Enable new FPGA cards for CAPI2.0

BSP and SNAP

Workgroup Notes

Revision 1.0_pre3 (August 10, 2019)

** OpenPOWER Foundation Work Group Confidential **

OpenPOWER™

www.openpowerfoundation.org

Enable new FPGA cards for CAPI2.0: BSP and SNAP

System Software Workgroup <syssw-chair@openpowerfoundation.org>
OpenPower Foundation

Revision 1.0_pre3 (August 10, 2019)

Abstract

The purpose of this document is to describe how to enable a new customer card to support CAPI SNAP framework. SNAP is a open-source programming framework for FPGA Accelerations. Its homepage is https://github.com/open-power/snap. With it, you can develop accelerators with CAPI technology easily.

This document describes the flow and steps to enable a new PCIe FPGA card to have CAPI2.0 features, and to support SNAP developing framework. If your PCIe FPGA card is not listed on today's available "SNAP enabled cards" (On the homepage README of SNAP Github), this document will guide you on **how to enable it**. Since all of the project files are open-source, you can create a Github repository fork, and create a new board support package (BSP) and walk through the working flow to enable SNAP.

This document is a Workgroup Note owned by the System Software Workgroup and handled in compliance with the requirements outlined in the *OpenPOWER Foundation Work Group (WG) Process* document. It was created using the *Master Template Guide* version 1.0.0. Comments, questions, etc. can be submitted to the public mailing list for this document at <capi-snap-doc@mailinglist.openpowerfoundation.org>.

Table of Contents

Preface vi
1. Conventions vi
2. Document change history vii
1. Introduction 1
1.1. What is CAPI 1
1.2. Enable PSL IP on FPGA 2
1.3. SNAP 3
2. Enable CAPI2.0 BSP 5
2.1. Structure 5
2.2. Step-by-step guidance 6
3. Enable CAPI2.0 SNAP 10
3.1. Work on github 10
3.2. SNAP structure 10
3.3. Modifications to snap git repositories 12
3.4. Update capi-utils 14
3.5. Strategy to enable a new card 14
3.6. Cleanup and submit 18
A. OpenPOWER Foundation overview 20
A.1. Foundation documentation 20
A.2. Technical resources 20
A.3. Contact the foundation 21

List of Figures

1.1. CAPI basic concept 1
1.2. Develop an accelerator in HDK mode 2
1.3. Develop an acceleration on SNAP 3
2.1. Project hierarchy for HDK mode 5
3.1. Project hierarchy for SNAP 10
3.2. Repository structure 11
3.3. Vivado Lab Edition 15
3.4. Add Configuration Memory Device and Program the flash 15

List of Tables

2.1. Information to collect 6
3.1. Config files to change 13
3.2. RTL/xdc/tcl files to change 13
3.3. Software files to change 13
3.4. Other files to change 13

Preface

1. Conventions

The OpenPOWER Foundation documentation uses several typesetting conventions.

Notices

Notices take these forms:

Note

A handy tip or reminder.

Important

Something you must be aware of before proceeding.

Warning

Critical information about the risk of data loss or security issues.

Changes

At certain points in the document lifecycle, knowing what changed in a document is important. In these situations, the following conventions will used.

- *New text will appear like this.* Text marked in this way is completely new.
- ~~Deleted text will appear like this.~~ Text marked in this way was removed from the previous version and will not appear in the final, published document.
- Changed text will appear like this. Text marked in this way appeared in previous versions but has been modified.

Command prompts

In general, examples use commands from the Linux operating system. Many of these are also common with Mac OS, but may differ greatly from the Windows operating system equivalents.

For the Linux-based commands referenced, the following conventions will be followed:

$ prompt Any user, including the `root` user, can run commands that are prefixed with the $ prompt.

 # prompt The `root` user must run commands that are prefixed with the # prompt. You can also prefix these commands with the **sudo** command, if available, to run them.

Document links

Document links frequently appear throughout the documents. Generally, these links include a text for the link, followed by a page number in parenthesis. For example, this link, Preface [vi], references the Preface chapter on page vi.

2. Document change history

This version of the guide replaces and obsoletes all earlier versions.

The following table describes the most recent changes:

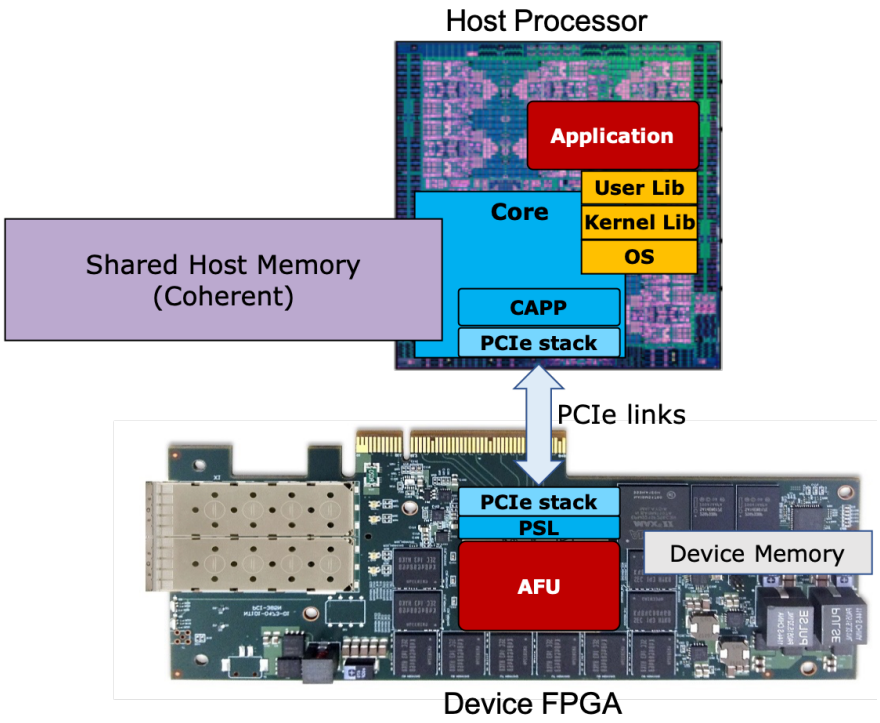
Revision Date	Summary of Changes
March 29, 2019	• Start from the original Word document

1. Introduction

1.1. What is CAPI

CAPI stands for "Coherent Accelerator Processor Interface" which enables FPGA to access Host memory by virtual address. You can find more introduction about this interface on https://developer.ibm.com/linuxonpower/capi/. It is an important feature to develop hardware accelerators in heterogeneous computing. In this document, the "hardware accelerators" are built on FPGA.

Figure 1.1. CAPI basic concept



A complete accelerator has software part (APP, or Applications) running on CPU Processor and the hardware part (AFU, Acceleration Function Unit) running on FPGA chip. APP and AFU are sharing host memory, that means, they both can read and write the 2^64 range of virtual memory address. To make it happen, CAPI technology has a CAPP (Coherent Acceleration Processor Proxy) logic unit in Processor chip, and also needs a PSL (Processor Service Layer) logic unit in FPGA chip. For CAPI1.0 and CAPI2.0, the interconnection between processor and FPGA is using PCIe physical links and PCIe form factor.

CAPI1.0 uses PCIe Gen3x8.

CAPI2.0 uses PCIe Gen4x8 or Gen3x16.

OpenCAPI is not covered in this document. Visit https://opencapi.org for more information.

1.2. Enable PSL IP on FPGA

This document only applies to the cards using Xilinx FPGA chips.

A customer FPGA card needs to have a PSL module (Processor Service Interface) to become a "CAPI-enabled" card. This PSL module is provided by OpenPower Foundation.

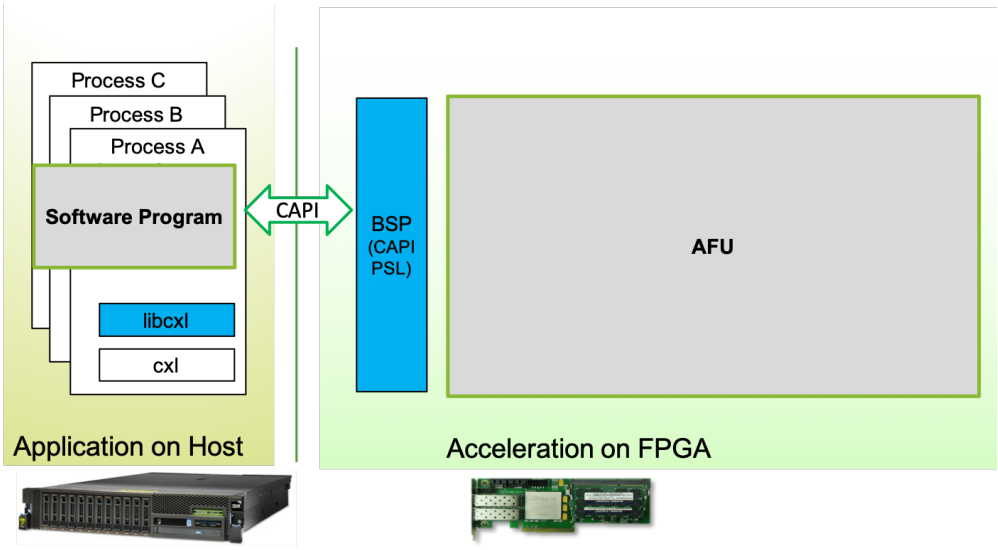
- For CAPI1.0, PSL module and the surrounding board specific modules are provided in the form of a routed dcp file (Xilinx Vivado design checkpoint). It's usually called b_route_design.dcp.
- For CAPI2.0, PSL is an IP package with encrypted source code. It's named like ibm.com_CAPI_PSL9_WRAP_2.00.zip.

They can be downloaded at https://www.ibm.com/systems/power/openpower. From the menu, select "CAPI","Coherent Accelerator Processor Interface (CAPI)" or directly click the "CAPI" icon to go to the CAPI section. Then download the appropriate files depending on your target system being POWER8 (CAPI 1.0) or POWER9 (CAPI 2.0). You need to register an IBM ID to download them.

Users can develop CAPI accelerators in two modes: HDK and SNAP.

HDK is the abbreviation of Hardware Development Kit. As shown in the diagram below, on the FPGA side, the Xilinx Vivado project includes two parts: BSP (Board Supporting Package, containing PSL module) and AFU (Acceleration Function Unit). How to generate BSP will be introduced in Chapter Enable CAPI2.0 BSP [5]

Figure 1.2. Develop an accelerator in HDK mode



AFU is where to implement user-defined functions. The developer working on AFU needs to understand the protocol between AFU and BSP, which is called PSL/AFU interface specification. Please refer to CAPI1.0 PSL Spec and CAPI2.0 PSL Spec or search "PSL/AFU interface" in your web browser.

When developing an acceleration, PSLSE (PSL Simulation Engine) is also needed to make a software-hardware co-simulation which guarantees the correctness of accelerator design.

When deploying the acceleration to OpenPower servers, it requires user library libcxl and kernel module **`cxl`** to run the application.

In all, HDK mode will provide the maximum control, utilization of resources and shortest latency. However, SNAP mode simplifies and standardizes the application development significantly and is more recommended.

1.3. SNAP

SNAP is the abbreviation of Storage, Networking and Analytics Programming. It is an open-source acceleration development framework https://github.com/open-power/snap. The benefits are:

1. On the FPGA side, SNAP framework adds a bridge to provide AXI interface to developers. So the developer can focus on acceleration function logic design, and doesn't need to study the details of PSL interface specification. AXI is the defacto industry standard for on-chip bus interconnections and is part of AMBA (Advanced Microcontroller Bus Architecture).

2. It also provides DDR SDRAM controller and an optional NVMe controller. The developer can use the card memory or storage directly.

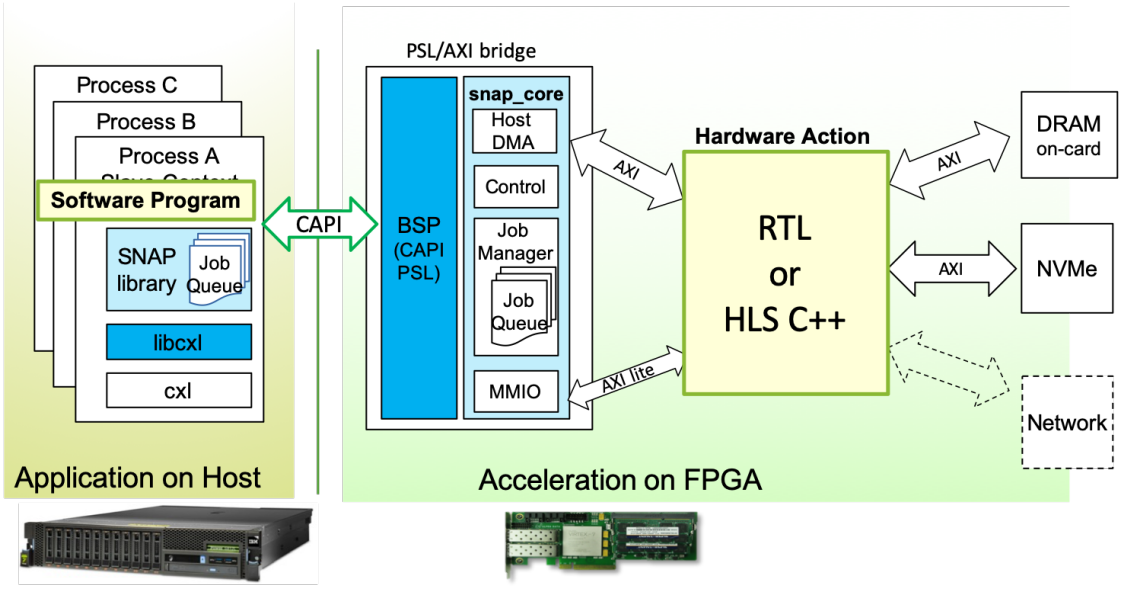
3. SNAP supports using HLS (High Level Synthesis) to develop the acceleration functional unit ("Hardware Action" in yellow box). Developers can write C++ functions and Vivado HLS will compile/convert them to Verilog or VHDL design automatically.

4. A new layer of user library "libsnap" provides more convenient APIs.

5. SNAP is an integrated developing environment that the developer can configure, create Vivado project, run co-simulation or build bitstream with simple commands.

6. Many action examples help new developers to get started.

Figure 1.3. Develop an acceleration on SNAP



Equipping the new FPGA card with SNAP framework needs a few additional steps and is introduced in Chapter Enable CAPI2.0 SNAP [10]

Note

This document focuses on CAPI2.0. For CAPI1.0 enablement, please contact <capi-snap-doc@mailinglist.openpowerfoundation.org> for more information.

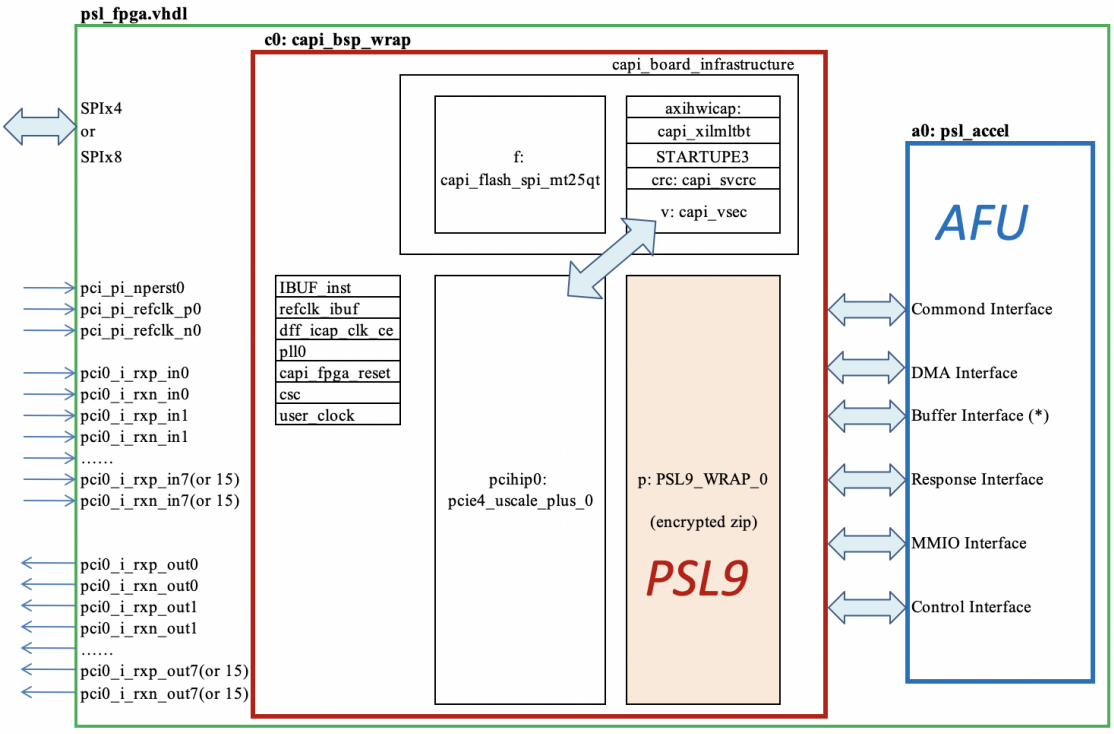
It is assumed the reader knows how to work on Vivado Project and SNAP already. Many materials on how to develop an accelerator with SNAP can be found in "docs" folder of snap github or other webpages so they are not discussed in this document.

2. Enable CAPI2.0 BSP

2.1. Structure

Each card supplier may design their FPGA board with different FPGA chips, circuit components, memory and IOs, so BSP (Board support package) is different from card to card. That's why an open-sourced project is so helpful: It allows card supplier and every developer to explore the functions of the card freely, and get benefits from CAPI technology.

Figure 2.1. Project hierarchy for HDK mode



CAPI2.0 BSP (capi_bsp_wrap) contains following modules:

- PSL9 (PSL9_WRAP)
- PCIe hard IP core (pcie3/4_ultrascale_plus_0)
- Flash Controller (capi_flash_spi_mt25qt)
- VSEC: Vendor Specific Extended Capability (capi_vsec)
- Xilinx MultiBoot control logic (capi_xilmltbt)
- Other miscelleneous logic

In this diagram, psl_fpga.vhdl is the top design. How to connect the ports to FPGA pins is different from card to card, and the information is provided by Card Supplier. They may include:

- Flash interface (usually SPIx4 or dual SPIx4)

- PCIe interface: perst, refclk, TX and RX data lanes (PCIe Gen3.0x16, or Gen4.0x8)
- Peripheral IPs: I2C, LED, DDR, Ethernet, etc.

At least Flash interface pins and PCIe interface pins are required to be assigned in xdc files precisely.

Between capi_bsp_wrap and user AFU (psl_accel), there are 6 groups of signals: Command, DMA, Buffer, Response, MMIO and Control. Please refer to CAPI2.0 PSL/AFU interface Spec for the details.

2.2. Step-by-step guidance

2.2.1. Work on github

capi2-bsp is a public Github repository. You need to have a Github account first. Then create a "fork" (Click the "fork" button) on https://github.com/open-power/capi2-bsp.

```
git clone https://github.com/[YOUR_USERNAME]/capi2-bsp
```

Note

capi2-bsp is also a submodule of snap.

Keep working on your own capi2-bsp fork, when it has been validated to work well, submit a pull request to "open-power/capi2-bsp" and request merging into the public upstream.

2.2.2. Preparations

First, define a FPGACARD name. It can start from the company's name, following with the card name and be short. For example, ADKU3 = Alpha-Data ADM-PCIE-KU3. Get information from the card supplier.

Table 2.1. Information to collect

Item	Description
FPGACARD	Short card name
FPGACHIP	FPGA part name, for example, xcvu9p-fsgd2104-2L-e
Flash Type and IO pins	Flash chip that attached to FPGA, for example mt28gu01gaax1e-bpi-x16. And the related xdc files for FPGA config.
PCIe Config and IO pins	The tcl/xdc information about the PCIe hardware IP for this card.
DDR MC IP	DDR memory controller Vivado IP tcl/xdc file.
Other peripherals	NVMe IP, Ethernet IP and so on (Optional)

Note

DDR MC and other peripheral IP's configurations are not needed at the beginning. They are not in the capi_bsp_wrap diagram. However, when you query information from the FPGA Supplier, it's wise to include them also.

2.2.3. Download PSL Zip package

Download PSL9 Zip package from OpenPower Portal and put it in "capi2-bsp/psl" directory.

2.2.4. Copy and modify

Choose an existing card as a base. Copy the folder to your new FPGACARD name. Then modify the contents according to the information collected from FPGA supplier.

Important

Make sure the information in xdc/tcl files are permitted to be open-source.

There are some other modifications you should pay attention to:

1. PCIe core IP creation:

   - "Vendor ID" and "Device ID" have to be 0x1014 and 0x0477, so kernel module cxl can recognize the card as a CAPI device. (See in pci.c)
   - If the card vendor has a code allocated by PCISIG (See in PCISIG Member companies), use it as "Subsystem Vendor ID". "Subsystem Device ID" can be chosen freely.
   - If the card vendor doesn't have a code allocated by PCISIG, or just for testing and evaluation purpose, please use default "Subsystem Vendor ID" = 0x1014, and send email to <aclwg-chair@openpowerfoundation.org> to get a distinct "Subsystem Device ID" to differentiate this card from others.

     Example: (in create_ip.tcl)

     ```
     create_ip -name pcie4_uscale_plus -vendor xilinx.com -library ip -module_name
      pcie4_uscale_plus_0 -dir $ip_dir >> $log_file
     set_property -dict [list                                                \
                         CONFIG.PF0_CLASS_CODE {1200ff}                      \
                         CONFIG.PF0_REVISION_ID {02}                         \
                         CONFIG.VENDOR_ID {1014}                             \
                         CONFIG.PF0_DEVICE_ID {0477}                         \
                         CONFIG.PF0_SUBSYSTEM_VENDOR_ID {1014}               \
                         CONFIG.PF0_SUBSYSTEM_ID {0661}                      \
                         ......                                              \
                         ......                                              \
                        ] [get_ips pcie4_uscale_plus_0] >> $log_file
     ```

     The corresponding "Subsytem Vendor ID" and "Subsystem Device ID" need to be added into capi-utils, file "psl-devices".

2. Product Family support:

   If the FPGA chip types are Xilinx VU33P or VU37P who have HBM, this is actually a new FPGA family **`virtexuplushbm`**. For a new FPGA Production family, additional steps need to take:

   - "capi2-bsp/psl/create_ip.tcl": **`set_property supported_families ...`**, add new family name like "virtexuplushbm Production"
   - "capi2-bsp/common/tcl/create_capi_bsp.tcl": **`set_property supported_families ...`**, do the same as above.
   - Add family support to PSL9 ZIP package: unzip the package, do the modifications, and zip them back again. Commands:

```
$ unzip ibm.com_CAPI_PSL9_WRAP_2.00.zip
(modify compnent.xml to add new family name, search "supportedFamilies")
$ zip -r ibm.com_CAPI_PSL9_WRAP_2.00.zip component.xml src/ xgui/
$ rm -fr component.xml src/ xgui/
```

3. VSEC starting address:

   VSEC (Vendor Specific Extended Capability Structure) is a part of PCIe capability list architecture. It needs to be properly linked in PCIe config space.

   "capi2-bsp/[FPGACARD]/src/capi_vsec.vhdl": **`vsec_addr[21:32]`** defines the address for VSEC. It should be matched with PCIe core value **`PF0_SECONDARY_PCIE_CAP_NEXTPTR`**. Take card U200 for example, its **`vsec_addr[21:32]`** starts from 12'h400 (12'b0100_0000_0000), and "tcl/patch_ip.tcl" modifies it from default value 12'h480 to 12'h400."

```
exec /bin/bash -c "sed -i \"s/PF0_SECONDARY_PCIE_CAP_NEXTPTR=0x480/
PF0_SECONDARY_PCIE_CAP_NEXTPTR=0x400/\" $pcie_source"
exec /bin/bash -c "sed -i \"s/PF0_SECONDARY_PCIE_CAP_NEXTPTR('H480)/
PF0_SECONDARY_PCIE_CAP_NEXTPTR('H400)/\" $pcie_source"
```

   Xilinx PCIe code information for extended configuration space can be found on PG156 (for Ultrascale Device) or PG213 (for Ultrascale+ Device).

   For Ultrascale+ HBM device's pcie4c core, the VSEC starts from 12'hE80. At this time **`vsec_addr[21:32]`** must be changed in "capi_vsec.vhdl". And the above two lines in "patch_ip.tcl" are not needed anymore.

4. Vital Product Data: **This step is optional.**

   "capi2-bsp/[FPGACARD]/src/capi_vsec.vhdl": Edit the hardcoded **`vpd44data_be`** to add VPD (Vital Product Data) information. Ideally this information should be read from an I2C EEPROM. The FPGA supplier wrote the content of EEPROM before shipping. Today the simpliest way is taken -- writing some hard coded value. "capi2-bsp/common/script" has a script "gen_vsec.sh" to do this.

5. User Image Address:

   "capi2-bsp/[FPGACARD]/src/capi_xilmltbt.vhdl": Edit the User image starting address **`wbstart_addr`**.

```
wbstart_addr <=  "User_image_address"  when (cpld_user_bs_req = '1') else
 "00000000000000000000000000000000";
```

   "capi_xilmltbt.vhdl" has a Xilinx multi-boot core. That means you can create two kinds of images: Factory image and User image. Factory images will be placed at address 0 of FPGA Flash, and User image will be placed at "User_image_address" on the flash. When power-on or the FPGA card is reset, the multiboot core knows where to load the image. Usually a Golden factory image is put on address 0 and is never changed. Multiboot core always tries to load user image first. If the user image has something wrong, multiboot logic will tell the FPGA to "fallback" to factory image. You still see the card in the system and you can just program a new user image to try again.

6. Check Vivado Version:

Make sure this version of Vivado tool supports the FPGA part name you have assigned in "capi2-bsp/[FPGACARD]/Makefile". For some very new FPGA chip types, in one Vivado version they may have a suffix of "es" (engineering sample), and in a newer Vivado version the "es" suffix is removed.

2.2.5. Generate capi_bsp_wrap

```
cd capi2-bsp
make [FPGACARD]
```

If it is successfully done, the generation of BSP for CAPI2.0 is completed. Developers using HDK mode can create their own Vivado project and import capi_bsp_wrap as an IP. But for SNAP developers there are some other work to do, see in next chapter.

3. Enable CAPI2.0 SNAP

3.1. Work on github

Snap is also a public Github repository. Create a "fork" (Click the "fork" button) on https://github.com/open-power/snap. Keep working on your own snap fork, when it works, submit a pull request to "open-power/snap" and require merging into the public upstream.

```
git clone https://github.com/[YOUR_USERNAME]/snap
```

Note

capi2-bsp is a submodule of snap. It is shown in ".gitmodules" file (this is a hidden file). Please point it to your own capi2-bsp fork. Then

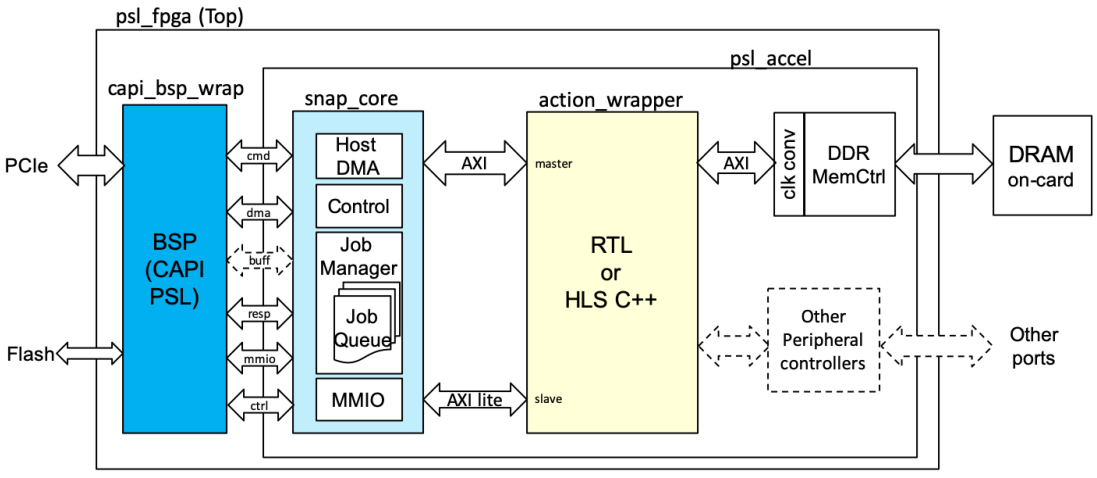
```
git submodule init
git submodule update
```

Anyway, make sure that "hardware/capi2-bsp" is the one just generated in last chapter.

3.2. SNAP structure

On the FPGA side, there are three parts that need to consider when moving to a new FPGA card. They are (a) BSP, (b) snap_core, (c) DDR memory controller (mig). And there are also some components in SNAP need to be updated for a new FPGA card.

Figure 3.1. Project hierarchy for SNAP

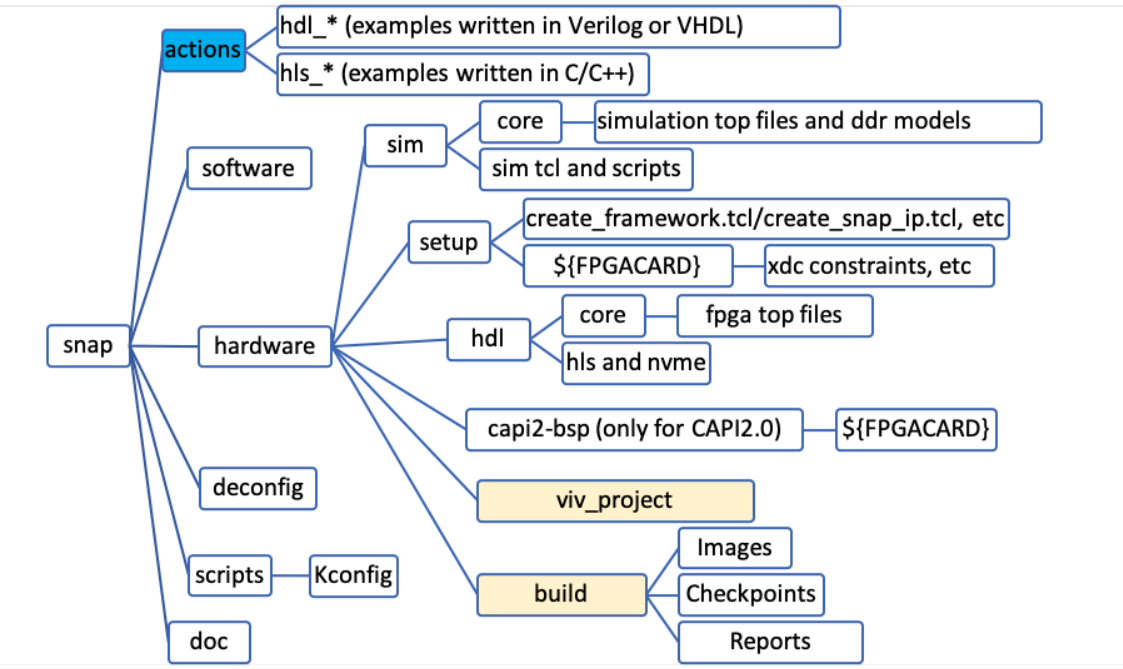


Note

Module **snap_core** on CAPI2.0 implemented the data path with DMA interface. Buffer interface is not used.

The following picture shows the SNAP github repository folders and files.

Figure 3.2. Repository structure



All of the user-developed accelerators are in "**actions**" directory. There are already some examples there. Each "action" has its "sw", "hw", "tests", and other sub-directories. The hardware part uses "action_wrapper" as its top.

"**software**" directory includes libsnap, header files and some tools. "**hardware**" directory is the main focus. "**deconfig**" has the config files for silent testing purpose, and "**scripts**" has the menu settings and other scripts.

How does SNAP work and what are the files used in each step?

- **`make snap_config`**: The menu to select cards and other options is controlled by "script/Kconfig"
- **`make model`**: This step creates a Vivado project. It firstly calls "hardware/setup/create_snap_ip.tcl" to generate the IP files in use, then calls "hardware/setup/create_framework.tcl" to build the project. About "create_framework.tcl":
  - It adds BSP (board support package). In CAPI1.0, it is also called PSL Checkpoint file (b_route_design.dcp) or base_image. It uses the path pointed to b_route_design.dcp and adds it into the design. In CAPI2.0, it will call the make process in capi2-bsp submodule to generate "capi_bsp_wrap" if it doesn't exist. This step is skipped if "capi_bsp_wrap" is already generated. Then "create_framework.tcl" adds the capi_bsp_wrap (xcix or xci file) into the design.
  - It adds FPGA top files and snap_core files (in hardware/hdl/core).
  - It adds constrain files: in "hardware/setup/[FPGACARD]" or in "hardware/capi2-bsp/[FPGACARD]"

  - It adds user files (in "actions/[ACTION_NAME]/hw"). User's action hardware uses top file named "action_wrapper.vhd"
  - It adds simulation files (in "hardware/sim/core") including simulation top files and simulation models. (If **`no_sim`** is selected in snap_config menu, this step is skipped.)

  After above steps, "**hardware/viv_project**" is created. Open it with Vivado GUI, and check the design hierarchy. And it will call the selected simulator to compile the simulation model.

- **`make image`**: This step runs synthesis, implementation and bitstream generation. It calls "hardware/setup/snap_build.tcl" and also uses some related tcl scripts to work together. In this step, "**hardware/build**" will be created and the output products like bit images, checkpoints (middle products for debugging) and reports (reports of timing, clock, IO, utilization, etc.) If everything runs well and timing passes, user will get the bitstream files (in "build/Images" sub directory) to program the FPGA card.

3.3. Modifications to snap git repositories

For a new FPGA card, the detailed items to update are listed as below.

- Hardware RTL, setup, simulation
- Software and tools
- Testing
- Publishing

The best way is to grep some keywords like "S241" or "AD8K5" under the directories and look for the locations that need modifications.

Note

A file ending with "_source", like "psl_fpga.vhd_source", means this file will be pre-processed to generate the output file without "_source" suffix, like "psl_fpga.vhd". There are **`#ifdef`** macros or comments like **`-- only for NVME_USED=TRUE`**. They help to create a target VHDL/Verilog file with different configurations.

Below lists the files to change:

- snap_config and environmental files
- Hardware: psl_accel and psl_fpga (top) RTL files
- Hardware: tcl files for the workflow
- Hardware: xdc files for IO, floorplan, clock and bitstream settings
- Hardware: create DDR Memory controller IP (mig) in create_snap_ip.tcl, create DDR memory sim model, and other xdc files
- Hardware: create_ip, sim model and xdc files for other IPs
- Software: New card type, register definition

- Testing: jenkins
- Readme and Documents

Table 3.1. Config files to change

File name	Changes to do
scripts/Kconfig	adding card to the Kconfig menu. Provide Flash information (size/type/user address)
hardware/doc/SNAP-Registers.md	SNAP registers for new card - doc
hardware/setup/snap_config.sh	SNAP registers - setting

Table 3.2. RTL/xdc/tcl files to change

File name	Changes to do
hardware/hdl/core/psl_accel_${FPGACARD}.vhd_source	specific to card
hardware/hdl/core/psl_accel_types.vhd_source	specific to card
hardware/hdl/core/psl_fpga_${FPGACARD}.vhd_source	specific to card
hardware/setup/${FPGACARD}/capi_bsp_pblock.xdc	specific to card
hardware/setup/${FPGACARD}/snap_${FPGACARD}.xdc	specific to card
hardware/setup/${FPGACARD}/snap_ddr4pins.xdc	specific to card
hardware/setup/build_mcs.tcl	declare card name
hardware/setup/create_framework.tcl	declare card name
hardware/setup/create_snap_ip.tcl	declare card name and the IPs in use
hardware/setup/flash_mcs.tcl	declare card name
hardware/setup/snap_bitstream_post.tcl	declare card name
hardware/setup/snap_bitstream_pre.tcl	declare card name
hardware/setup/snap_bitstream_step.tcl	declare card name
hardware/setup/snap_impl_step.tcl	declare card name
hardware/sim/ddr4_dimm_???.sv	DDR memory model for simulation. Please get the information about how many DDR chips are connected together, the density and data width of each chip, and whether there is one chip is used for ECC (redundant). Take an existing one as a template and modify.
hardware/sim/top_capi?0.sv_source	Instantiate the DDR memory model
hardware/snap_check_psl (Only for CAPI1.0)	declare card name

Table 3.3. Software files to change

File name	Changes to do
software/lib/snap.c	declare card name
software/tools/snap_find_card	declare card name + SUBSYSTEM_ID
software/include/snap_regs.h	SNAP registers - setting

Table 3.4. Other files to change

File name	Changes to do
actions/scripts/snap_jenkins.sh	jenkins tests (optional)
defconfig/${FPGACARD}*.defconfig	For silent jenkins testing (optional)
README.md	Announce a new card is supported

3.4. Update capi-utils

capi-utils is the third git repository that needs a few modifications. Same as before, fork it, make the modifications and submit a pull request.

```
git clone https://github.com/[YOUR_USERNAME]/capi-utils
```

There is only one file to be modified: "psl-devices". Add a new line, for example

```
0x1014 0x0665 U200 Xilinx 0x1002000 64 SPIx4
```

It lists the Subsystem Vendor ID, Subsystem Device ID, Card name, FPGA chip, then it is the "User_image_address" on the flash. For SPI device, size of block is 64Bytes. "SPIx4" is the flash interface type. It may also be "DPIx16" or "SPIx8".

"SPIx8" uses two bitstreams so another starting address also needs to be provided. Script "capi-flash-script" needs two input bitstream files (primary and secondary) to program the flash.

3.5. Strategy to enable a new card

To enable a new card on SNAP, complete following tasks one by one.

3.5.1. Stage 1: Verify PCIe interface

1. Generate capi_bsp_wrap in capi2-bsp.
2. Make modifications to snap git repository as described above.
3. Select an action example without DDR, for example: hls_helloworld.
4. Go through the **`make model`** and **`make image`** processes and build the bitstream files.
5. Plug the card onto Power9 server and connect a JTAG/USB cable to a laptop. Install Vivado Lab on this laptop (it requires Windows or Linux operating system). Start Vivado Lab tool, open Hardware manager.
6. Power on the server. Soon the FPGA target is recognized by Vivado Lab tool.
7. Program the generated bitstream files (bin or mcs) to the card. On Vivado Lab tool, select the FPGA chip and right-click, choose "Add Configuration Memory Device..." and program the bin or mcs files to the flash. See in picture Figure 3.3, “Vivado Lab Edition” [15] and Figure 3.4, “Add Configuration Memory Device and Program the flash” [15]
8. Wait it done (It may take 10 minutes). Unplug the JTAG/USB cable, reboot the server.
9. After the server is booted, log into OS, run **`lspci`** to see if the card is there. (Usually with Device ID 0x0477). Then download snap, capi-utils, libcxl (from github). Go to snap directory, **`make apps`** and run the application.

Note

There is another way to replace step 6 to 8 which is called "Fast program bit-file when power on". Prepare the **bit** file on laptop in advance. Not like bin/mcs files which are for the flash, the bit file is used to program the FPGA chip directly. When the server is

powered on, after Vivado Lab sees the FPGA, right click the device, "program device..." and select the bit file **immediately**. This action only takes about 10 seconds and can be done before skiboot on the server starts to scan PCIe devices.

Be aware of the fact that now only FPGA chip is programmed, (the flash memory is still empty or holding old data), so when the server is powered off or reboot the recent programming to FPGA chip will be lost.

Figure 3.3. Vivado Lab Edition

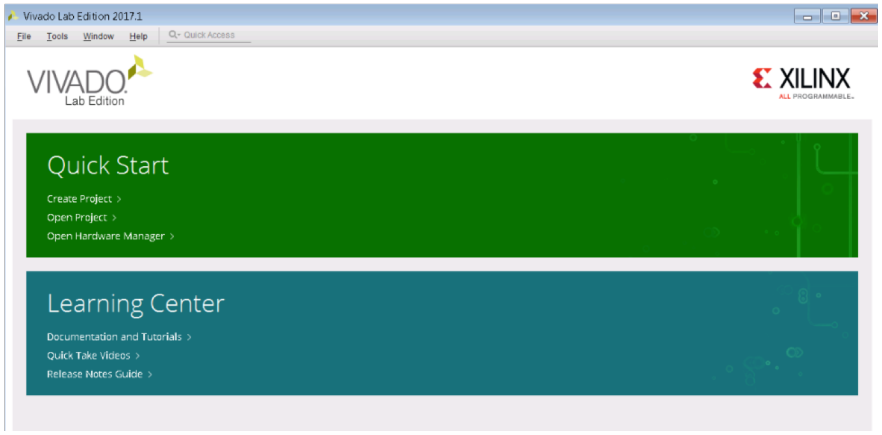
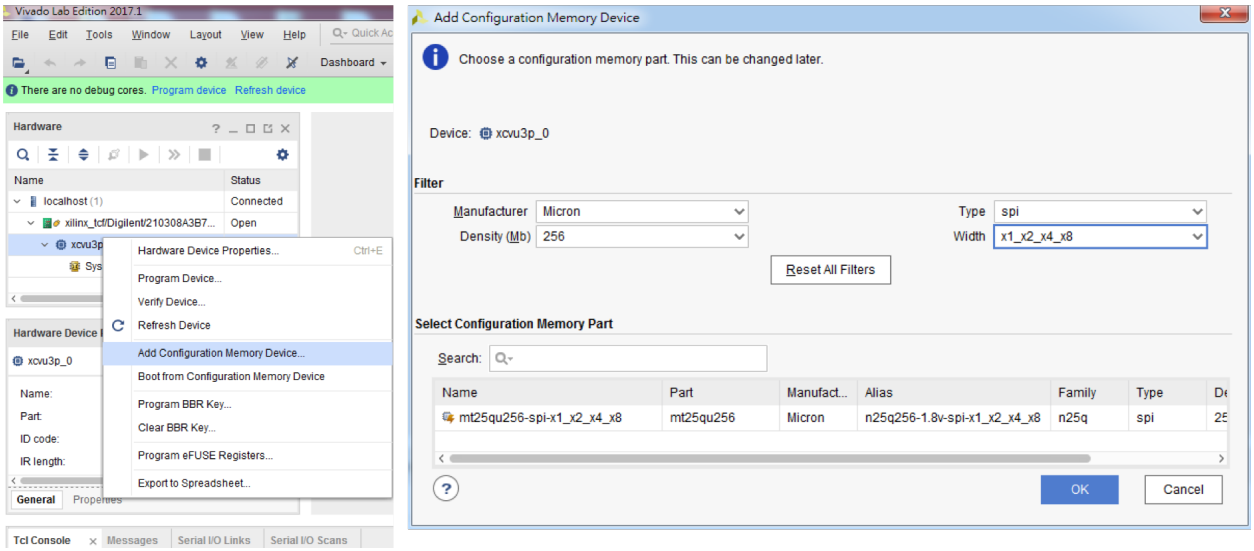


Figure 3.4. Add Configuration Memory Device and Program the flash



Important

When installing **Vivado Lab**, please choose as same version as the Vivado tool which was used to build images.

Tips to debugging:

1. Seeing 0477 by **`lspci`** is the most important milestone. If not, please check file "**/sys/firmware/opal/msglog**" to see whether there are link training failed messages. A successful message

looks like this, which means this PCIe device has been scanned and recognized. The number followed "PHB#" is the PCIe device identifier in the format of "domain:bus:slot.func":

```
[   63.403485191,5] PHB#0000:00:00.0 [ROOT] 1014 04c1 R:00 C:060400 B:01..01 SLOT=
CPU1 Slot2 (16x)
[   63.403572553,5] PHB#0000:01:00.0 [EP  ] 1014 0477 R:02 C:1200ff (
  device) LOC_CODE=CPU1 Slot2 (16x)
```

2. Check **`dmesg`**. Run "dmesg > dmesg.log" and search "cxl" in "dmesg.log" file. A normal output should be look like this:

```
[    9.301403] cxl-pci 0000:01:00.0: Device uses a PSL9
[    9.301523] cxl-pci 0000:01:00.0: enabling device (0140 -> 0142)
[    9.303327] cxl-pci 0000:01:00.0: PCI host bridge to bus 0006:00
[    9.306749] cxl afu0.0: Activating AFU directed mode
```

3. Today most of the linux kernel versions already include cxl module. Doublecheck it by:

```
modinfo cxl
```

4. If the PCIe device has been recognized as CAPI, `ls /dev/cxl` and "afu*" devices should be there. Then application software can open the device like operating an ordinary file.

```
ls /dev/cxl
afu0.0m  afu0.0s
```

Some other useful commands to check PCIe config (with the right PCIe identifier "domain:bus:slot.func")

```
sudo lspci -s 0000:01:00.0 -vvv
```

It shows the settings coded in Xilinx PCIe core, like Subsystem Device ID:

```
0000:01:00.0 Processing accelerators: IBM Device 0477 (rev 02) (prog-if ff)
 Subsystem: IBM Device 0660
```

Link Speed

```
LnkSta: Speed 8GT/s, Width x16, TrErr- Train- SlotClk+ DLActive- BWMgmt- ABWMgmt-
```

Vital Product Data which was coded in capi_vsec.vhdl

```
Capabilities: [b0] Vital Product Data
Product Name: U200 PCIe CAPI2 Adapter
Read-only fields:
 [PN] Part number: Xilinx.U200
 [V1] Vendor specific: 0000000000000000
 [V2] Vendor specific: 0000000000000000
 [V3] Vendor specific: 0000000000000000
 [V4] Vendor specific: 0000000000000000
 [RV] Reserved: checksum good, 3 byte(s) reserved
End
```

And see VSEC and kernel module:

```
Capabilities: [400 v1] Vendor Specific Information: ID=1280 Rev=0 Len=080 <?>
Kernel driver in use: cxl-pci
Kernel modules: cxl
```

5. If nothing shows by **`ls /dev/cxl`**, check PCIe config space:

```
sudo hexdump /sys/bus/pci/devices/0000\:00\:00.1/config
```

Please pick up the correct PCIe device identifier (0000:00:00.1). Make sure the VSEC is properly linked. If not, go back to check "capi_vsec.vhdl".

```
0000000 1014 0477 0146 0010 ff02 1200 0000 0000
0000010 000c 0000 2200 0006 000c 1000 2200 0006
0000020 000c 0000 0000 0002 0000 0000 1014 0668
0000030 0000 0000 0040 0000 0000 0000 00ff 0000
0000040 4801 0003 0008 0000 7005 0080 0000 0000
0000050 0000 0000 0000 0000 0000 0000 0000 0000
0000060 7011 0000 0000 0000 0000 0000 0000 0000
0000070 b010 0002 8022 0000 2950 0000 f103 0043
0000080 0000 1103 0000 0000 0000 0000 0000 0000
0000090 0000 0000 0016 0000 0010 0000 000e 0000
00000a0 0003 001e 0000 0000 0000 0000 0000 0000
00000b0 0003 0000 2082 5300 0000 0000 0000 0000
00000c0 0000 0000 0000 0000 0000 0000 0000 0000
*
0000100 0001 1c01 0000 0000 0000 0044 2030 0046 --> next_ptr: 1c0
0000110 0000 0000 e000 0000 0000 0000 0000 0000
0000120 0000 0000 0000 0000 0000 0000 0000 0000
*
00001c0 0019 1f01 0000 0000 0000 0000 0000 0000 --> next_ptr: 1f0
00001d0 0000 0000 0000 0000 0000 0000 0000 0000
*
00001f0 0002 e801 0000 0000 0000 3100 0000 0000 --> e80 (or 400) points to VSEC
0000200 0000 0000 00ff 8000 0000 0000 0000 0000
0000210 0000 0000 0000 0000 0000 0000 0000 0000
*
0000e80 000b 0001 1280 0800 0801 0021 0006 0200 --> VSEC starts from e80 (or 400)
0000e90 0000 b000 0000 0000 0000 0000 0000 0000
0000ea0 0100 0000 0040 0000 0200 0000 0400 0000
0000eb0 0000 0000 0000 0000 0000 0000 0000 0000
*
0000ed0 0000 0000 0000 0000 0000 8000 0000 0000
0000ee0 0000 0000 0000 0000 0000 0000 0000 0000
*
0001000
```

3.5.2. Stage 2: Verify Flash interface

Use capi-utils to program the bitstream files. If it succeeds, it proves that the Flash interface has been configured correctly. After this step, JTAG connector is not needed anymore. Use "capi-flash-script" to program the FPGA bitstreams.

The mechanic behind "capi-flash-script" is:

There is a flash controller on FPGA (in capi_bsp_wrap), and it connects to PCIe config space. The flash controller exposes four VSEC registers to allow host system to control. They are:

- Flash Address Register
- Flash Size Register
- Flash Status/Control Register

- Flash Data Port

The details are decribed in Coherent Accelerator Interface Architecture, Chapter 12.3, "CAIA Vendor-Specific Extended Capability Structure". So the C file in capi-utils reads FPGA bitstream "bin" file, and writes the data to VSEC "Flash Data Port" register. So the bytes are sent through PCIe, to Flash controller and finally arrive to flash memory on the card.

3.5.3. Stage 3: Verify DDR interface

1. Select another action example (hdl_example with DDR) or hls_memcopy.
2. **`make model`** and **`make sim`**. Make sure the DDR simulation model works well.
3. **`make image`** to generate the bitstream files.
4. Use capi-utils to program the bitstream "bin" file to the card.
5. Run the application to see whether it works.

Basically SNAP only implemented one DDR Bank (or channel) while most cards have two to four banks. (N250S+ is one of the rare card which has only one DDR bank). To implement more DDR channels, depending on user's needs, there are two options: the first is to just extend the size of the first bank by adding this second bank on the same DDR memory controller. The other option is to use two (or more) memory controllers in parallel to have a higher throughput. This later option means duplicating the DDR memory controller in place and this will take twice the place in the design. In this case, the action_wrapper also requires change to add the additional DDR ports. For HLS design, another HLS DDR port should be added into "actions/[YOUR_ACTION]/hw/XXX.CPP". As for an opensource project, everyone is welcomed to add your contribution by implementing it and add it to the SNAP design.

3.5.4. Stage 4: Verify Other IO interface

This step is decided by the card's capabilities and the specific IOs that the card can provide. Like the second or more DDR channels, user can add their code for other IO interface freely.

3.5.5. Stage 5: Performance Validation

Check the result of "snap/actions/hls_memcopy/tests/test_*_throughput.sh" for bandwidth and "snap/actions/hls_latency_eval/test/test*.sh" for latency.

3.5.6. Stage 6: Pressure Test

Prepare bitstream files for basic tests, throughput tests, latency tests, max-power tests. Adding image flashing tests, card reset tests and others. Run them intensively.

3.6. Cleanup and submit

Now a new FPGA card has been enabled to CAPI2.0 SNAP. Cleanup your workspace, check files and submit your work!

- Submit the pull request of your "capi2-bsp fork" before "snap fork". Assign the reviewer and wait capi2-bsp to be merged into https://github.com/open-power/capi2-bsp master branch

- Update the submodule pointer to the latest "open-power/capi2-bsp" master and then submit the pull request of your forked snap.
- Capi-utils is independent. Just create a pull request and assign a reviewer. It can only been merged into master branch after having been reviewed.

Appendix A. OpenPOWER Foundation overview

The OpenPOWER Foundation was founded in 2013 as an open technical membership organization that will enable data centers to rethink their approach to technology. Member companies are enabled to customize POWER CPU processors and system platforms for optimization and innovation for their business needs. These innovations include custom systems for large or warehouse scale data centers, workload acceleration through GPU, FPGA or advanced I/O, platform optimization for SW appliances, or advanced hardware technology exploitation. OpenPOWER members are actively pursing all of these innovations and more and welcome all parties to join in moving the state of the art of OpenPOWER systems design forward.

To learn more about the OpenPOWER Foundation, visit the organization website at openpowerfoundation.org.

A.1. Foundation documentation

Key foundation documents include:

- *Bylaws of OpenPOWER Foundation*
- *OpenPOWER Foundation Intellectual Property Rights (IPR) Policy*
- *OpenPOWER Foundation Membership Agreement*
- *OpenPOWER Anti-Trust Guidelines*

More information about the foundation governance can be found at openpowerfoundation.org/about-us/governance.

A.2. Technical resources

Development resouces fall into the following general categories:

- Foundation work groups
- Remote development environments (VMs)
- Development systems
- Technical specifications
- Software
- Developer tools

The complete list of technical resources are maintained on the foundation Technical Resources web page.

A.3. Contact the foundation

To learn more about the OpenPOWER Foundation, please use the following contact points:

- General information -- <info@openpowerfoundation.org>
- Membership -- <membership@openpowerfoundation.org>
- Technical Work Groups and projects -- <tsc-chair@openpowerfoundation.org>
- Events and other activities -- <admin@openpowerfoundation.org>
- Press/Analysts -- <press@openpowerfoundation.org>

More contact information can be found at openpowerfoundation.org/get-involved/contact-us.