Enable a FPGA card for SNAP2.0

# Introduction

## HDK and SDK

CAPI stands for Coherent Accelerator Processor Interface which enables FPGA to access Host memory by virtual address.

For CAPI1.0, on the FPGA side, 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 IBM, in the form of a post-implemented dcp file (Xilinx Vivado design checkpoint). A Vivado project, including PSL module and sample user logic (AFU), is delivered to acceleration developers. This Vivado project is called **HDK** (Hardware Development Kit).



Figure 1 Enable CAPI interface on FPGA

On the host (Power8 server), acceleration developers need to download “libcxl” which provides the basic user APIs to operate CAPI cards. Software programs will call the user APIs from libcxl.

When developing the acceleration function, there is a way to simulate the software programs and hardware logic together in a logic simulator (Cadence ncsim, Vivado xsim, Mentor questasim or Synopsys vcs) which gives the confidence to run an acceleration function on the FPGA board with “first-time-pass”. This is called PSLSE (PSL Simulation Engine).

The sample software program, together with libcxl (open-source code on github) and PSLSE (open-source code on github) make a package named **SDK** (Software Development Kit).

* *When we are saying “Enable a customer FPGA card to a CAPI card”, it firstly points to generate a HDK. SDK is general and doesn’t need special efforts for a new FPGA card.*

## SNAP framework

SNAP is the developing framework for CAPI, and is the abbreviation of Storage, Networking and Analytics Programming. It is an open-source framework (<https://github.com/open-power/snap>)

On the FPGA side, SNAP framework adds a PSL/AXI bridge, a DDR SDRAM controller and an optional NVMe controller. Thus, the developer can focus on their acceleration kernel logic (here we call it hardware action) and interface the framework via several AXI ports.

On the host side, SNAP framework adds SNAP library. It makes the software coding even easier.



Figure 2 SNAP Framework

For a new FPGA card, the majority work to enable SNAP framework is still on the hardware side. After we generate the PSL module, we also need to add the PSL/AXI bridge (Internally we call it snap\_core) and the controllers on the right side into the Vivado project.

There are also some small modifications to SNAP library to announce the name of newly supported card and recognize this card.

# Enable HDK

## PSL module



Figure 3 PSL diagram

Generally, PSL contains surrounding logic and the kernel logic:

* PCIe hard IP core (pcie3\_ultrascale\_0)
* Flash Controller (psl\_flash)
* VSEC: Vender Specific Extended Capability (psl\_vsec)
* Xilinx MultiBoot control logic (psl\_xilmltbt)
* PSL kernel logic (psl)

The interface between PSL and AFU has 5 groups of signals, described in PSL spec <http://openpowerfoundation.org/wp-content/uploads/resources/psl-afu-spec/content/go01.html>

The interface between PSL and Chip IOs are card specific, and the information need to be provided by Card Vendor. Generally, they include:

* Flash interface
* PCIe interface:
  + perst
  + refclk
  + data: 8 lanes of TX and RX
* Miscellaneous: I2C, LED and so on.

## Generate PSL checkpoint

### Steps and directory structure

We use an “Out-of-context” flow to generate a PSL dcp file. For a new FPGA card, following steps need to be done:

Figure 4 Steps to build a PSL checkpoint

We use an “Out-of-context” flow to generate a PSL dcp file. The directory structure is as following:

Figure 5 HDK directory structure

In **build\_dir/Sources**,

* The “prj” directory includes the source file lists.
* The “top” directory includes the top design file “psl\_fpga.vhdl” and the wrapper for AFU “psl\_accel.vhdl”
* PSL source files are in “psl” directory.
* AFU source files are in “afu” directory.
* “cores” includes 4 Xilinx IP cores used by PSL.
* “xdc” are the constraint files used by PSL and the top design.

In build\_dir, psl\_fpga.tcl is the script “**entrance**”. It assigns the FPGA chip information, and the build flow.

FPGA chip information is needed for a new FPGA card, for example:

set device "xcku115"

set package "-flva1517"

set speed "-2-e"

And some controlling bits are for two build flows:

1. Build a PSL checkpoint.
2. Build a whole FPGA image (including AFU).

In this section, we just talk about the first build flow – “build a PSL checkpoint”, and the controlling bits should be set as:

####flow control

set run.topSynth 1

set run.oocSynth 1

set run.tdImpl 0

set run.oocImpl 1

set run.topImpl 0

set run.flatImpl 0

The outfile file will be placed in “Checkpoint” directory, the file name is “b\_route\_design.dcp”.

### Upgrade Xilinx IP cores

When a FPGA chip type is changed, or the Vivado tool version has been upgraded, you need to upgrade the Xilinx IP cores that are used in PSL module. PSL module has instantiated four Xilinx IP cores (in Sources/cores):

* pcie3\_ultrascale\_0
* sem\_ultra\_0 (Soft Error Migration)
* clk\_wiz\_0
* tx\_wr\_fifo

Steps to upgrade them:

1. Open Vivado GUI
2. Create a new project. For the second time, just open the project with the four IP cores.
3. Import IP cores (by importing \*.xci files under “Source/cores/xxx” directory). For the second time, this step is not needed.
4. Set FPGA type in Project Settings.
5. Run “Tools->Report->Report IP Status”
6. “Upgrade All” and read the upgrade log.

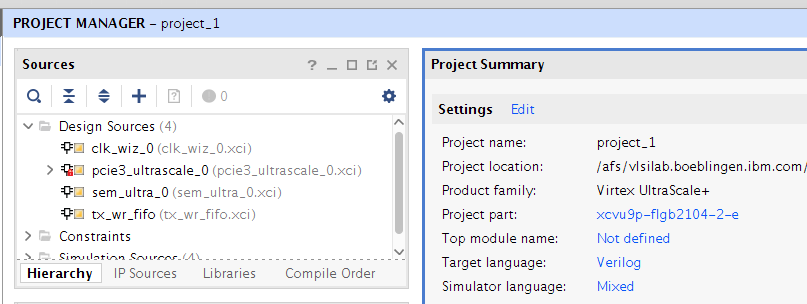


Figure 6 Small project to upgrade IP cores

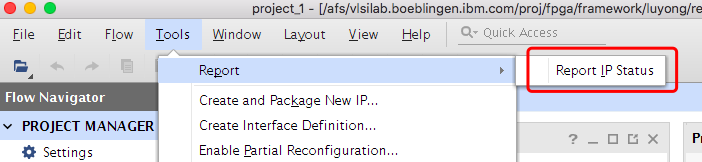


Figure 7 Report IP Status

* For PCIe IP, you need to change subsystem\_id for a new card. Right click pcie3\_ultrascale\_0 -> Reconfig IP and change the subsystem ID field.
* Ask an IBM representative for the subsystem ID allocation.

### Input xdc files

The IO pin package information for the new card should be provided by card vendor. Generally, they include Flash Interface, PCIe Interface and other interfaces like I2C and LED. Sample code with IO pins in b\_phys.xdc:

Refer to Xilinx document UG575 for detailed pin package information.

set\_property PACKAGE\_PIN AJ15 [get\_ports {o\_flash\_a[1]}]

set\_property PACKAGE\_PIN AK15 [get\_ports {o\_flash\_a[2]}]

set\_property PACKAGE\_PIN AH14 [get\_ports {o\_flash\_a[3]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {o\_flash\_a[1]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {o\_flash\_a[2]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {o\_flash\_a[3]}]

Some other constraints also must be updated for the new selection of FPGA chip. It defines the floorplan for PSL.

* #There is also a hacking to keep VSEC address for Vivado2017.4:

set\_property PF0\_SECONDARY\_PCIE\_CAP\_NEXTPTR 12'h400 [get\_cells \*pcihip0/psl\_pcihip0\_inst/inst/pcie3\_uscale\_top\_inst/pcie3\_uscale\_wrapper\_inst/PCIE\_3\_1\_inst]

### Run Vivado

vivado -mode batch -source psl\_fpga.tcl –notrace

The checkpoint file b\_route\_design.dcp will be generated and put in “Checkpoint” directory. With this checkpoint file, we can continue to build a full FPGA bit image and validate it on hardware.

## Generate full FPGA image

### Steps

Figure 8 Steps to build the full FPGA image

This time the controlling bits should be set to:

####flow control

set run.topSynth 1

set run.oocSynth 0

set run.tdImpl 0

set run.oocImpl 0

set run.topImpl 1

set run.flatImpl 0

### Check top design file psl\_fpga.vhdl

For a new card, the IO pins and functions may be different to your reference card design. So the logic in top file psl\_fpga.vhdl needs to be updated.

Similarly, the xdc file “……topimp.xdc” also needs to be updated.

### Prepare filelist for psl\_fpga.prj

The “prj” file is a file list. It should reflect the AFU design changes. Section

### Run Vivado

Two sub-steps here:

vivado -mode batch -source psl\_fpga.tcl –notrace

Now the bit file is generated.

vivado –mode batch –source write\_bitstream.tcl –notrace

Now you get the bin file to be program to the flash attached to FPGA.

For more information about FPGA configuration, please refer to Xilinx Document UG570.

Then you can program the generated bin file to FPGA either by JTAG or on-line programming tools capi-utils (<https://github.com/ibm-capi/capi-utils>)

# Enable SNAP

From Figure 2 SNAP Framework, for a new FPGA card, the detailed items to update can be classified into following sections:

* Preparations
* Hardware RTL, setup, simulation
* Software and tools
* Testing
* Publishing

## Preparations

First, give a FPGACARD name. It should start from the company’s name, following with the card ID and be short. For example. ADKU3 = Alpha-Data ADM-PCIE-KU3.

Get follow information from the card vendor. (You can use the “Status” column to trace the progress.)

Table 1 Information by Card Vendor

|  |  |  |
| --- | --- | --- |
| Status | Item | Description |
|  | FPGACARD | Short card name used in SNAP |
|  | FPGACHIP | FPGA part name, for example, xcvu9p-fsgd2104-2L-e |
|  | Flash Type | The information of flash chip that attached to FPGA.  For example, mt28gu01gaax1e-bpi-x16 |
|  | DDR MC IP | DDR memory controller Vivado IP  DDR memory type, speed, capacity … |
|  | Other peripherals | NVMe IP, Ethernet IP and so on (Optional) |
|  | IO pins | PACKAGE\_PIN for base\_image or bsp: flash, pcie, i2c etc.  PACKAGE\_PIN for peripheral IPs |

Then refer to “Chapter 2: Enable HDK” to generate PSL checkpoint and test a sample AFU on the new card.

## SNAP environment updates

The best way is to grep some keywords like “S121B” or “AD8K5” under the directories and look for the locations that need modifications. Check the coding blocks around “CONFIG\_” that may differ on each card.

Table 4 Sim files to change

|  |  |
| --- | --- |
| **File name** | **Changes done** |
| snap\_env | configurating SNAP config |
| scripts/Kconfig | adding card to the Kconfig menu |
| hardware/Makefile | configurating SNAP config |
| hardware/doc/SNAP-Registers.md | SNAP registers - doc |
|  |  |
| hardware/setup/snap\_config.sh | SNAP registers - setting |

## SNAP hardware updates

Table 2 RTL changes

|  |  |
| --- | --- |
| **File name** | **Changes done** |
| hardware/hdl/core/psl\_accel\_rcxvup.vhd\_source | [NEW] specific to card |
| hardware/hdl/core/psl\_accel\_types.vhd\_source | specific to card |
| hardware/hdl/core/psl\_fpga\_rcxvup.vhd\_source | [NEW] specific to card |
| hardware/setup/RCXVUP/capi\_bsp\_pblock.xdc | [NEW] specific to card |
| hardware/setup/RCXVUP/snap\_RCXVUP.xdc | [NEW] specific to card |
| hardware/setup/RCXVUP/snap\_ddr4pins.xdc | [NEW] specific to card |

Table 3 Setup changes

|  |  |
| --- | --- |
| **File name** | **Changes done** |
| hardware/setup/build\_mcs.tcl | declare card name |
| hardware/setup/create\_framework.tcl | declare card name |
| hardware/setup/create\_ip.tcl | declare card name |
| 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/snap\_check\_psl | declare card name |

## SNAP software updates

Table 5 Software and tools to update

|  |  |
| --- | --- |
| **File name** | **Changes done** |
| software/lib/snap.c | declare card name |
| software/tools/snap\_find\_card | declare card name + id |
| software/include/snap\_regs.h | SNAP registers - setting |

## Testing

Table 6 Checklist for testing

|  |  |
| --- | --- |
| **File name** | **Changes done** |
| actions/scripts/snap\_jenkins.sh | jenkins tests |

## Publishing

Table 7 Checklist for publishing

|  |  |
| --- | --- |
| **File name** | **Changes done** |
| README.md | Add |

|  |  |  |
| --- | --- | --- |
| Status | Target | Comment |
|  | File header (license note) | Get the permission from open card vendor to let some files (CVS, XDC) be open-source |
|  | README.md | Announce SNAP supports a new card |
|  | PSL Checkpoint | Update to OpenPower Portal webpage |

Now we can say a new card has been enabled for SNAP1.0.

NEW DRAFT FROM BRUNO 2019/01/10

**Flash interface**

3 types of interfaces have been implemented up to now: SPIx4, SPIx8 and BPIx16. Most of CAPI1.0 cards are using BPIx16. Most of CAPI2.0 cards are using SPI interface.

Implementation are slightly different between CAPI1.0 and CAPI2.0 since BSP was created for CAPI2.0.

To enable a new card, it is recommended to take the closest configuration and copy it.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **SNAP Code Name** | **Flash Interface** | **Flash Size  (in MB)** | **Flash device** | **RS pins** | **Flash Factory @** | **Flash User @ (for BPIx16: 2B@ for SPI, 1B @)** |
| *Variable🡺* | ***FPGACARD*** | ***FLASH\_INTERFACE*** | ***FLASH\_SIZE*** | ***flashdevice*** | ***rs\_pins*** | ***FLASH\_FACTORYADDR*** | ***FLASH\_USERADDR*** |
| P8 / CAPI1.0 | **ADKU3** | **BPIx16** | 128 | mt28gu01gaax1e-bpi-x16 | ***{25:24}*** | 0x0000\_0000 | 0x0100\_0000 |
| P8 / CAPI1.0 | **AD8K5** | **BPIx16** | 128 | mt28gu01gaax1e-bpi-x16 | ***{25:24}*** | 0x0000\_0000 | 0x0200\_0000 |
| P8 / CAPI1.0 | **N250S** | **BPIx16** | 64 | mt28gu512aax1e-bpi-x16 | ***{25:24}*** | 0x0000\_0000 | 0x0100\_0000 |
| P8 / CAPI1.0 | **S121B** | **BPIx16 (v16.1)** | 128 | mt28gu01gaax1e-bpi-x16 | {26:25) (removed) | 0x0000\_0000 | 0x0200\_0000 |
| P8 / CAPI1.0 | **S121B** | **SPIx4 (v16.2)** | 256 | Added INFO but still in comment  => mt25qu02g-spi-x1x2x4 | - | 0x0000\_0000 | 0x0800\_0000 |
| P9 / CAPI2.0 | **N250SP** | **BPIx16** | 128 | mt28ew01ga-bpi-x16 | **{26:25}** | 0x0000\_0000 | 0x0200\_0000 |
| P9 / CAPI2.0 | **RCXVUP** | **SPIx8** | 256 | mt25qu01gbbb8e12-0sit | - | 0x0000\_0000 | 0x0800\_0000 |
| P9 / CAPI2.0 | **FX609** | **SPIx4** | 128 | mt25qu01gbbb8e12-0sit | - | 0x0000\_0000 | 0x0400\_0000 |
| P9 / CAPI2.0 | **S241** | **SPIx4** | 256 | MT25QU02GCBB8E12-0SIT (modified) | - | 0x0000\_0000 | 0x0800\_0000 |
| P9 / CAPI2.0 | **AD9V3** | **SPIx8** | 64 | mt25qu256aba8e12-1sit | - | 0x0000\_0000 | 0x0200\_0000 |

**SNAP parameters definition:**

The Flash interface parameters are defined and used in the following files. Please modify

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variable** | **Value set in:** | **Value used in:** | **Value used for:** | **Local variable name** |
| **FLASH\_INTERFACE** | scripts/  Kconfig | *Specific case since 2 configs: hardware/setup/S121B/*  *snap\_bitstream\_pre.tcl* | wr\_cfgmem | flash\_interface |
| hardware/setup/  build\_mcs.tcl |
| hardware/setup/  snap\_bitstream\_post.tcl |
| hardware/setup/  snap\_bitstream\_step.tcl |
|  |  |  |  |  |
| **Variable** | **Value set in:** | **Value used in:** | **Value used for:** | **Local variable name** |
| **FLASH\_SIZE** | scripts/  Kconfig | hardware/setup/  build\_mcs.tcl | wr\_cfgmem | flash\_size |
| hardware/setup/  snap\_bitstream\_post.tcl |
| hardware/setup/  snap\_bitstream\_step.tcl |
|  |  |  |  |  |
| **Variable** | **Value set in:** | **Value used in:** | **Value used for:** | **Local variable name** |
| **flashdevice** | hardware/setup/  flash\_mcs.tcl | hardware/setup/  flash\_mcs.tcl | create\_hw\_cfg | - |
|  |  |  |  |  |
| **Variable** | **Value set in:** | **Value used in:** | **Value used for:** | **Local variable name** |
| **rs\_pins** | hardware/setup/  flash\_mcs.tcl | hardware/setup/  flash\_mcs.tcl | set\_property PROGRAM.BPI\_RS\_PINS | - |
|  |  |  |  |  |
| **Variable** | **Value set in:** | **Value used in:** | **Value used for:** | **Local variable name** |
| **FLASH\_FACTORYADDR** | scripts/  Kconfig | hardware/setup/  build\_mcs.tcl | wr\_cfgmem | factory\_addr |
|  |  |  |  |  |
| **Variable** | **Value set in:** | **Value used in:** | **Value used for:** | **Local variable name** |
| **FLASH\_USERADDR** | scripts/  Kconfig | hardware/setup/  build\_mcs.tcl | wr\_cfgmem | user\_addr |

**SNAP logic implementation:**

For naming convention, let’s take the ADKU3 card (CAPI1.0) as the BPIx16 implementation example and the AD9V3 card (CAPI2.0) as the SPI implementation example.

BPI

Interface is:

o\_flash\_oen: out std\_logic;

o\_flash\_wen: out std\_logic;

o\_flash\_rstn: out std\_logic;

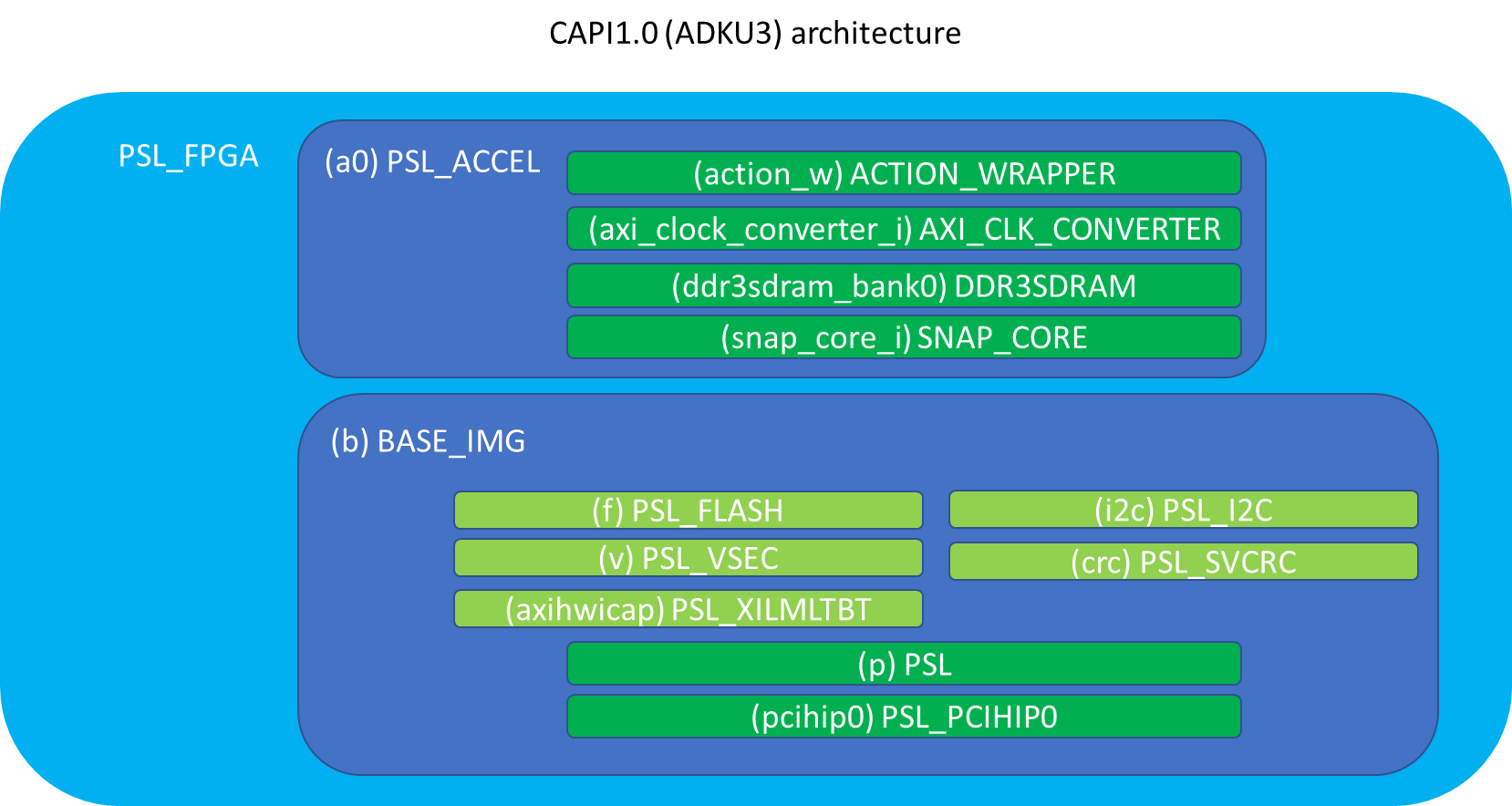
o\_flash\_a: out std\_logic\_vector(1 to 26);

o\_flash\_advn: out std\_logic;

b\_flash\_dq: inout std\_logic\_vector(0 to 11);

CAPI1.0

VHDL hierarchy



(action\_w) ACTION\_WRAPPER

(axi\_clock\_converter\_i) AXI\_CLK\_CONVERTER

(ddr3sdram\_bank0) DDR3SDRAM

(snap\_core\_i) SNAP\_CORE

(b) BASE\_IMG

(p) PSL

(pcihip0) PSL\_PCIHIP0

CAPI1.0 (ADKU3) architecture

(f) PSL\_FLASH

(v) PSL\_VSEC

(axihwicap) PSL\_XILMLTBT

(i2c) PSL\_I2C

(crc) PSL\_SVCRC

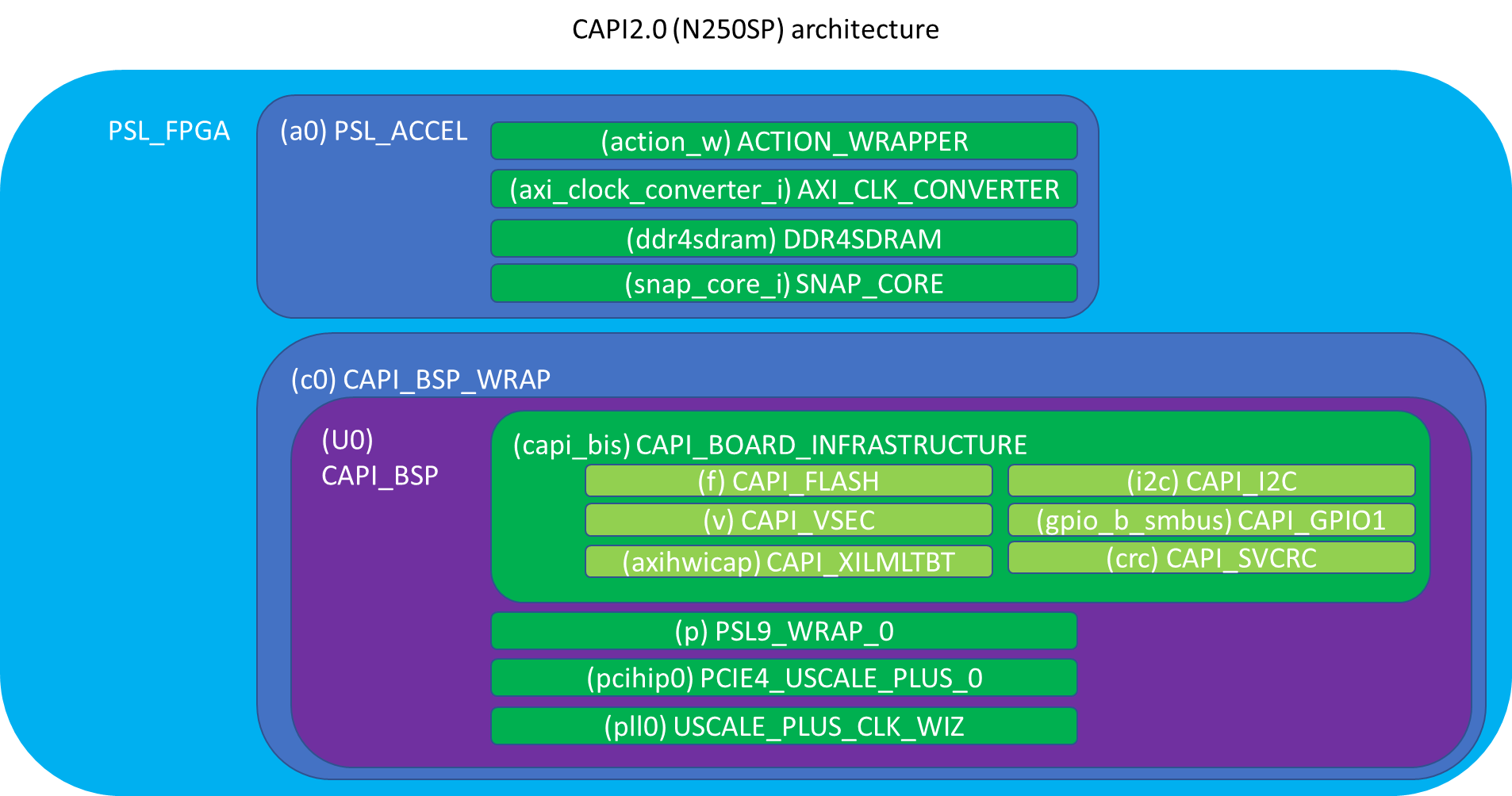
|  |  |  |
| --- | --- | --- |
| **Top function** | **Function called** | **file** |
| psl\_fpga | base\_img | hardware/hdl/core/psl\_fpga\_adku3.vhd\_source |
| base\_img | psl\_flash | ADKU3/b\_route\_design.dcp (base\_img\_stub.vhdl) |

Constraints

|  |  |  |
| --- | --- | --- |
| **Top function** | **Property type example** | **file** |
| Pins assignments |  | ADKU3/b\_route\_design.dcp (psl\_fpga\_synth.xdc) |
| Interface configuration | set\_property CONFIG\_MODE BPI16 | hardware/setup/ADKU3/snap\_bitstream\_pre.tcl |

CAPI2.0

VHDL hierarchy



|  |  |  |
| --- | --- | --- |
| **Top function** | **Function called** | **file** |
| psl\_fpga | capi\_bsp\_wrap | hardware/hdl/core/psl\_fpga\_**n250sp**.vhd\_source |
| capi\_bsp\_wrap | capi\_bsp | hardware/capi2-bsp/common/tcl/create\_capi\_bsp.tcl |
| capi\_bsp | capi\_board\_infrastructure | hardware/capi2-bsp/**N250SP**/src/capi\_bsp.vhdl |
| capi\_board\_infrastructure | capi\_flash | hardware/capi2-bsp/**N250SP**/src/capi\_board\_infrastructure.vhdl |
| capi\_flash | *Logic design* | hardware/capi2-bsp/**N250SP**/build/capi\_bsp\_gen/sim/capi\_flash.vhdl |

Constraints

|  |  |  |
| --- | --- | --- |
| **Top function** | **Property type example** | **file** |
| Pins assignments | set\_property PACKAGE\_PIN M20 | hardware/capi2-bsp/**N250SP**/xdc/capi\_bsp\_io.xdc |
| Interface configuration | set\_property CONFIG\_MODE BPI16 | hardware/capi2-bsp/**N250SP**/xdc/ capi\_bsp\_config.xdc |

SPI

VHDL hierarchy (with the BSP/CAPI2.0)

Interface is

spi\_clk: out std\_logic;

spi\_miso\_secondary : in std\_logic;

spi\_mosi\_secondary : out std\_logic;

spi\_cen\_secondary : out std\_logic

|  |  |  |
| --- | --- | --- |
| **Top function** | **Function called** | **file** |
| psl\_fpga | capi\_bsp\_wrap | hardware/hdl/core/psl\_fpga\_**ad9v3**.vhd\_source |
| capi\_bsp\_wrap | capi\_bsp | hardware/capi2-bsp/common/tcl/create\_capi\_bsp.tcl |
| capi\_bsp | capi\_board\_infrastructure | hardware/capi2-bsp/**AD9V3**/src/capi\_bsp.vhdl |
| capi\_board\_infrastructure | capi\_flash\_spi\_mt25qt | hardware/capi2-bsp/**AD9V3**/src/capi\_board\_infrastructure.vhdl |
| capi\_flash\_spi\_mt25qt | *Logic design* | hardware/capi2-bsp/**AD9V3**/src/capi\_flash\_spi\_mt25qt.vhdl |

Constraints

|  |  |  |
| --- | --- | --- |
| **Top function** | **Property type example** | **file** |
| Pins assignments | set\_property PACKAGE\_PIN AG30… | hardware/capi2-bsp/AD9V3/xdc/capi\_bsp\_io.xdc |
| Interface configuration | set\_property CONFIG\_MODE SPIx4/8 | ***CAPI1.0 path ok (S121B)***  hardware/setup***/<card\_name>/***snap\_bitstream\_pre.tcl  ***CAPI2.0 path ok (AD9V3, FX609, S241, RCXVUP)***  hardware/capi2-bsp***/<card\_name>/***xdc/capi\_bsp\_config.xdc  ***~~CAPI2.0 “Old” path (RCXVUP, AD9V3, FX609, S241)~~***  ~~hardware/setup~~***~~/<card\_name>/~~***~~snap\_bitstream\_pre.tcl~~  Duplicated path for AD9V3, FX609, S241!!! 🡺 old **removed**  Move path for RCXVUP to BSP 🡺 old **removed** |