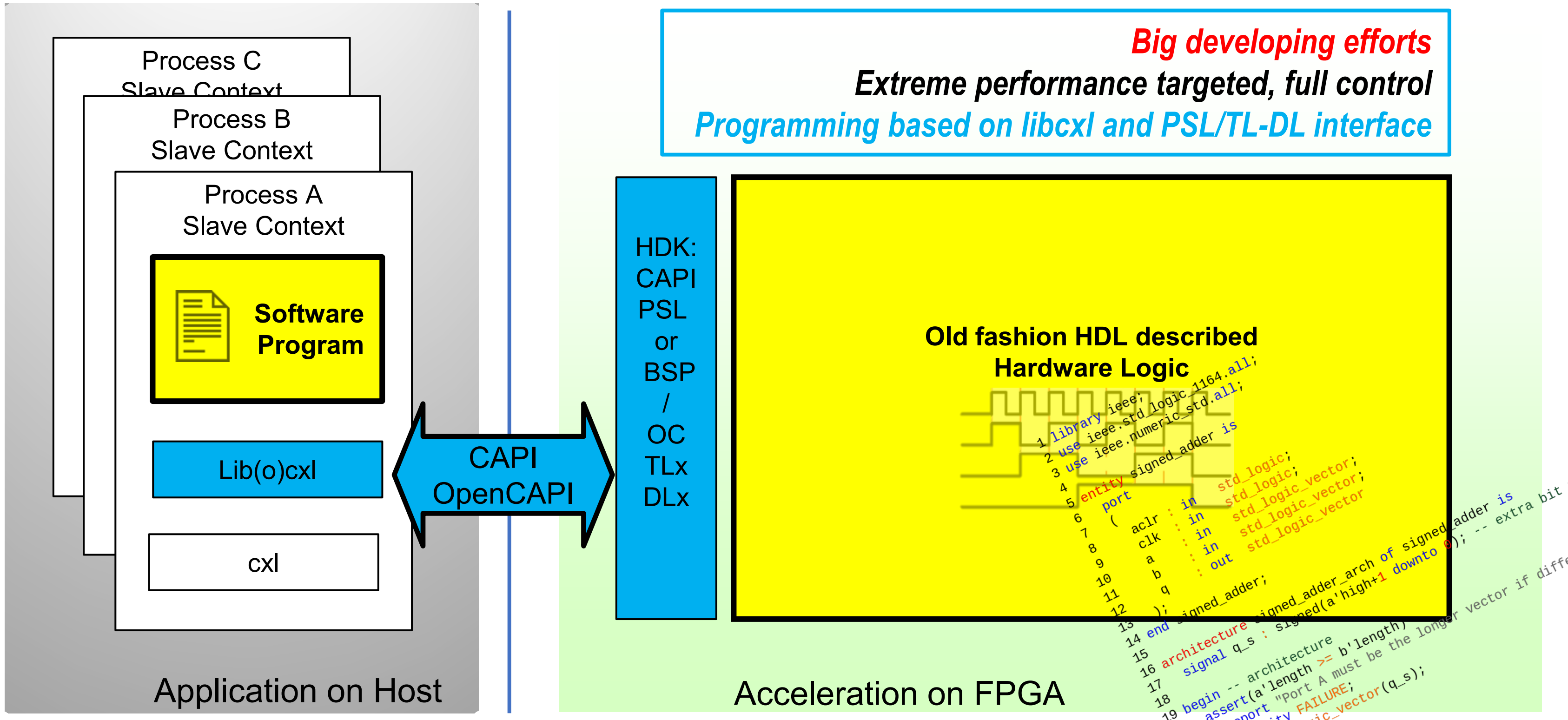
FPGA development : no framework with HDL

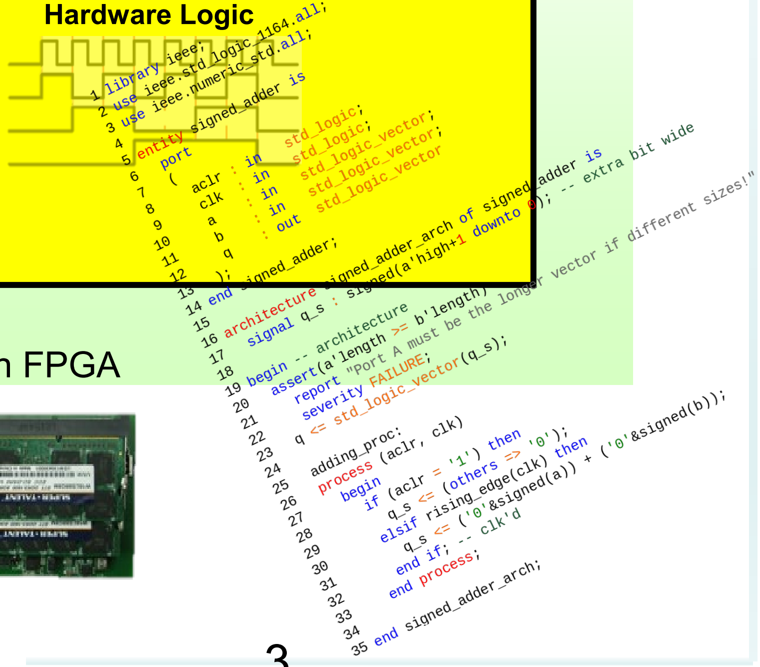
OpenPOWER™

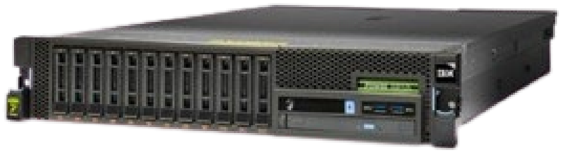
HDL : Hardware Description Langage

Develop your code

- Software side:
  - lib(o)cxl APIs
- FPGA side:
  - CAPI PSL interface
  - OpenCAPI TLx
  - Your action in HDL







Power™ Coherent Acceleration Processor Interface (CAPI)

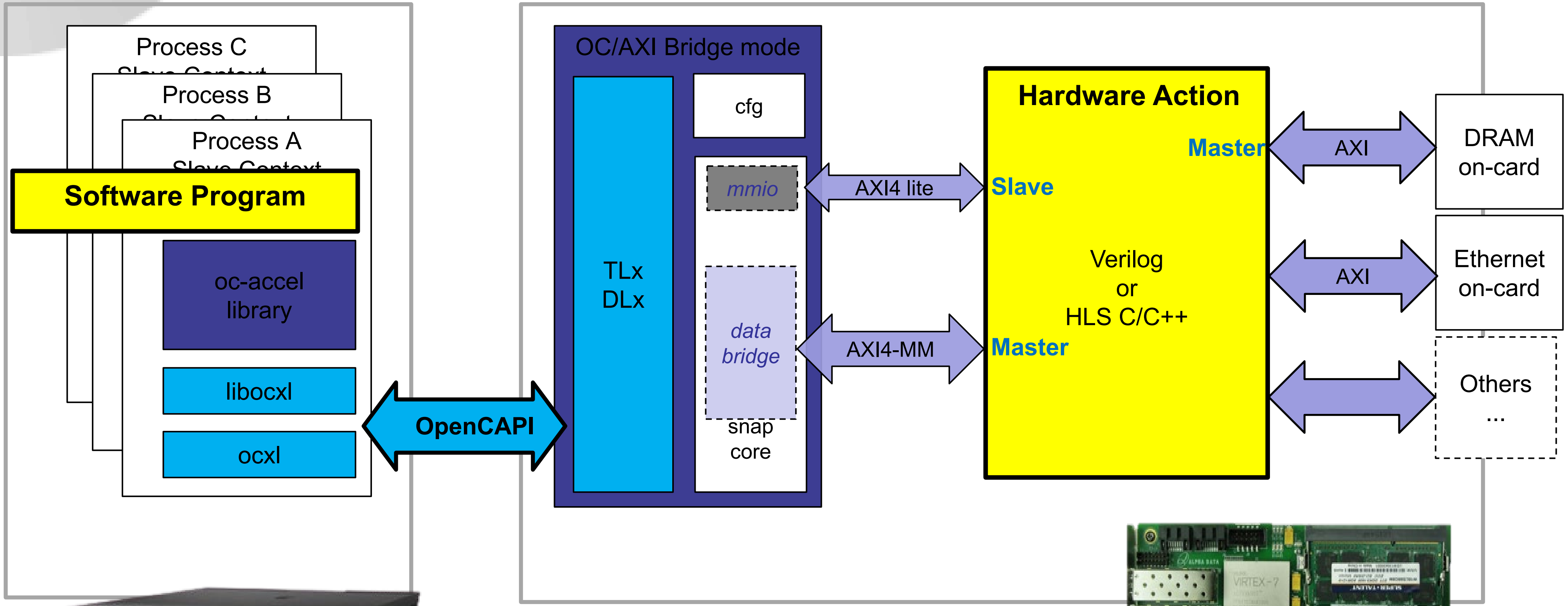
OC-ACCEL : OpenCAPI Acceleration Framework

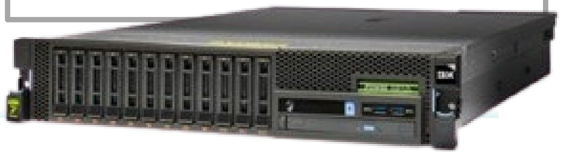
OpenPOWER™

- It is an opensource development environment like SNAP was for CAPI1&2)
- Code is at https://github.com/OpenCAPI/oc-accel
- Doc is at https://opencapi.github.io/oc-accel-doc/
- POWER Utils tools at : https://github.com/OpenCAPI/oc-utils
- How to setup a project
  - Easy to re-use CAPI1/2
  - Ease to change card or setup a new one
- How to simulate a project (simple examples)
- How to generate the FPGA flash memory content
- How to test on Power

OC-ACCEL Overview

Quick and easy development framework for OpenCAPI Accelerators





Power™ Coherent Acceleration Processor Interface (CAPI)

OC-ACCEL documentation

OpenPOWER™

https://opencapi.github.io/oc-accel-doc/

https://opencapi.github.io/oc-accel-doc/

OC-Accel Doc | About | User Guide | Examples | Deep Dive | Appl. Notes | Misc | Search | Previous | Next | GitHub

OC-Accel Overview

OpenCAPI™ Acceleration Framework

OpenCAPI Acceleration Framework, abbreviated as **OC-Accel**, the Integrated Development Environment (IDE) for creating application FPGA-based accelerators. By the nature of the FPGA, all these types of accelerators are reconfigurable.

The underlying enabling technology is OpenCAPI 3.0, the third-generation coherent interface allowing processing elements (Code running on CPU cores and accelerator logic implemented in FPGA chip) to seamlessly and coherently share host memory.

All the code and related materials are contributed to Github here: https://github.com/OpenCAPI/oc-accel.

![conceptual_system] (pictures/conceptual_system.svg) The predecessor to OC-Accel was SNAP for CAPI 2.0 and CAPI 1.0.

What is OpenCAPI

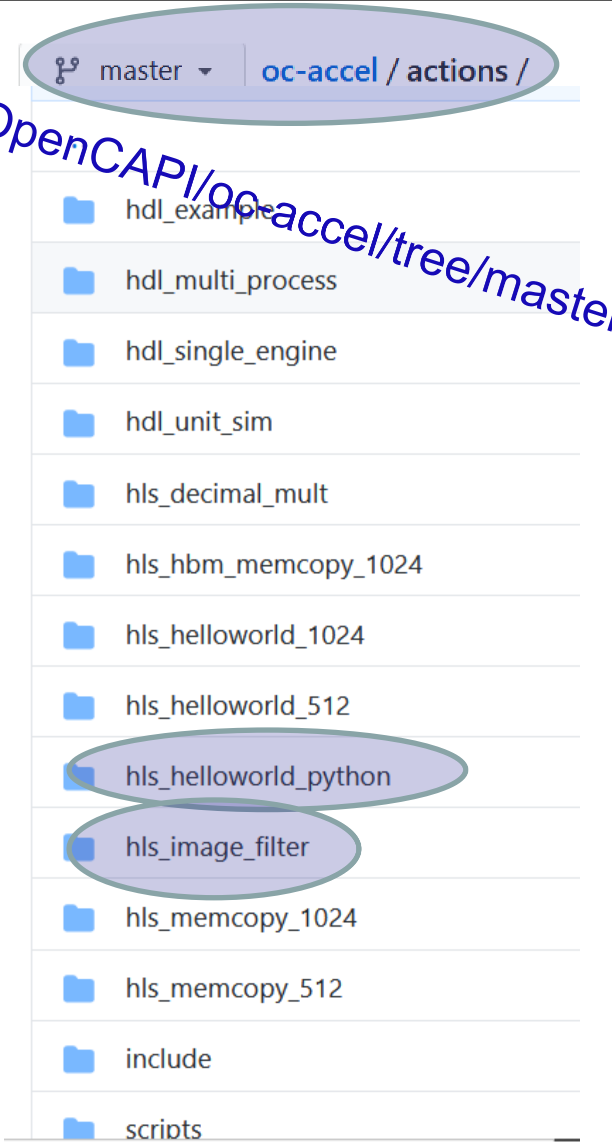
OpenCAPI (Open Coherent Accelerator Processor Interface) is the third-generation coherent interface and is an open standard for coherent high-performance bus interface. Driven by emerging, accelerated heterogeneous computing and advanced memory and storage solutions, it provides the open interface that allows any microprocessor to attach to:

- Coherent user-level accelerators and I/O devices

Oc-accel examples

OpenPOWER™

https://github.com/OpenCAPI/oc-accel/tree/master/actions



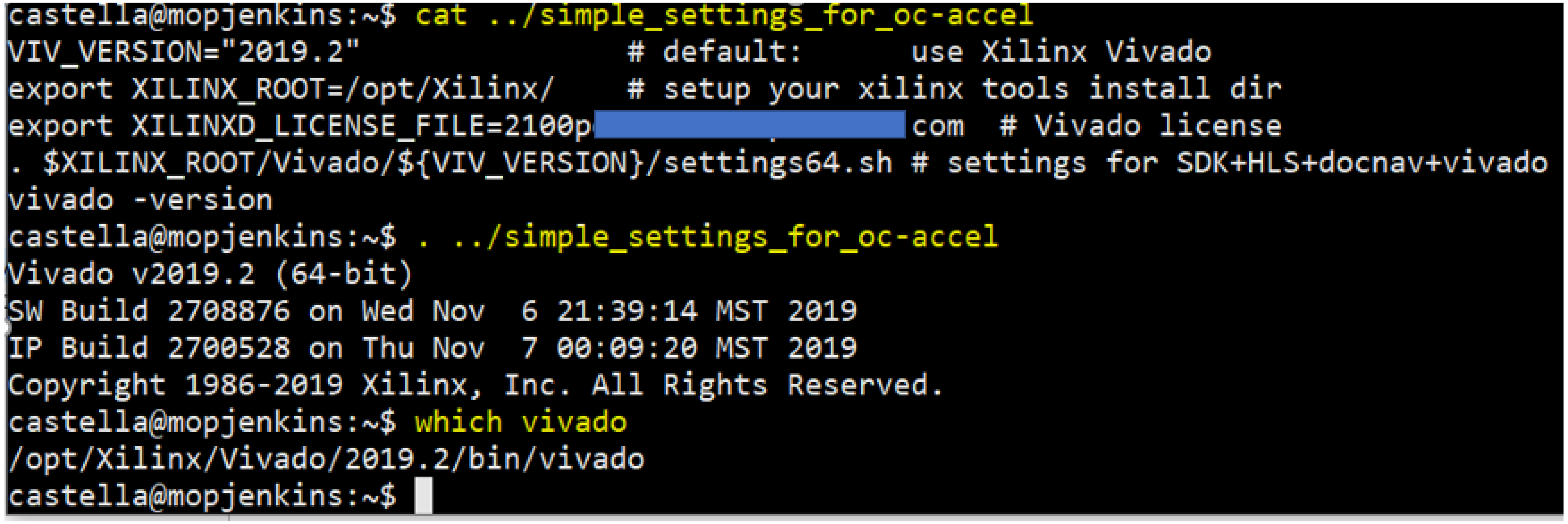
Different examples are provided
Each directory has a **/sw** with main calling application
and a **/hw** directory with the action coded either in RTL or in C/C++

We will briefly explore:

- The pixel manipulation example
- The python example

Example of Xilinx tools Setup

```
VIV_VERSION="2019.2"                  # default:      use Xilinx Vivado
export XILINX_ROOT=/opt/Xilinx    # setup your xilinx tools install dir
export XILINXD_LICENSE_FILE=2100@xxxxx.com   # Vivado license
. $XILINX_ROOT/Vivado/${VIV_VERSION}/settings64.sh # settings for SDK+HLS+docnav+vivado
vivado -version
```



```
castella@mopjenkins:~$ cat ../simple_settings_for_oc-accel
VIV_VERSION="2019.2"                  # default:      use Xilinx Vivado
export XILINX_ROOT=/opt/Xilinx/     # setup your xilinx tools install dir
export XILINXD_LICENSE_FILE=2100p[redacted]com  # Vivado license
. $XILINX_ROOT/Vivado/${VIV_VERSION}/settings64.sh # settings for SDK+HLS+docnav+vivado
vivado -version
castella@mopjenkins:~$ . ../simple_settings_for_oc-accel
Vivado v2019.2 (64-bit)
SW Build 2708876 on Wed Nov  6 21:39:14 MST 2019
IP Build 2700528 on Thu Nov  7 00:09:20 MST 2019
Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.
castella@mopjenkins:~$ which vivado
/opt/Xilinx/Vivado/2019.2/bin/vivado
castella@mopjenkins:~$
```

Configuration

OpenPOWER™

Predefined configuration, avoiding setup mistake

« make snap_config »

```
oc-image_sim : make – Konsole
File  Edit  View  Bookmarks  Settings  Help
config - Kernel Configuration
                              Kernel Configuration
 Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters
 are hotkeys.  Pressing <Y> selectes a feature, while <N> will exclude a feature.  Press <Esc><Esc> to exit,
 <?> for Help, </> for Search.  Legend: [*] feature is selected  [ ] feature is excluded

                 Card Type    (OpenCAPI3.0: AlphaData 9V3 (VU3P with DDR))  --->
                 Action Type (HLS Image Filter)  --->
                 Simulator (xsim)  --->
             (../ocse) OCSE_ROOT: Path to OpenCAPI Simulation Engine Root
                 *** ================= Advanced Options: ================= ***
                 PHY Speed (25G: Transceive IP default speed: 25.78125 Gbps)  --->
             (1) axi id width on oc-accel&action interface, range 1-5
                 Clock Frequency for Action   (Action work on 200MHz (201.420MHz to be accurate) clock)
             (4ns) The HLS clock period constraint used when C code (only used for hls action)
             [ ] Enable ILA Debug
             [*] Enable FLASH (Disabling will require JTAG programming)

                 <Select>    < Exit >    < Help >    < Save >    < Load >
```

Example of HLS usage

This is how we prepare the hardware using vivado HLS.
Two in/out streams will collect/return the data to the host mem
The pixel manipulation is described in C/C++

hw/action_pixel_filter.cpp:

```
static void strmRead(hls::stream<unsigned char> &in_stream, pixel_t *pixel )
{

#pragma HLS INLINE
#pragma HLS stream depth=16 variable=in_stream
#pragma HLS PIPELINE
        pixel->red = in_stream.read();
        pixel->green = in_stream.read();
        pixel->blue = in_stream.read();
}
```

hw/action_pixel_filter.cpp

```
static void strmInWrite(hls::stream<unsigned char> &in_stream, snap_membus_512_t
*din_gmem, action_reg *act_reg, uint64_t idx, uint32_t nbPixel )
{
        unsigned char  elt[BPERDW_512];
        uint32_t nb, done;
        int i;

        #pragma HLS INLINE // dataflow
    nb    = act_reg->Data.in.size / BPERDW_512;
    L1:
        //#pragma HLS PIPELINE
        for ( int j = 0; j < nb; j ++) {
                rBurstOfDataMem(din_gmem, (snapu64_t)idx, elt );
                L11:
                for ( i = 0; i < BPERDW_512; i++ ) {
                        #pragma HLS UNROLL factor 64
                        done = j*BPERDW_512 + i;
                        if ( done < nbPixel ) in_stream.write(elt[i]);
                }
                idx++;
        }
}
```

hw/action_pixel_filter.cpp

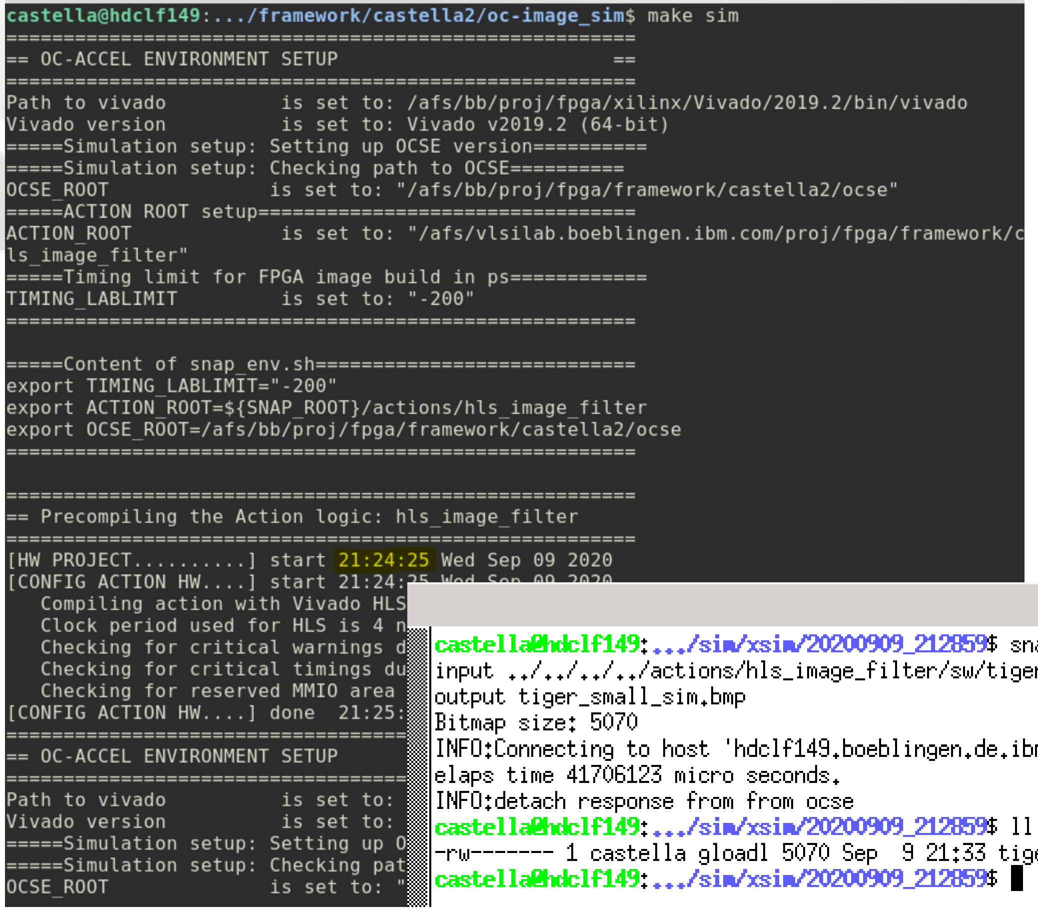
```
static void grayscale(pixel_t *pixel_in, pixel_t *pixel_out){

        uint8_t gray=(((pixel_in->red)   * RED_FACTOR)>> 8) + (((pixel_in->green) *
        GREEN_FACTOR)>> 8) + (((pixel_in->blue)  * BLUE_FACTOR)>> 8);
        pixel_out->red   = gray;
        pixel_out->green = gray;
        pixel_out->blue  = gray;

  return;
}

static void transformPixel(pixel_t *pixel_in_add, pixel_t *pixel_out_add) {

  if (pixel_in_add->red < pixel_in_add->green || pixel_in_add->red < pixel_in_add->blue)
  {

     grayscale(pixel_in_add, pixel_out_add);
     return;
  }
  else
  {
    pixel_out_add->red = pixel_in_add->red;
    pixel_out_add->blue = pixel_in_add->blue;
    pixel_out_add->green = pixel_in_add->green;
    return;
  }
}
```

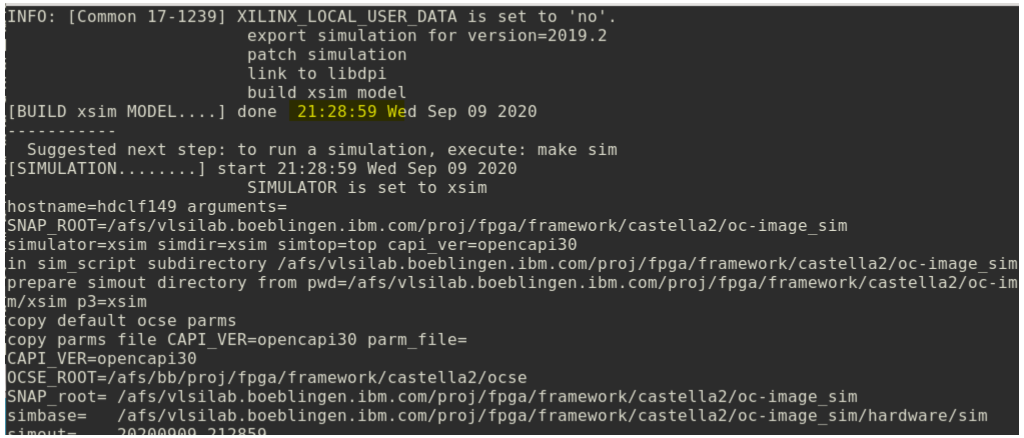
Simulation

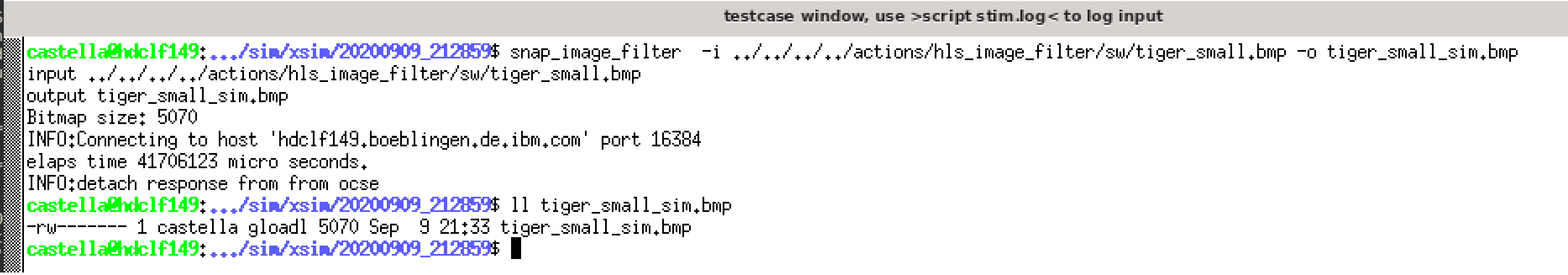
Run a simulation

« make sim »

In 5' you can simulate WITH the Host server and the actual memory



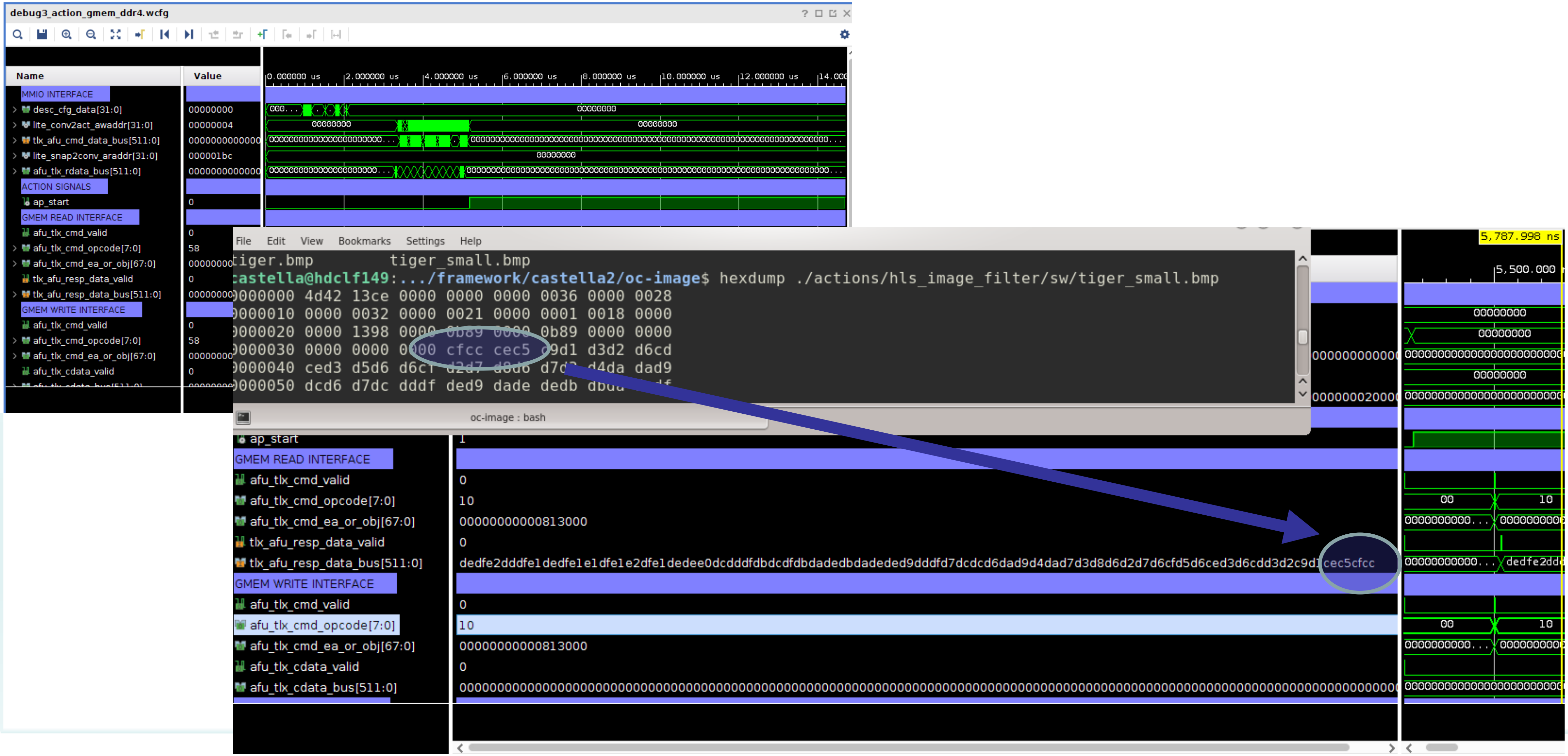




Hardware exchanges & computation analysis

OpenPOWER™

Once simulation is performed if required, you can check/debug the exact transmissions with the « ***./display_traces*** » command



Card programming

Once simulation and chronograms are satisfactory it is time to generate an image with **« *make image* »** command

This will actually prepare the synthesis of the circuitry. It takes some time

And it will provide a binary file *(in $SNAP_ROOT/hardware/build/Images/xxx.bin)* ready to be stored in the flash memory of the FPGA card

https://github.com/OpenCAPI/oc-utils

```
****** xsim v2019.2 (64-bit)
  **** SW Build 2708876 on Wed Nov  6 21:39:14 MST 2019
  **** IP Build 2700528 on Thu Nov  7 00:09:20 MST 2019
    ** Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.

start_gui
INFO: [Common 17-206] Exiting xsim at Wed Sep  9 21:52:23 2020...
make castella@hdclf149:.../framework/castella2/oc-image_sim$ mak
=======================================================
== OC-ACCEL ENVIRONMENT SETUP
=======================================================
Path to vivado          is set to: /afs/bb/proj/fpga/      ado/2019.2/bi
Vivado version          is set to: Vivado v2019.2 (64
=====Simulation setup: Setting up OCSE version==========
=====Simulation setup: Checking path to OCSE==========
OCSE_ROOT              is set to: "/afs/bb/proj/fpga/framework/castella2/ocs
=====ACTION ROOT setup=================================
ACTION_ROOT             is set to: "/afs/vlsilab.boeblingen.ibm.com/proj/fpg
ls_image_filter"
=====Timing limit for FPGA image build in ps============
TIMING_LABLIMIT         is set to: "-200"
=======================================================

=====Content of snap_env.sh============================
export TIMING_LABLIMIT="-200"
```

From X86 to POWER server

```
castella@orpington:/home/capiteam/Images/AD9V3_OC/image_filter$ sudo oc-flash-script oc_2020_0909_17
28_25G_hls_image_filter_noSDRAM_OC-AD9V3_-72_primary.bin oc_2020_0909_1728_25G_hls_image_filter_noSD
RAM_OC-AD9V3_-72_secondary.bin

===============================
== OpenCAPI programming tool ==
===============================
oc-flash_script version is 2.3
Tool compiled on: Jun 18 14:46

In this server: 1 OpenCAPI card(s) found.
Current date is Wed 09 Sep 2020 08:13:59 PM CEST

Logs shows that last programming was:
#                    Card                          Flashed                        by
   Last Image
card0:0006:00:00.0    Alphadata9V3(VU3P)           Tue 08 Sep 2020 03:57:35 PM C ST mesnet
      ./Images/AD9V3_OC/oc_2020_0908_1352_25G_hls_memcopy_512_SDRAM_OC-AD9V3_-35_primary.bin ./Image
s/AD9V3_OC/oc_2020_0908_1352_25G_hls_memcopy_512_SDRAM_OC-AD9V3_-35_secondary.bin


Which card do you want to flash? [0-0] 0

REMINDER: It is safer to CLOSE all JTAG tools (SDK, hardware_manager) before starting programming.
You will flash card0 with:
     oc_2020_0909_1728_25G_hls_image_filter_noSDRAM_OC-AD9V3_-72_primary.bin
 and oc_2020_0909_1728_25G_hls_image_filter_noSDRAM_OC-AD9V3_-72_secondary.bin
Do you want to continue? [y/n] y

Using spi x8 mode
Primary bitstream: oc_2020_0909_1728_25G_hls_image_filter_noSDRAM_OC-AD9V3_-72_primary.bin !
-------------------------------
QSPI master core setup: completed
```

Test on POWER server

Once simulation and chronograms are satisfactory, it is time to generate an flash image with « ***make image*** » command
This will actually prepare the synthesis of the circuitry. It takes some time
And it will provide a binary file ready to be stored in the flash memory of the FPGA card.

```
castella@orpington:~/oc-accel-image$ sudo ~/oc-accel/software/tools/oc_find_card -v -AALL
[sudo] password for castella:
oc_find_card version is 2.4

AD9V3 card has been detected in CAPI card position: 0
 PSL Revision is                                          : 0x6
 Device ID    is                                          : 0x0632
 Sub device   is                                          : 0x060f
 Image loaded is self defined as                          : user
 Next image to be loaded at next reset (load_image_on_perst) is : user
 Hardware Card PCI location is                            : 0030:01:00.0
 Virtual  Card PCI location is                            : 0008:00:00.0
 Card PCI physical slot is (requires sudo priv)           : SLOT0

OC-AD9V3 card has been detected in OPENCAPI card position: 0
 Device ID    is                                          : 0x062b
 Sub device   is                                          : 0x060f
 Image loaded is self defined as                          : factory
 Virtual Card PCI location is                             : 0006:00:00.1
 Card PCI physical slot is                                : Not Applicable

Total 2 cards detected
```

```
castella@orpington:~/oc-accel-image$ ./actions/hls_image_filter/sw/snap_image_filter -i ./actions/hls_image_filter/sw/tiger.bmp -o ./actions/hls_image_filter/sw/tiger_out.bmp
input ./actions/hls_image_filter/sw/tiger.bmp
output ./actions/hls_image_filter/sw/tiger_out.bmp
Bitmap size: 873234
elaps time 24023 micro seconds.
```

Bandwidth testing

- Each hls_*memcopy_* actions offers a simple performance test case to run on your P9 hardware
- Highlighted we see 17.7 GB/s from host mem to FPGA and more than 20GB/S going from FPGA to host mem.

Build Date: [00000008] 0000202009150921

OC-Accel hls_memcopy_1024 Throughput (MBytes/s)

LCL stands for DDR or HBM memory according to hardware

bytes	Host->FPGA_RAM	FPGA_RAM->Host	FPGA(LCL->BRAM)	FPGA(BRAM->LCL)
512	8.828	10.240	10.240	11.907
1024	23.814	20.480	1.484	1.476
2048	3.225	2.926	2.985	2.985
4096	5.971	6.554	6.491	80.314
8192	11.924	6.192	6.141	6.466
16384	12.337	12.911	12.921	12.870
32768	24.768	24.693	24.787	25.863
65536	49.461	95.118	92.959	102.721
131072	204.800	188.052	188.322	97.815
262144	195.484	203.055	195.193	202.741
524288	404.856	399.305	380.194	383.251
1048576	759.838	1351.258	775.574	741.567
2097152	1457.368	1408.430	1402.777	1391.607
4194304	2720.042	4185.932	4096.000	4096.000
8388608	7483.147	6732.430	6091.945	6061.133
16777216	7584.637	10292.771	6193.140	6181.730
33554432	10525.230	13584.790	9683.819	9703.422
67108864	13899.930	16615.218	10789.206	10764.977
134217728	17563.168	16927.447	11443.237	11411.131
268435456	17688.156	20650.470	11786.409	11749.265

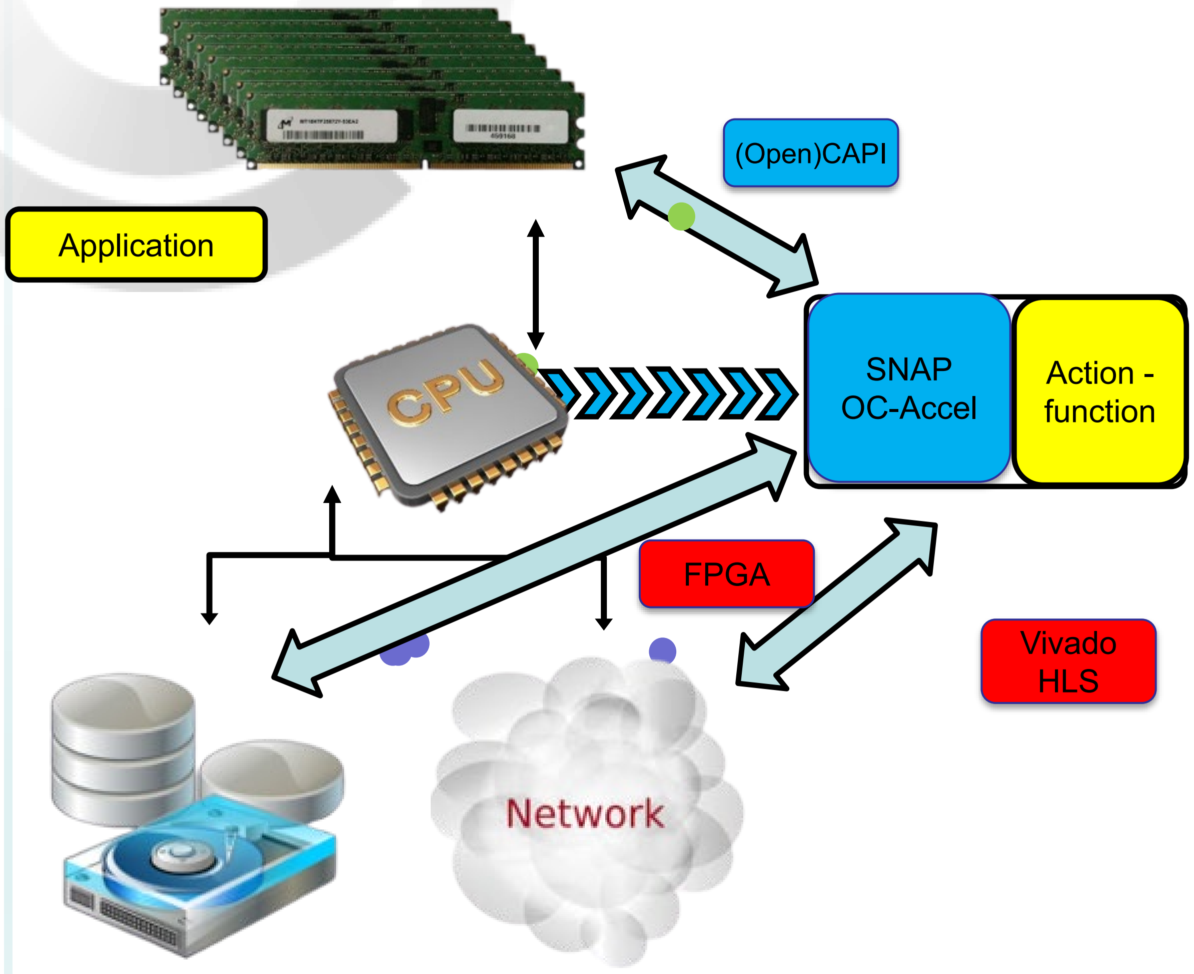
Note :

- Make sure ensure you have the OpenCAPI link attached to the core where the software is executed.
- Use numactl to control this

- Using SWIG, CURL and pip3 to ensure environment is controlled
- FPGA contains the hello_world_1024 binary (Helloworld HLS (C/C++) description reused)
- Host memory is accessed by the python, which in turn exchanges with the hardware through the OpenCAPI interface
- Can run in a Jupyter notebook

https://github.com/OpenCAPI/oc-accel/tree/master/actions/hls_helloworld_python

The CAPI SNAP/OC-Accel concept



(Open)CAPI — FPGA becomes a peer of the CPU
➔ Action **directly** accesses host memory

+

SNAP OC-Accel — Manage server threads and actions
Manage access to IOs (memory, network)
➔ Action **easily** accesses resources

+

FPGA — Gives on-demand compute capabilities
Gives direct IOs access (storage, network)
➔ Action **directly** accesses external resources

+

Vivado HLS — Compile Action written in C/C++ code
Optimize code to get performance
➔ Action code **can be ported efficiently**

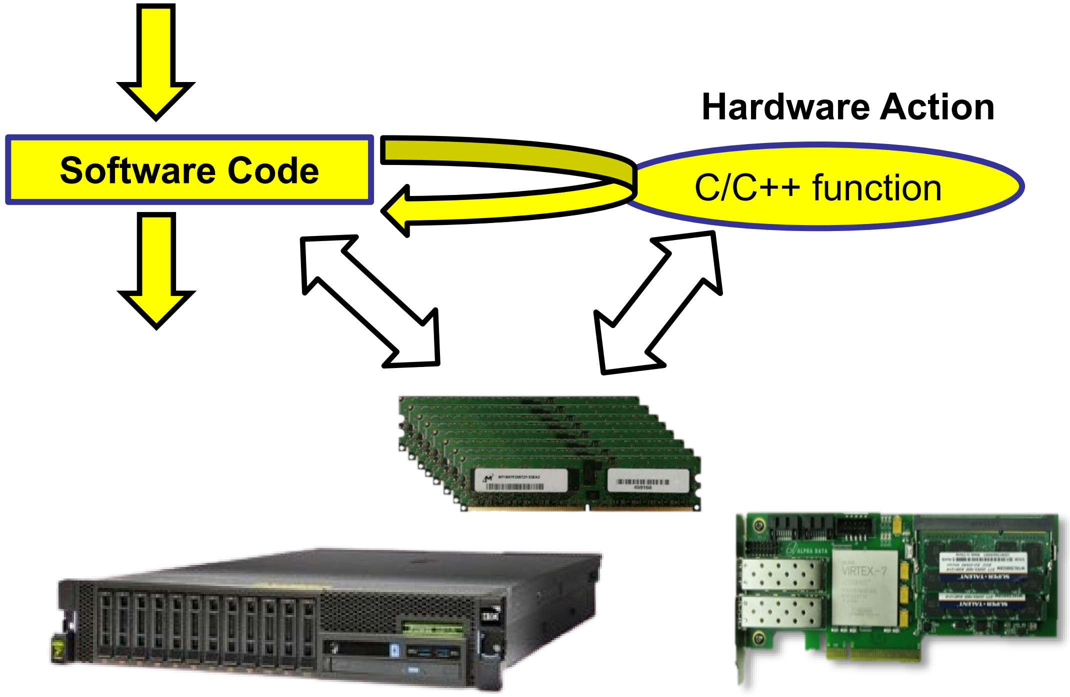
=

Offload/accelerate a C/ C++ code with :
- Quick porting
- Minimum change in code
- Better performance than CPU

2 different working modes

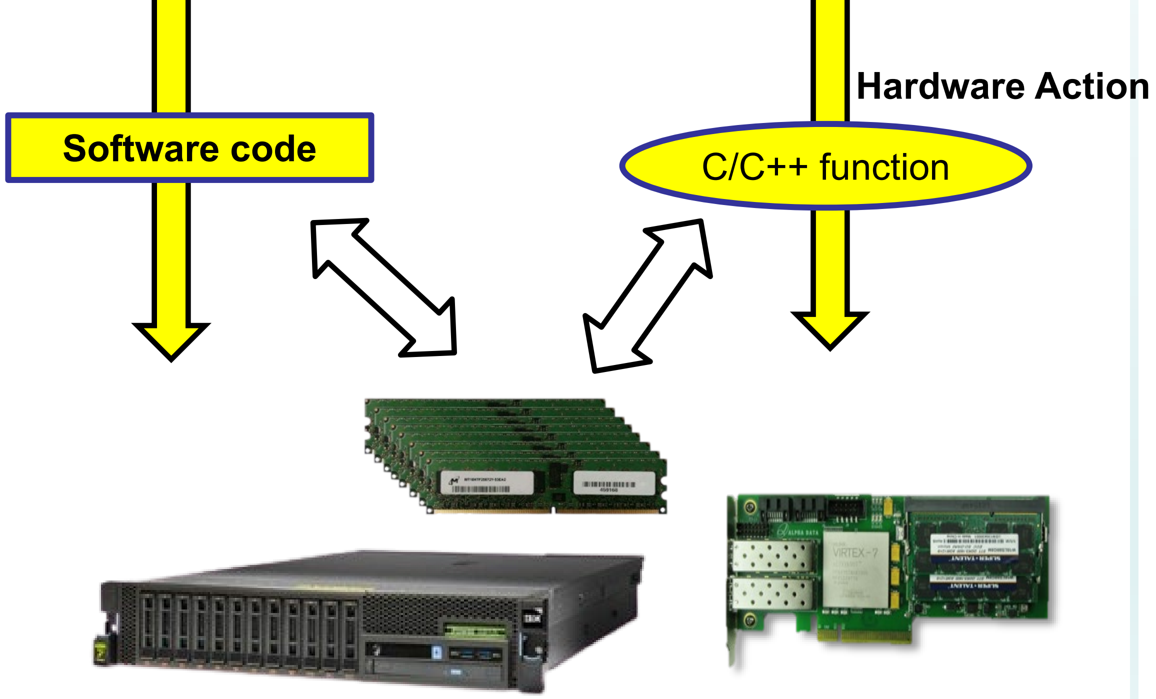
The Job-Queue Mode
SERIAL MODE

FPGA-action executes a job and returns after completion



The Fixed-Action Mode
PARALLEL MODE

FPGA-action is designed to permanently run Data-streaming approach with data-in and data-out queue



Presentation Outline

OpenPOWER™

- Application porting at a glance
- Coding wo framework
- Open-source framework architecture
  - Ease of coding
  - Ease of moving
  - Ease of adapting
- FPGA acceleration: a 3 steps process

A SIMPLE 3 STEPS PROCESS

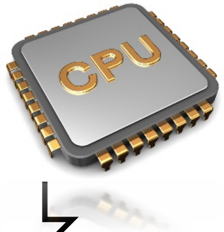
OpenPOWER™

No specific test bench required
Use your actual application

1 EXAMPLE

SNAP_CONFIG=**CPU snap_helloworld** –i /tmp/t1 -o /tmp/t2

Application **Action**



"Lower case" processing ➔ "software" action

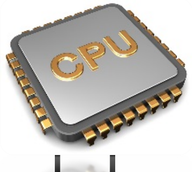
x86 server

command: **make snap_config**

2 SIMULATION

SNAP_CONFIG=**FPGA snap_helloworld** –i /tmp/t1 –o /tmp/t2

Application **Action**



FPGA Card emulation with **Power Server** IBM's **S**imulation **E**ngine

OCSE or PSLSE Instead !

"Upper case" processing ➔ "hardware" action

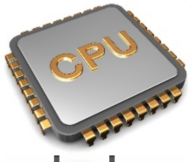
x86 server

command: **make sim**

3 EXECUTION

SNAP_CONFIG=**FPGA snap_helloworld** –i /tmp/t1 –o /tmp/t2

Application **Action**



"Upper case" processing ➔ "hardware" action

POWER8/9 server

command: **make image**

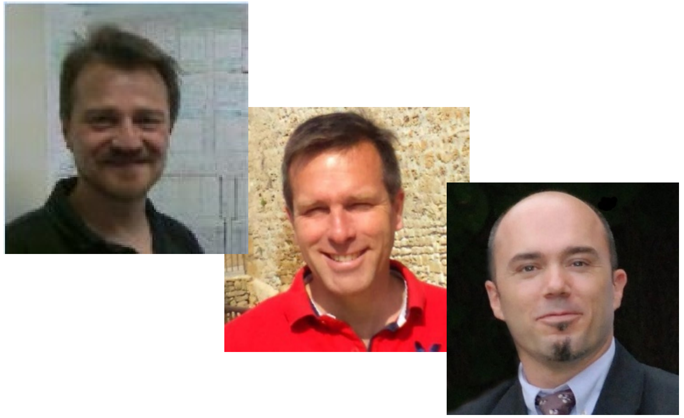
Takeaways

- **CAPI / OPENCAPI** removes the driver latency that a *classic* “FPGA + drivers” adds
- **HLS** can be easily tuned to get performances as good as low level language
- **SNAP / OC-ACCEL** follow the CAPI / OpenCAPI and FPGAs evolution without a change in user’s code
- **Open-source** helps integration with other software (libfuse…) and motivate new IPs/projects coded based on SNAP and CAPI/OpenCAPI
- **Complex C/C++ codes** *(3000 lines)* can be used for FPGA programming
- CAPI / OpenCAPI **Simulation Engines** save huge time for debuging

- *Know more about accelerators ?*
- *See a live demonstration?*
- *Do a benchmark ?*
- *Get answers to your questions?*

Contact us

alexandre.castellane@fr.ibm.com
bruno.mesnet@fr.ibm.com
fabrice_moyen@fr.ibm.com



OpenCAPI Consortium: https://www.opencapi.org

OpenCAPI Repository: https://github.com/OpenCAPI

OC-Accel Documentation: https://opencapi.github.io/oc-accel-doc/