IBM CAPI SNAP framework

Version 1.0

Quick Start Guide on a General environment.

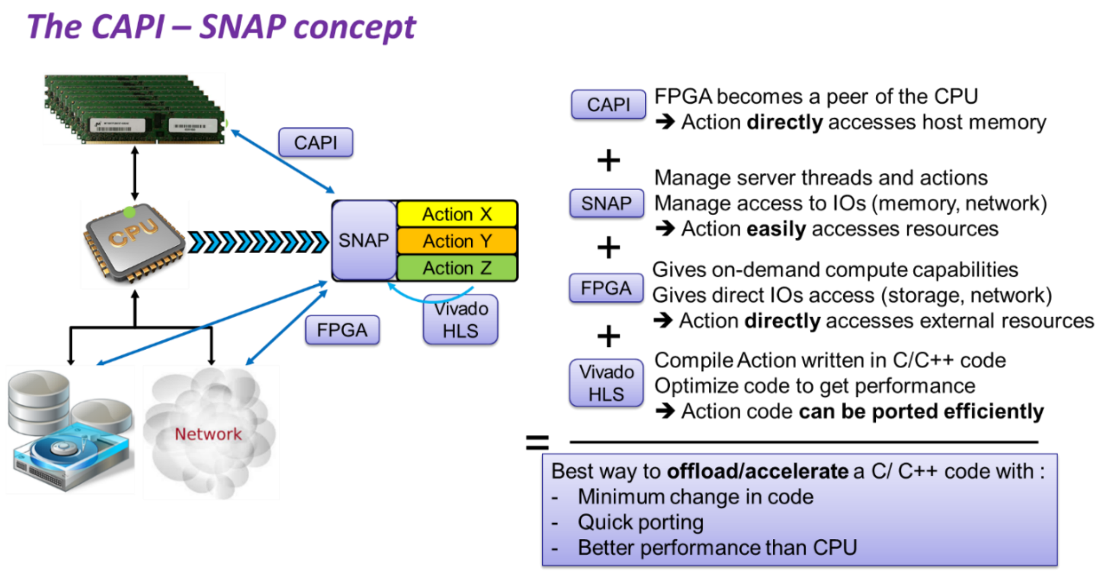
The Quick Start Guide describes how to log to SuperVessel and use SNAP environment.

Product Overview

The CAPI SNAP Framework is an open source enablement environment developed by members of the OpenPOWER Accelerator Work Group. SNAP, which is an acronym for Storage, Network, Analytics and Programming, indicates the dual thrusts of the framework:

- Enabling application programmers to embrace FPGA acceleration and all of CAPI's technology benefits.
- Placing the accelerated compute engines, or FPGA "actions", closer to the data to provide higher performance

SNAP hides the complexity of porting an function/action to an external card. By integrating together the following 4 components, you will get the best you can to port and offload or even accelerate your code.


The CAPI – SNAP concept
CAPI
CPU
SNAP
Action X
Action Y
Action Z
FPGA
Vivado HLS
Network
CAPI
FPGA becomes a peer of the CPU
➔ Action directly accesses host memory
+
SNAP
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
=
Best way to offload/accelerate a C/ C++ code with :
- Minimum change in code
- Quick porting
- Better performance than CPU

This document will succesfully go through the following items:
- General documentation to understand SNAP and the hls_helloworld example
- Process to install tools and set your environement.
- Configure SNAP for the card and the action
- Go through the **3 steps of the SNAP flow** :
    1) application + CPU executed action, application
    2) modelisation of the FPGA executed action,
    3) application + FPGA executed action

Contents

1. Access the software and documentation

The SNAP framework can be downloaded from github at: https://github.com/open-power/snap.

Follow the instructions in https://github.com/open-power/snap/blob/master/README.md#dependencies to find all you need to run the whole flow. This means downloading:

- the libraries (libcxl),
- the hardware component for the card you intend to use(**P**rocessor **S**ervice **L**ayer checkpoint file "b_route_design.dcp"),
- the simulation model (PSLSE: PSL **S**imulation **E**ngine) and tools to synthesize your design (Xilinx Vivado with the Ultrascale family chips)
- and the scripts to configure your FPGA with the binary file you created (capi-utils) on your deployment Power CPU system.

We will go through these different steps later in section 3

All education documentation can also be found at: https://developer.ibm.com/linuxonpower/capi/snap/ or direct link http://ibm.biz/powercapi_snap

2. Supported development and deployment environment

Development environment:

Development server (x86)	Minimum	Recommended	Command to check
Linux level	**Ubuntu 16.04.x LTS**	**Ubuntu 16.04.1 LTS**	lsb_release -a
	Red Hat 6.4		
	CentOS Linux 7		
	SUSE 11.4		
gcc	4.4.7	latest	gcc -v
Vivado	2016.4 (64 bit)	2016.4 (64 bit)	vivado -version
Vivado HLS	2017.4	2017.4	vivado_hls -version
Cadence ncsim (optional) *Default is Vivado xsim*	15.10-s019	15.10-s019	ncsim -version

– Deployment environment (server examples supporting CAPI SNAP)

IBM	**CAPI enabled Power servers**		
MTM	**PowerLinux**	**Code Name - P8**	**CAPI Capacity (per PCIe slots priority)**
8247-21L	Power S812L	Tuleta 1S/2U Linux Only	2x CAPI adapters per socket => 2 CAPI (C7-C6)
8247-22L	Power S822L	Tuleta 2S/2U Linux Only	2x CAPI adapters per socket => 4 CAPI (C7, C6, C5, C3)
8247-42L	Power S824L	Tuleta 2S/4U Linux Only w/GPU	2x CAPI adapters + 2GPUs (C3-C6)=> 4 CAPI (C3,C5,C6,C7)
MTM	**Cloud / Technical**	**Code Name - P8**	
8348-21C	Power Systems S812LC	Habanero - Cloud	2x CAPI adapters per socket => 2 CAPI (C3- C4)
8335-GCA	Power Systems S822LC	Firestone - MSP/Cloud	4 of the 5 PCIe slots are CAPI capable => 4 CAPI(C4-C1-C5-C2)

Test server (P8)	Minimum	Recommended	Command to check
Linux level	**Ubuntu 16.04.x LTS**	**Ubuntu 16.04.1 LTS**	lsb_release -a
	RedHat RHEL 7.3	latest RHEL 7.3	
gcc	4.4.7	latest	gcc -v
server Firmware : skiboot	5.1.13 (= to FW840.20)	latest	update_flash -d

3. Setup your environment on your x86 development server

Important: We will call **~snap and ~pslse** the directories in which you have installed snap, and pslse. Please adapt your paths accordingly. Having them in the same directory may be simpler for user.

1) Clone the snap github and then the pslse and prepare libraries.

```
git clone https://github.com/open-power/snap.git
git clone https://github.com/ibm-capi/pslse
```

2) Follow instructions on https://github.com/open-power/snap/blob/master/README.md#dependencies to have **your x86 development environment** installed
   a. No need to install "libcxl" as this library is already included in the "pslse". However this will be required in Power8 environment, as we use the actual PSL.
   b. PSL Checkpoint Files ("b_route_design.dcp") for the CAPI SNAP Design Kit
   c. Install Xilinx Vivado 2017.4 + Vivado HLS 2017.4 with Ultrascale family chips (KU060)
   d. Kconfig should be installed automatically (ncurses library may be needed)
   e. Do not install capi_utils. This is only for Power8 environment.
   f. Download PSLSE
3) Follow instructions on https://github.com/open-power/snap/tree/master/hardware/README.md to set your hardware-specific variables
   a. This will set Xilinx variables
   b. In your **snap** directory, create the file **snap_env.sh**

```
gedit snap_env.sh
```

In the file write the following 2 lines:

```
export PSLSE_ROOT=~pslse
export PSL_DCP=~path_to_CAPI_PSL_Checkpoints/ADKU3_Checkpoint/b_route_design.dcp
```

and save and close the file.
If you intend to use the N250S card, just adapt your path accordingly

```
export PSL_DCP=~path_to_CAPI_PSL_Checkpoints/N250S_Checkpoint/b_route_design.dcp
```

NOTE: you can also type the following commands to create the file :

```
echo "export PSLSE_ROOT=~pslse" > snap_env.sh
echo "export PSL_DCP=~path_to_CAPI_PSL_Checkpoints/ADKU3_Checkpoint/b_route_design.dcp" >>
snap_env.sh
```

4) Follow instructions on https://github.com/open-power/snap/blob/master/hardware/sim/README.md to set your simulation-specific variables. Vivado xsim is the default simulator and nothing needs to be set to have it run.

4. Setup your environment on your Power8 deployment server

1) Clone the snap github
   git clone https://github.com/open-power/snap.git
2) Following instructions on https://github.com/open-power/snap#dependencies you should have installed on **your Power8 deployment environment**.
   a. libcxl installation ➔(*for ubuntu*) sudo apt-get install libcxl-dev
   b. Image loader ➔ see https://github.com/ibm-capi/capi-utils

Your 2 environments are now fully prepared for SNAP.

5. Choose the card that will fit your requirements

SNAP 1.0 for Power8 supports actually 3 cards:

- Nallatech N250S with a Xilinx XCKU060 FPGA
- AlphaData ADM-PCIE-KU3 with a Xilinx XCKU060 FPGA
- Semptian NSA-121B with a Xilinx XCKU115 FPGA

Depending on the algorithm you want to port on the FPGA card, you will need to choose one of the card supported which brings you the resources you want to access.

Alpha-Data ADM-PCIE-KU3

Choose this card for:
External IO, Offload and DRAM

3.5MB Block Ram on FPGA

FPGA to Host Memory Access
Latency to/from FPGA: 0.8us
Bandwidth to FPGA: ~3.8GB/s reads and writes (CAPI limit)

8GB DDR3
Latency to FPGA: 230ns

Two 40Gb QSFP+ Ports
Future Use: Currently no Bridge to SNAP

Nallatech 250S

Choose this card for:
2TB of on-card Flash

3.5MB Block Ram on FPGA
FPGA to Host Memory Access
Latency to/from FPGA: 0.8us
Bandwidth to FPGA: ~3.8GB/s reads and writes (CAPI limit)

4GB DDR4 (on back of card)
Latency to FPGA: 184ns Read / 105ns write

Two 1TB NVMe sticks (1.92TB effective)
Latency to FPGA: ~0.8us
Bandwidth to FPGA: Read 1.8GB/s

Semptian NSA121B

Choose this card for:
Large FPGA, External IO, Offload and DDR4

8.4MB Block Ram on FPGA
FPGA to Host Memory Access
Latency to/from FPGA: 0.8us
Bandwidth to FPGA: ~3.8GB/s reads and writes (CAPI limit)

8GB DDR4 (on back of card)
Latency to FPGA: 184ns Read / 105ns write

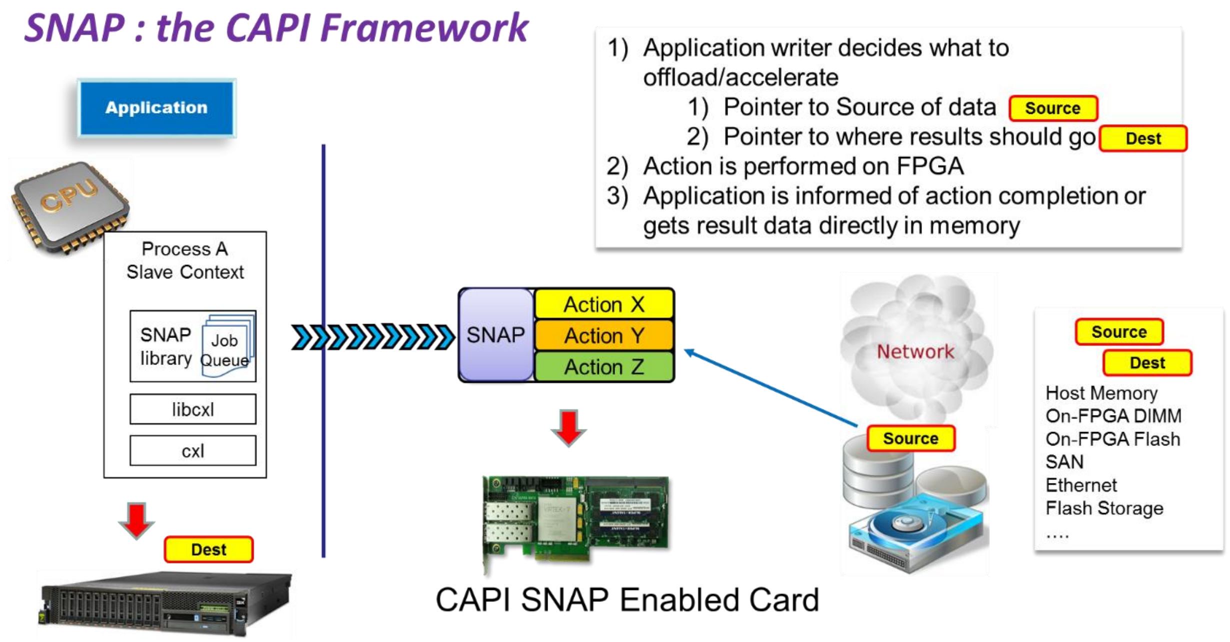
Two 10GE SFP+ Ports
Future Use: Currently no Bridge to SNAP

6. Understand how data are exchanged with FPGA

SNAP has been designed such a way that you can move data from a **Source** to a **Destination** without knowing the specific protocol to access the different memories. In the application, the coder decides where the data are located. These data locations can be either in the Host server memory, as well as on the card memory, or even outside the server and the card on an external storage accessed directly by the FPGA card.

The **data transfer is not handled by the host processor**, we just need to communicate the data source or destination address and size and the FPGA will handle it.

Important: We will call "**application**" the code executed on the server which calls the "**action**" containing the function to off-load and/or accelerate. This action can be "software" or "hardware" whether it is coded to be executed on CPU or FPGA.

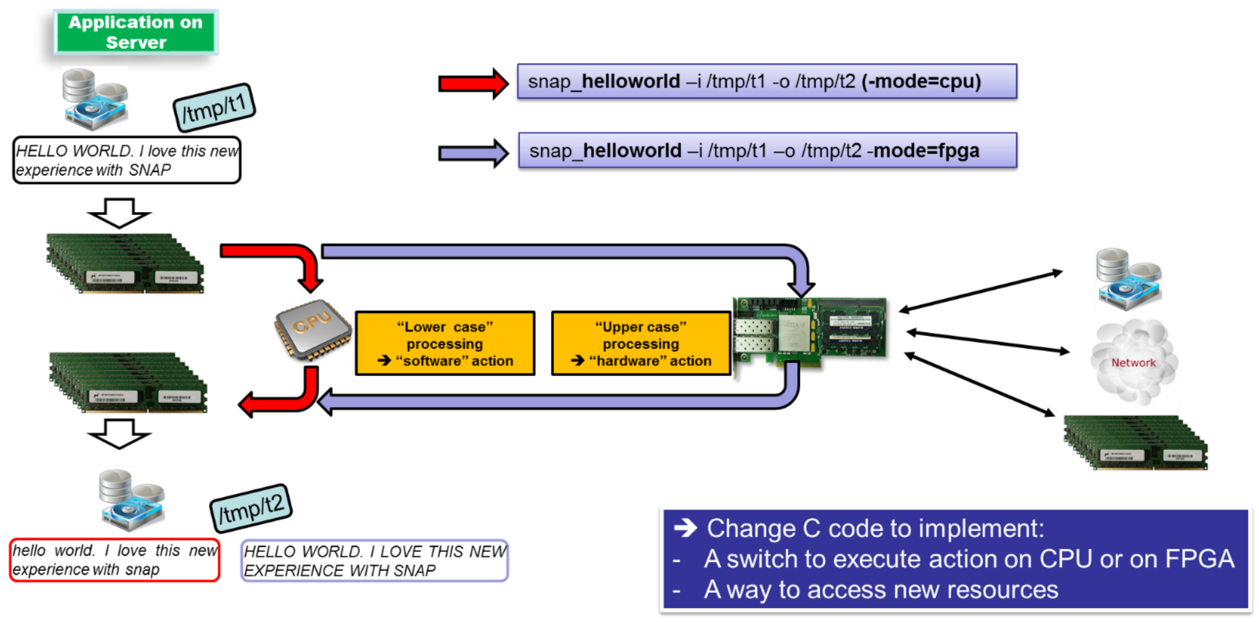

SNAP : the CAPI Framework
1) Application writer decides what to offload/accelerate
1) Pointer to Source of data
Source
2) Pointer to where results should go
Dest
2) Action is performed on FPGA
3) Application is informed of action completion or gets result data directly in memory
Application
CPU
Process A
Slave Context
SNAP library
Job Queue
libcxl
cxl
Dest
SNAP
Action X
Action Y
Action Z
CAPI SNAP Enabled Card
Network
Source
Source
Dest
Host Memory
On-FPGA DIMM
On-FPGA Flash
SAN
Ethernet
Flash Storage
....

7. Understand the HLS helloworld example

Let’s consider a simple example of a C code **changing the case of a text** read from a file and writing back the result into another file.
Once typed a text in a file t1, the user calls the program with path of input and output files as arguments. The different steps (red flow) are then typically:

1) Text is read and stored in the server’s memory (source).
2) CPU processes the text and writes back the result to the server memory (destination).
3) Text modified is then written from memory to disk.

Let’s now consider that we decide to “***export***” this “changing case processing” from the server processor to an external FPGA card. To keep that simple, we need to implement this switch so that the program call remains with no great changes (blue flow). To be able to differentiate which of these 2 paths we are using, we will use a **lower case** algorithm in the “**software**” action, so called because it runs on the CPU, and we will use a **upper case** algorithm in the “**hardware”** action so called because it will run on the FPGA.


Application on Server
/tmp/t1
HELLO WORLD. I love this new experience with SNAP
snap_helloworld –i /tmp/t1 -o /tmp/t2 (-mode=cpu)
snap_helloworld –i /tmp/t1 –o /tmp/t2 -mode=fpga
CPU
“Lower case” processing → “software” action
“Upper case” processing → “hardware” action
Network
/tmp/t2
hello world. I love this new experience with snap
HELLO WORLD. I LOVE THIS NEW EXPERIENCE WITH SNAP
→ Change C code to implement:
- A switch to execute action on CPU or on FPGA
- A way to access new resources

Through this simple example, we will discover some of the advantages and strength of SNAP tools to help exporting the previously CPU executed task.

8. Preliminary Step : configure SNAP environment

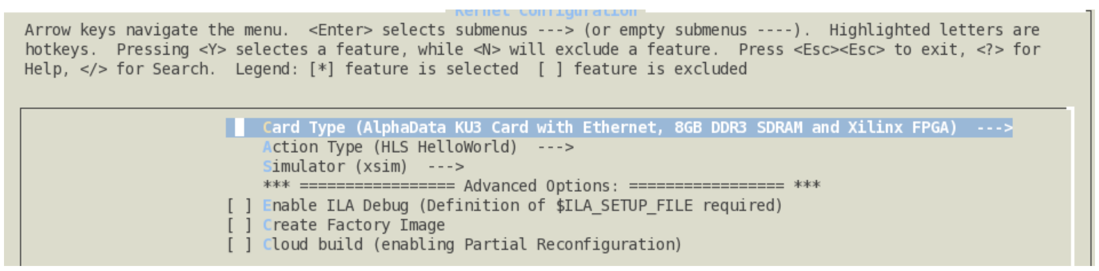
You now need to configure SNAP for the card and the action you will want to use.

For a first test, from snap directory, type **make snap_config**.

- ⇨ Make sure the terminal window is large enough to allow the opensource configurator to appear. Otherwise it behaves as if you opened it and closed it suddenly ! you get an **unexpected SNAP config done**
- ⇨ Menu uses simple kconfig opensource tool. Thus menus might change depending on the preselection we apply, it is not always possible to go back and forth without artefacts.
  For example selecting N250S/hls_nvme_memcopy/cloud mode will enforce nosim mode.
  the make model will then inform you it can build a model in "nosim" mode. Come back and select "xsim".
- ⇨ Do hesitate to use context helps.

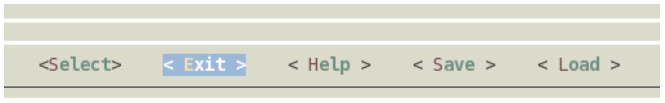
This opens a menu window as the following one. Let's select the Card Type **ADKU3** and the Action Type **HLS helloworld** with the default vivado **xsim** simulator.

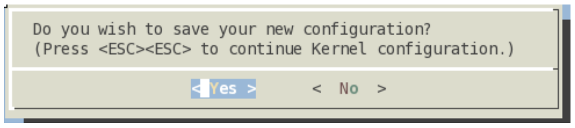
- ⇨ Use "space bar" or "return" to select depending if a submenu is present or not.



```
 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 (AlphaData KU3 Card with Ethernet, 8GB DDR3 SDRAM and Xilinx FPGA)  --->
                         Action Type (HLS HelloWorld)  --->
                         Simulator (xsim)  --->
                         *** ================= Advanced Options: ================= ***
                     [ ] Enable ILA Debug (Definition of $ILA_SETUP_FILE required)
                     [ ] Create Factory Image
                     [ ] Cloud build (enabling Partial Reconfiguration)
```

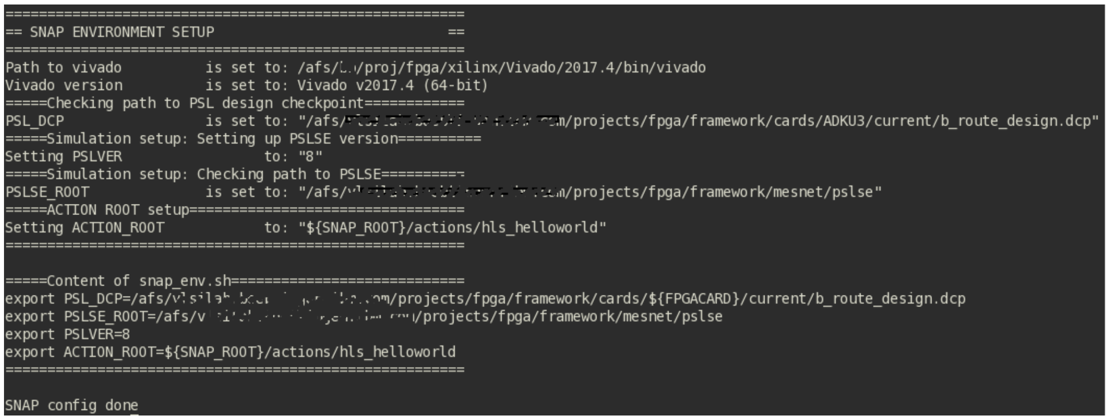
Then select **<Exit>** and **<Yes>** to save the configuration





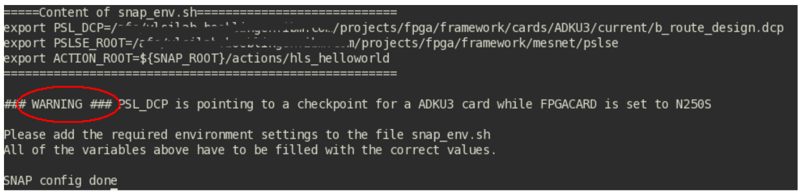
```
 Do you wish to save your new configuration?
 (Press <ESC><ESC> to continue Kernel configuration.)

                  < Yes >      <  No  >
```

If everything is set ok, then you should have displayed the SNAP ENVIRONMENT SETUP summary and the content of snap_env.sh which is the configuration file that we have prepared earlier.



```
======================================================
== SNAP ENVIRONMENT SETUP                           ==
======================================================
Path to vivado          is set to: /afs/[redacted]/proj/fpga/xilinx/Vivado/2017.4/bin/vivado
Vivado version          is set to: Vivado v2017.4 (64-bit)
=====Checking path to PSL design checkpoint===========
PSL_DCP                 is set to: "/afs/[redacted]m/projects/fpga/framework/cards/ADKU3/current/b_route_design.dcp"
=====Simulation setup: Setting up PSLSE version=========
Setting PSLVER                 to: "8"
=====Simulation setup: Checking path to PSLSE=========
PSLSE_ROOT              is set to: "/afs/[redacted]m/projects/fpga/framework/mesnet/pslse"
=====ACTION ROOT setup================================
Setting ACTION_ROOT            to: "${SNAP_ROOT}/actions/hls_helloworld"
======================================================

=====Content of snap_env.sh===========================
export PSL_DCP=/afs/[redacted]m/projects/fpga/framework/cards/${FPGACARD}/current/b_route_design.dcp
export PSLSE_ROOT=/afs/[redacted]/projects/fpga/framework/mesnet/pslse
export PSLVER=8
export ACTION_ROOT=${SNAP_ROOT}/actions/hls_helloworld
======================================================

SNAP config done
```

You may get some warnings if one of the 3 variables of this snap_env.sh is not set appropriately. It is recommended to correct it before going further.



```
=====Content of snap_env.sh===========================
export PSL_DCP=/[redacted]/projects/fpga/framework/cards/ADKU3/current/b_route_design.dcp
export PSLSE_ROOT=/[redacted]n/projects/fpga/framework/mesnet/pslse
export ACTION_ROOT=${SNAP_ROOT}/actions/hls_helloworld
======================================================

### WARNING ### PSL_DCP is pointing to a checkpoint for a ADKU3 card while FPGACARD is set to N250S

Please add the required environment settings to the file snap_env.sh
All of the variables above have to be filled with the correct values.

SNAP config done
```

In this example, you typically selected the N250S card with the path PSL_DCP set to ADKU3!

⇨ The selected hls_helloworld example appears as a new variable in snap_env.sh, as it will define the directory used for simulation and hardware tests.

⇨ Should you need to change anything in the configuration, you would need to make clean_config in the snap directory to reset those variables. (then copy again the snap_env.sh reference from your home dir to your snap dir)

⇨ SNAP contains several examples which can be used as references in the same manner.

⇨ We are now using Vivado 2107.4 version

9. Step 1: Run your application with your CPU-executable action

As we use dynamic libraries, we need to prepare our environment, this is easy to do while preparing the software tools :

```
cd ~/snap
make software
```

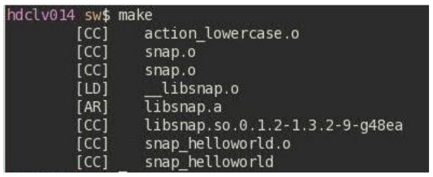
All the code for the hls_helloworld example can be found in ~snap/actions/hls_helloworld

Let's start with the **application** running with the **CPU-executable action**. Let's look to the files located in **sw** sub-directory: `cd ~snap/actions/hls_helloworld/sw`

You will find 2 C code files : **snap_helloworld.c** which is what we call the application which will be always run on the CPU (Power or x86) and **action_lowercase.c** which is the "software" action.

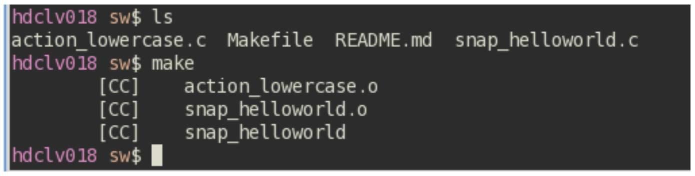
1) Let's first compile the code executing the command `make`

   The first time you'll get



```
hdclv014 sw$ make
        [CC]    action_lowercase.o
        [CC]    snap.o
        [CC]    snap.o
        [LD]    __libsnap.o
        [AR]    libsnap.a
        [CC]    libsnap.so.0.1.2-1.3.2-9-g48ea
        [CC]    snap_helloworld.o
        [CC]    snap_helloworld
```

   If you already made the file, you'll will only get :



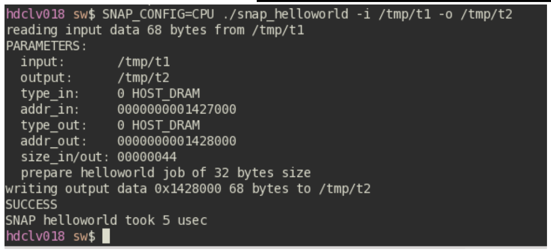
```
hdclv018 sw$ ls
action_lowercase.c  Makefile  README.md  snap_helloworld.c
hdclv018 sw$ make
        [CC]    action_lowercase.o
        [CC]    snap_helloworld.o
        [CC]    snap_helloworld
hdclv018 sw$ 
```

⇨ Note that a "make apps" from the snap dir will create all demo applications (here we just want to show you just the necessary files)

2) Create a text file (if not done previously) to be processed by the FPGA. Use a mix of lower and upper case to see the difference when using both actions. Remember "hardware" action will change all characters in Upper case and the "software" action which will change all characters in Lower case.

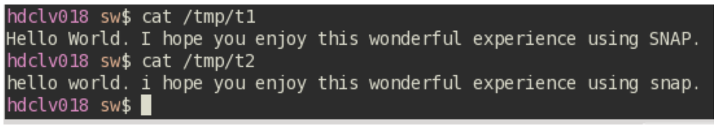
```
echo " Hello World. I hope you enjoy this wonderful experience using SNAP." > /tmp/t1
```

3) Run the application `SNAP_CONFIG=CPU ./snap_helloworld -i /tmp/t1 -o /tmp/t2`



```
hdclv018 sw$ SNAP_CONFIG=CPU ./snap_helloworld -i /tmp/t1 -o /tmp/t2
reading input data 68 bytes from /tmp/t1
PARAMETERS:
  input:       /tmp/t1
  output:      /tmp/t2
  type_in:     0 HOST_DRAM
  addr_in:     0000000001427000
  type_out:    0 HOST_DRAM
  addr_out:    0000000001428000
  size_in/out: 00000044
  prepare helloworld job of 32 bytes size
writing output data 0x1428000 68 bytes to /tmp/t2
SUCCESS
SNAP helloworld took 5 usec
hdclv018 sw$
```

Display the 2 files :



```
hdclv018 sw$ cat /tmp/t1
Hello World. I hope you enjoy this wonderful experience using SNAP.
hdclv018 sw$ cat /tmp/t2
hello world. i hope you enjoy this wonderful experience using snap.
hdclv018 sw$
```

Displaying the 2 files confirms that the “lower case” processing has been correctly done – using the “software” action code.

You can try the Debug option to see MMIO exchanges between application and action typing:

```
SNAP_TRACE=0xF SNAP_CONFIG=CPU ./snap_helloworld -i /tmp/t1 -o /tmp/t2
```

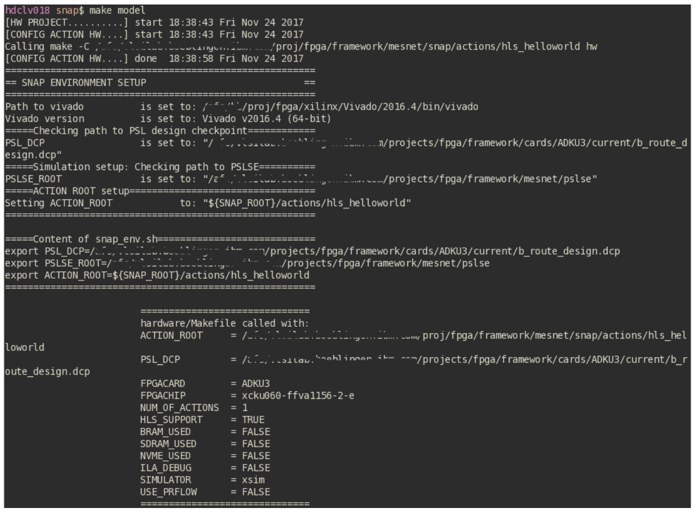
10. Step 2: run your application with a simulated model of your FPGA-executable action

Now that we have checked that the application works ok with the "software" action, let's use the "hardware" action. We'll keep using the **snap_helloworld** application located in **hls_helloworld/sw**, but will now use the "hardware" action located in **hls_helloworld/hw.**

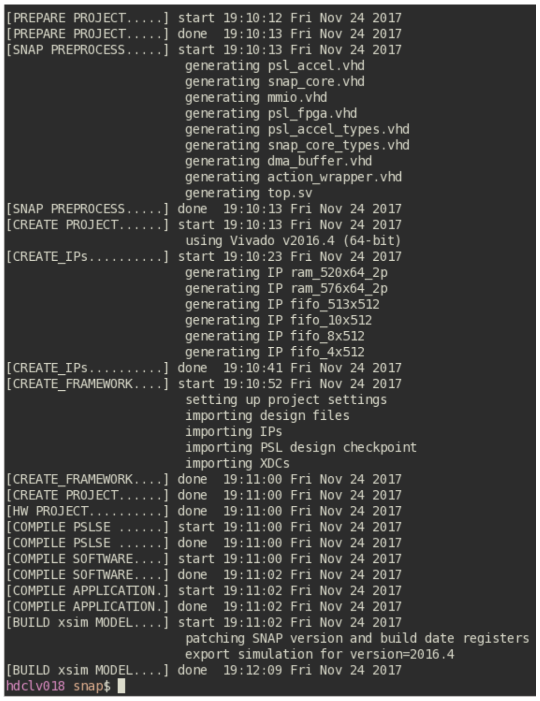
> *<u>Optional:</u> This "hardware" action can compiled alone:*
> *- from* *cd ~/snap/actions/hls_helloworld/hw* *, and type* *make*
> *or compiled with the whole SNAP design:*
> *- from* *cd ~/snap, and type* *make apps*
> *This compilation is optional since included in the following commands of the flow :*

Let's first build the model of the code so that you can run your application with a simulated model of your FPGA executable code.

1) From the snap directory `cd ~snap` , type `make model`

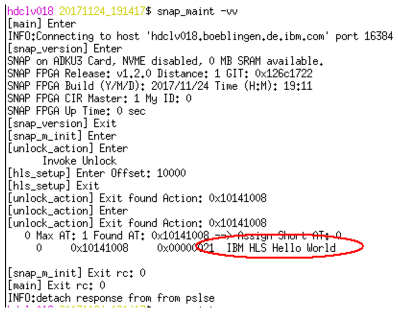


```
hdclv018 snap$ make model
[HW PROJECT.........] start 18:38:43 Fri Nov 24 2017
[CONFIG ACTION HW....] start 18:38:43 Fri Nov 24 2017
Calling make -C [redacted]/proj/fpga/framework/mesnet/snap/actions/hls_helloworld hw
[CONFIG ACTION HW....] done  18:38:58 Fri Nov 24 2017
========================================================
== SNAP ENVIRONMENT SETUP                             ==
========================================================
Path to vivado          is set to: [redacted]/proj/fpga/xilinx/Vivado/2016.4/bin/vivado
Vivado version          is set to: Vivado v2016.4 (64-bit)
=====Checking path to PSL design checkpoint============
PSL_DCP                 is set to: "/[redacted]n/projects/fpga/framework/cards/ADKU3/current/b_route_d
esign.dcp"
=====Simulation setup: Checking path to PSLSE==========
PSLSE_ROOT              is set to: "/[redacted]/projects/fpga/framework/mesnet/pslse"
=====ACTION ROOT setup==================================
Setting ACTION_ROOT            to: "${SNAP_ROOT}/actions/hls_helloworld"
========================================================

=====Content of snap_env.sh============================
export PSL_DCP=/[redacted]/projects/fpga/framework/cards/ADKU3/current/b_route_design.dcp
export PSLSE_ROOT=/[redacted]/projects/fpga/framework/mesnet/pslse
export ACTION_ROOT=${SNAP_ROOT}/actions/hls_helloworld
========================================================

                        ==============================
                        hardware/Makefile called with:
                        ACTION_ROOT     = /[redacted]proj/fpga/framework/mesnet/snap/actions/hls_hel
loworld
                        PSL_DCP         = /[redacted]/projects/fpga/framework/cards/ADKU3/current/b_r
oute_design.dcp
                        FPGACARD        = ADKU3
                        FPGACHIP        = xcku060-ffva1156-2-e
                        NUM_OF_ACTIONS  = 1
                        HLS_SUPPORT     = TRUE
                        BRAM_USED       = FALSE
                        SDRAM_USED      = FALSE
                        NVME_USED       = FALSE
                        ILA_DEBUG       = FALSE
                        SIMULATOR       = xsim
                        USE_PRFLOW      = FALSE
                        ==============================
```



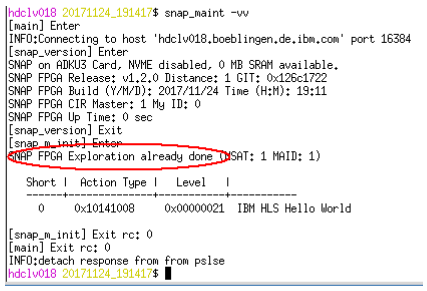
```
[PREPARE PROJECT.....] start 19:10:12 Fri Nov 24 2017
[PREPARE PROJECT.....] done  19:10:13 Fri Nov 24 2017
[SNAP PREPROCESS.....] start 19:10:13 Fri Nov 24 2017
                         generating psl_accel.vhd
                         generating snap_core.vhd
                         generating mmio.vhd
                         generating psl_fpga.vhd
                         generating psl_accel_types.vhd
                         generating snap_core_types.vhd
                         generating dma_buffer.vhd
                         generating action_wrapper.vhd
                         generating top.sv
[SNAP PREPROCESS.....] done  19:10:13 Fri Nov 24 2017
[CREATE PROJECT......] start 19:10:13 Fri Nov 24 2017
                         using Vivado v2016.4 (64-bit)
[CREATE_IPs..........] start 19:10:23 Fri Nov 24 2017
                         generating IP ram_520x64_2p
                         generating IP ram_576x64_2p
                         generating IP fifo_513x512
                         generating IP fifo_10x512
                         generating IP fifo_8x512
                         generating IP fifo_4x512
[CREATE_IPs..........] done  19:10:41 Fri Nov 24 2017
[CREATE_FRAMEWORK....] start 19:10:52 Fri Nov 24 2017
                         setting up project settings
                         importing design files
                         importing IPs
                         importing PSL design checkpoint
                         importing XDCs
[CREATE_FRAMEWORK....] done  19:11:00 Fri Nov 24 2017
[CREATE PROJECT......] done  19:11:00 Fri Nov 24 2017
[HW PROJECT..........] done  19:11:00 Fri Nov 24 2017
[COMPILE PSLSE ......] start 19:11:00 Fri Nov 24 2017
[COMPILE PSLSE ......] done  19:11:00 Fri Nov 24 2017
[COMPILE SOFTWARE....] start 19:11:00 Fri Nov 24 2017
[COMPILE SOFTWARE....] done  19:11:02 Fri Nov 24 2017
[COMPILE APPLICATION.] start 19:11:02 Fri Nov 24 2017
[COMPILE APPLICATION.] done  19:11:02 Fri Nov 24 2017
[BUILD xsim MODEL....] start 19:11:02 Fri Nov 24 2017
                         patching SNAP version and build date registers
                         export simulation for version=2016.4
[BUILD xsim MODEL....] done  19:12:09 Fri Nov 24 2017
hdclv018 snap$
```

2) If the build is succesful, then go into ~snap/hardware/sim typing `cd hardware/sim` and start the simulator typing `./run_sim`
3) A new window will popup.
   Type in it `snap_maint -vv` to execute the discovery mode. This step is mandatory in simulation simulation as well as in real hardware. It allows the SNAP logic to discover all the actions from all the cards. Should an application request an action, it will thus be assigned to the proper hardware.



```
hdclv018 20171124_191417$ snap_maint -vv
[main] Enter
INFO:Connecting to host 'hdclv018.boeblingen.de.ibm.com' port 16384
[snap_version] Enter
SNAP on ADKU3 Card, NVME disabled, 0 MB SRAM available.
SNAP FPGA Release: v1.2.0 Distance: 1 GIT: 0x126c1722
SNAP FPGA Build (Y/M/D): 2017/11/24 Time (H:M): 19:11
SNAP FPGA CIR Master: 1 My ID: 0
SNAP FPGA Up Time: 0 sec
[snap_version] Exit
[snap_m_init] Enter
[unlock_action] Enter
      Invoke Unlock
[hls_setup] Enter Offset: 10000
[hls_setup] Exit
[unlock_action] Exit found Action: 0x10141008
[unlock_action] Enter
[unlock_action] Exit found Action: 0x10141008
   0 Max AT: 1 Found AT: 0x10141008 --> Assign Short AT: 0
     0     0x10141008     0x00000021  IBM HLS Hello World

[snap_m_init] Exit rc: 0
[main] Exit rc: 0
INFO:detach response from from pslse
```

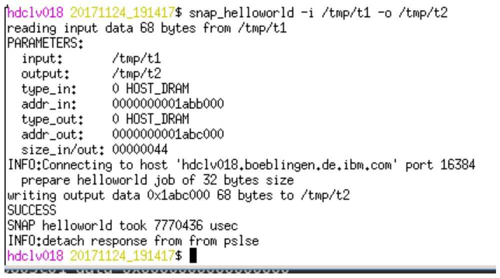
This shows that the action found is HLS Hello World,…as expected.
Running it a second time will confirm that this discovery step has already been done.



```
hdclv018 20171124_191417$ snap_maint -vv
[main] Enter
INFO:Connecting to host 'hdclv018.boeblingen.de.ibm.com' port 16384
[snap_version] Enter
SNAP on ADKU3 Card, NVME disabled, 0 MB SRAM available.
SNAP FPGA Release: v1.2.0 Distance: 1 GIT: 0x126c1722
SNAP FPGA Build (Y/M/D): 2017/11/24 Time (H:M): 19:11
SNAP FPGA CIR Master: 1 My ID: 0
SNAP FPGA Up Time: 0 sec
[snap_version] Exit
[snap_m_init] Enter
SNAP FPGA Exploration already done (NSAT: 1 MAID: 1)

   Short |  Action Type |   Level    |
   ------+--------------+------------+-----------
     0     0x10141008     0x00000021  IBM HLS Hello World

[snap_m_init] Exit rc: 0
[main] Exit rc: 0
INFO:detach response from from pslse
hdclv018 20171124_191417$ █
```

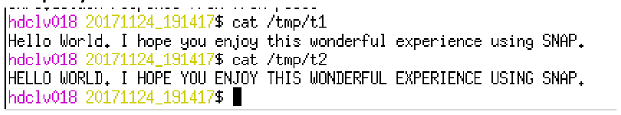
4) Then create a text file (if not done previously) to be processed by the FPGA.
   `echo " Hello World. I hope you enjoy this wonderful experience using SNAP." > /tmp/t1`

5) Run the application `snap_helloworld -i /tmp/t1 -o /tmp/t2` (Please note the difference compared to calling software action is: We don't use SNAP_CONFIG environmental variable. Actually here implies the default value of SNAP_CONFIG=FPGA)



```
hdclv018 20171124_191417$ snap_helloworld -i /tmp/t1 -o /tmp/t2
reading input data 68 bytes from /tmp/t1
PARAMETERS:
  input:       /tmp/t1
  output:      /tmp/t2
  type_in:     0 HOST_DRAM
  addr_in:     0000000001abb000
  type_out:    0 HOST_DRAM
  addr_out:    0000000001abc000
  size_in/out: 00000044
INFO:Connecting to host 'hdclv018.boeblingen.de.ibm.com' port 16384
  prepare helloworld job of 32 bytes size
writing output data 0x1abc000 68 bytes to /tmp/t2
SUCCESS
SNAP helloworld took 7770436 usec
INFO:detach response from from pslse
hdclv018 20171124_191417$ █
```

6) Display the 2 files :



```
hdclv018 20171124_191417$ cat /tmp/t1
Hello World. I hope you enjoy this wonderful experience using SNAP.
hdclv018 20171124_191417$ cat /tmp/t2
HELLO WORLD. I HOPE YOU ENJOY THIS WONDERFUL EXPERIENCE USING SNAP.
hdclv018 20171124_191417$ █
```

This confirms that the "upper case" processing has been correctly done – using the "hardware" code.

You can try other options before exiting

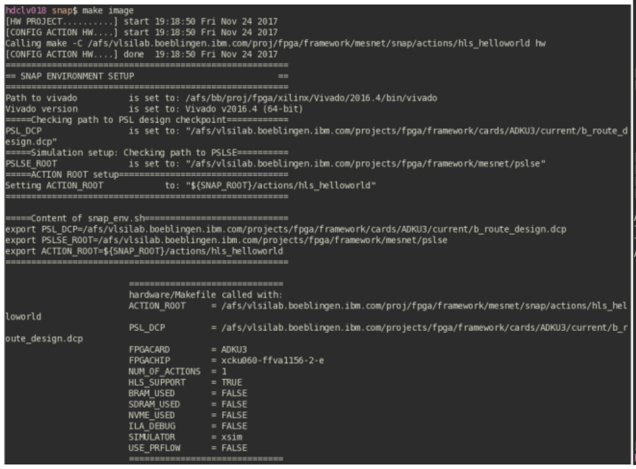
- Run the software code: `SNAP_CONFIG=CPU snap_helloworld -i /tmp/t1 -o /tmp/t2`
- Run the debug mode on FPGA code: `SNAP_TRACE=0xF snap_helloworld -i /tmp/t1 -o /tmp/t2`

You can now `exit` the simulator. The poped up terminal window will disappear.

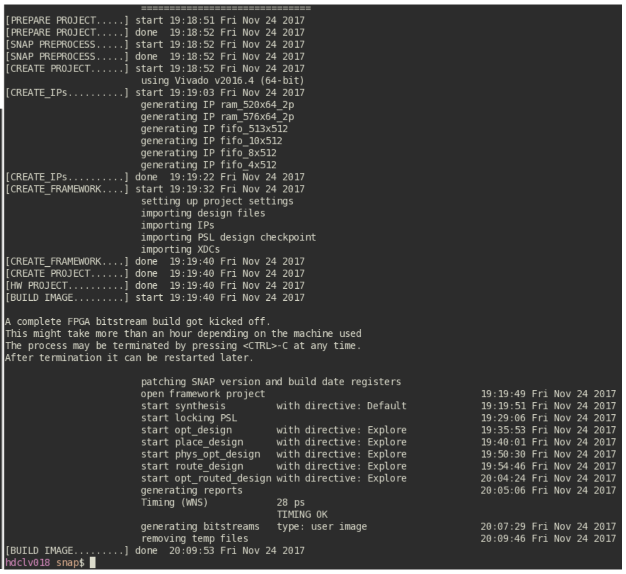
11. Step 3: Run your application with your FPGA-excutable action

Now that your application has been succefully executed with the simulated “hardware” action, let’s generate the “image” of the code which will be put into the FPGA.

1) From the snap directory `cd ~snap` , type `make image` ***(this takes an hour or so)***

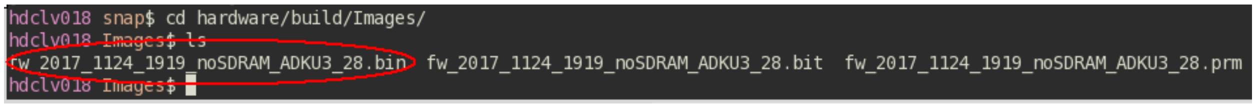


```
hdclv018 snap$ make image
[HW PROJECT..........] start 19:18:50 Fri Nov 24 2017
[CONFIG ACTION HW....] start 19:18:50 Fri Nov 24 2017
Calling make -C /afs/vlsilab.boeblingen.ibm.com/proj/fpga/framework/mesnet/snap/actions/hls_helloworld hw
[CONFIG ACTION HW....] done  19:18:50 Fri Nov 24 2017
=======================================================
== SNAP ENVIRONMENT SETUP                            ==
=======================================================
Path to vivado          is set to: /afs/bb/proj/fpga/xilinx/Vivado/2016.4/bin/vivado
Vivado version          is set to: Vivado v2016.4 (64-bit)
=====Checking path to PSL design checkpoint============
PSL_DCP                 is set to: "/afs/vlsilab.boeblingen.ibm.com/projects/fpga/framework/cards/ADKU3/current/b_route_d
esign.dcp"
=====Simulation setup: Checking path to PSLSE==========
PSLSE_ROOT              is set to: "/afs/vlsilab.boeblingen.ibm.com/projects/fpga/framework/mesnet/pslse"
=====ACTION ROOT setup=================================
Setting ACTION_ROOT            to: "${SNAP_ROOT}/actions/hls_helloworld"
=======================================================

=====Content of snap_env.sh============================
export PSL_DCP=/afs/vlsilab.boeblingen.ibm.com/projects/fpga/framework/cards/ADKU3/current/b_route_design.dcp
export PSLSE_ROOT=/afs/vlsilab.boeblingen.ibm.com/projects/fpga/framework/mesnet/pslse
export ACTION_ROOT=${SNAP_ROOT}/actions/hls_helloworld
=======================================================

                        ==============================
                        hardware/Makefile called with:
                        ACTION_ROOT     = /afs/vlsilab.boeblingen.ibm.com/proj/fpga/framework/mesnet/snap/actions/hls_hel
loworld
                        PSL_DCP         = /afs/vlsilab.boeblingen.ibm.com/projects/fpga/framework/cards/ADKU3/current/b_r
oute_design.dcp
                        FPGACARD        = ADKU3
                        FPGACHIP        = xcku060-ffva1156-2-e
                        NUM_OF_ACTIONS  = 1
                        HLS_SUPPORT     = TRUE
                        BRAM_USED       = FALSE
                        SDRAM_USED      = FALSE
                        NVME_USED       = FALSE
                        ILA_DEBUG       = FALSE
                        SIMULATOR       = xsim
                        USE_PRFLOW      = FALSE
                        ==============================
```



```
                        ==============================
[PREPARE PROJECT.....] start 19:18:51 Fri Nov 24 2017
[PREPARE PROJECT.....] done  19:18:52 Fri Nov 24 2017
[SNAP PREPROCESS.....] start 19:18:52 Fri Nov 24 2017
[SNAP PREPROCESS.....] done  19:18:52 Fri Nov 24 2017
[CREATE PROJECT......] start 19:18:52 Fri Nov 24 2017
                        using Vivado v2016.4 (64-bit)
[CREATE_IPs..........] start 19:19:03 Fri Nov 24 2017
                        generating IP ram_520x64_2p
                        generating IP ram_576x64_2p
                        generating IP fifo_513x512
                        generating IP fifo_10x512
                        generating IP fifo_8x512
                        generating IP fifo_4x512
[CREATE_IPs..........] done  19:19:22 Fri Nov 24 2017
[CREATE_FRAMEWORK....] start 19:19:32 Fri Nov 24 2017
                        setting up project settings
                        importing design files
                        importing IPs
                        importing PSL design checkpoint
                        importing XDCs
[CREATE_FRAMEWORK....] done  19:19:40 Fri Nov 24 2017
[CREATE PROJECT......] done  19:19:40 Fri Nov 24 2017
[HW PROJECT..........] done  19:19:40 Fri Nov 24 2017
[BUILD IMAGE.........] start 19:19:40 Fri Nov 24 2017

A complete FPGA bitstream build got kicked off.
This might take more than an hour depending on the machine used
The process may be terminated by pressing <CTRL>-C at any time.
After termination it can be restarted later.

                        patching SNAP version and build date registers
                        open framework project                                          19:19:49 Fri Nov 24 2017
                        start synthesis         with directive: Default                 19:19:51 Fri Nov 24 2017
                        start locking PSL                                               19:29:06 Fri Nov 24 2017
                        start opt_design        with directive: Explore                 19:35:53 Fri Nov 24 2017
                        start place_design      with directive: Explore                 19:40:01 Fri Nov 24 2017
                        start phys_opt_design   with directive: Explore                 19:50:30 Fri Nov 24 2017
                        start route_design      with directive: Explore                 19:54:46 Fri Nov 24 2017
                        start opt_routed_design with directive: Explore                 20:04:24 Fri Nov 24 2017
                        generating reports                                              20:05:06 Fri Nov 24 2017
                        Timing (WNS)            28 ps
                                                TIMING OK
                        generating bitstreams   type: user image                        20:07:29 Fri Nov 24 2017
                        removing temp files                                             20:09:46 Fri Nov 24 2017
[BUILD IMAGE.........] done  20:09:53 Fri Nov 24 2017
hdclv018 snap$
```

An image has been built and can be found in ~snap/hardware/build/Images



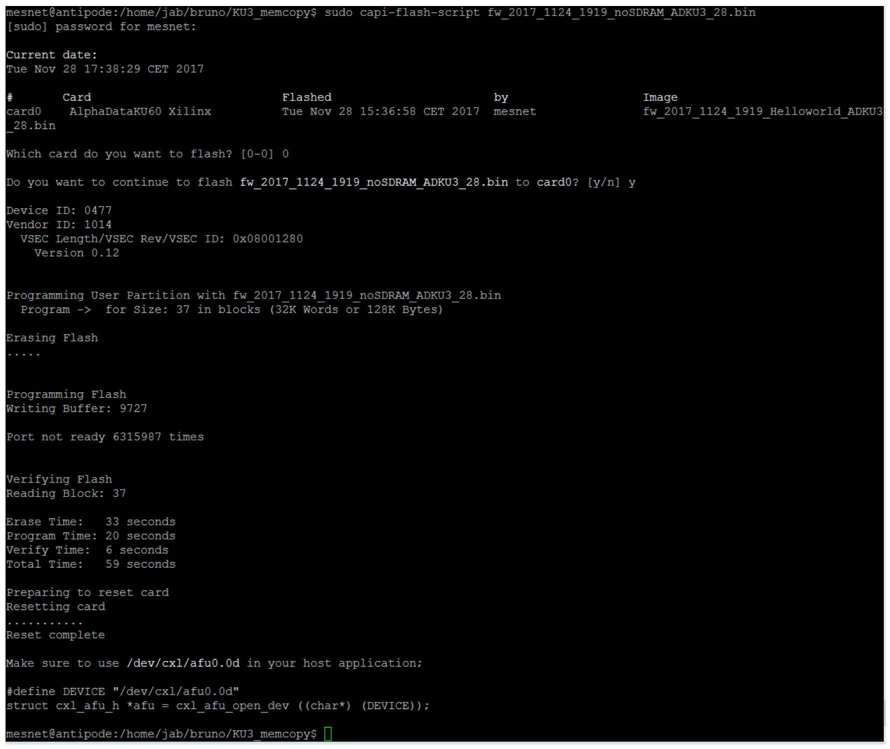
```
hdclv018 snap$ cd hardware/build/Images/
hdclv018 Images$ ls
fw_2017_1124_1919_noSDRAM_ADKU3_28.bin  fw_2017_1124_1919_noSDRAM_ADKU3_28.bit  fw_2017_1124_1919_noSDRAM_ADKU3_28.prm
hdclv018 Images$
```

You can check subdirectories in /home/opuser/snap/hardware/build/ for more information that Vivado generates.

2) Copy this image located into ~snap/hardware/build/Images into the Power8 server on which is plugged the FPGA card
3) Log to your Power8 server
4) Use capi-flash-script to download the binary image into the FPGA (cf. https://github.com/ibm-capi/capi-utils)

From the directory where you copied your bin file type:

```
sudo capi-flash-script fw_2017_1124_1919_noSDRAM_ADKU3_28.bin
```

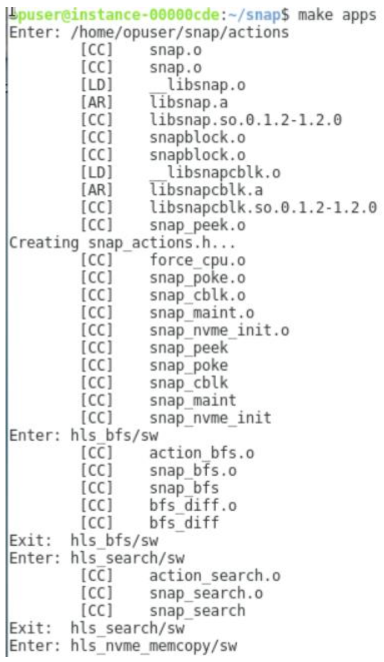


```
mesnet@antipode:/home/jab/bruno/KU3_memcopy$ sudo capi-flash-script fw_2017_1124_1919_noSDRAM_ADKU3_28.bin
[sudo] password for mesnet:

Current date:
Tue Nov 28 17:38:29 CET 2017

#       Card                           Flashed                        by                   Image
card0    AlphaDataKU60 Xilinx          Tue Nov 28 15:36:58 CET 2017  mesnet               fw_2017_1124_1919_Helloworld_ADKU3
_28.bin

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

Do you want to continue to flash fw_2017_1124_1919_noSDRAM_ADKU3_28.bin to card0? [y/n] y

Device ID: 0477
Vendor ID: 1014
  VSEC Length/VSEC Rev/VSEC ID: 0x08001280
    Version 0.12


Programming User Partition with fw_2017_1124_1919_noSDRAM_ADKU3_28.bin
  Program ->  for Size: 37 in blocks (32K Words or 128K Bytes)

Erasing Flash
.....


Programming Flash
Writing Buffer: 9727

Port not ready 6315987 times


Verifying Flash
Reading Block: 37

Erase Time:   33 seconds
Program Time: 20 seconds
Verify Time:  6 seconds
Total Time:   59 seconds

Preparing to reset card
Resetting card
...........
Reset complete

Make sure to use /dev/cxl/afu0.0d in your host application;

#define DEVICE "/dev/cxl/afu0.0d"
struct cxl_afu_h *afu = cxl_afu_open_dev ((char*) (DEVICE));

mesnet@antipode:/home/jab/bruno/KU3_memcopy$
```

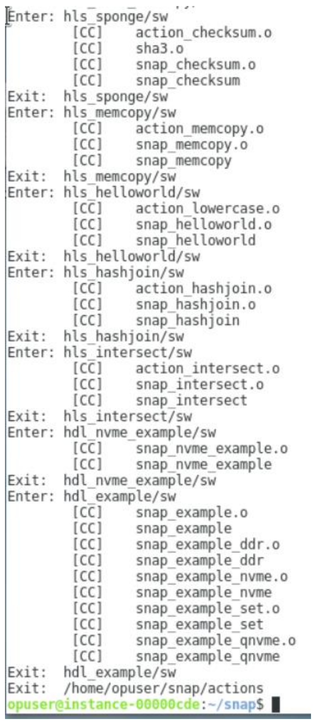
5) Clone the snap github

```
git clone https://github.com/open-power/snap.git
```

6) Enter directory **snap** by typing `cd ~snap`
and compile the software and all application and action `make software apps`

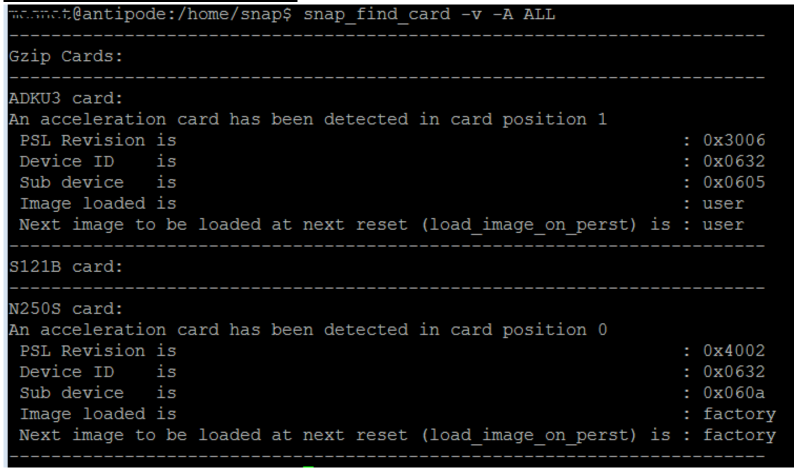


```
opuser@instance-00000cde:~/snap$ make apps
Enter: /home/opuser/snap/actions
         [CC]     snap.o
         [CC]     snap.o
         [LD]     __libsnap.o
         [AR]     libsnap.a
         [CC]     libsnap.so.0.1.2-1.2.0
         [CC]     snapblock.o
         [CC]     snapblock.o
         [LD]     __libsnapcblk.o
         [AR]     libsnapcblk.a
         [CC]     libsnapcblk.so.0.1.2-1.2.0
         [CC]     snap_peek.o
Creating snap_actions.h...
         [CC]     force_cpu.o
         [CC]     snap_poke.o
         [CC]     snap_cblk.o
         [CC]     snap_maint.o
         [CC]     snap_nvme_init.o
         [CC]     snap_peek
         [CC]     snap_poke
         [CC]     snap_cblk
         [CC]     snap_maint
         [CC]     snap_nvme_init
Enter: hls_bfs/sw
         [CC]     action_bfs.o
         [CC]     snap_bfs.o
         [CC]     snap_bfs
         [CC]     bfs_diff.o
         [CC]     bfs_diff
Exit:  hls_bfs/sw
Enter: hls_search/sw
         [CC]     action_search.o
         [CC]     snap_search.o
         [CC]     snap_search
Exit:  hls_search/sw
Enter: hls_nvme_memcopy/sw
```



```
Enter: hls_sponge/sw
         [CC]     action_checksum.o
         [CC]     sha3.o
         [CC]     snap_checksum.o
         [CC]     snap_checksum
Exit:  hls_sponge/sw
Enter: hls_memcopy/sw
         [CC]     action_memcopy.o
         [CC]     snap_memcopy.o
         [CC]     snap_memcopy
Exit:  hls_memcopy/sw
Enter: hls_helloworld/sw
         [CC]     action_lowercase.o
         [CC]     snap_helloworld.o
         [CC]     snap_helloworld
Exit:  hls_helloworld/sw
Enter: hls_hashjoin/sw
         [CC]     action_hashjoin.o
         [CC]     snap_hashjoin.o
         [CC]     snap_hashjoin
Exit:  hls_hashjoin/sw
Enter: hls_intersect/sw
         [CC]     action_intersect.o
         [CC]     snap_intersect.o
         [CC]     snap_intersect
Exit:  hls_intersect/sw
Enter: hdl_nvme_example/sw
         [CC]     snap_nvme_example.o
         [CC]     snap_nvme_example
Exit:  hdl_nvme_example/sw
Enter: hdl_example/sw
         [CC]     snap_example.o
         [CC]     snap_example
         [CC]     snap_example_ddr.o
         [CC]     snap_example_ddr
         [CC]     snap_example_nvme.o
         [CC]     snap_example_nvme
         [CC]     snap_example_set.o
         [CC]     snap_example_set
         [CC]     snap_example_qnvme.o
         [CC]     snap_example_qnvme
Exit:  hdl_example/sw
Exit:  /home/opuser/snap/actions
opuser@instance-00000cde:~/snap$
```

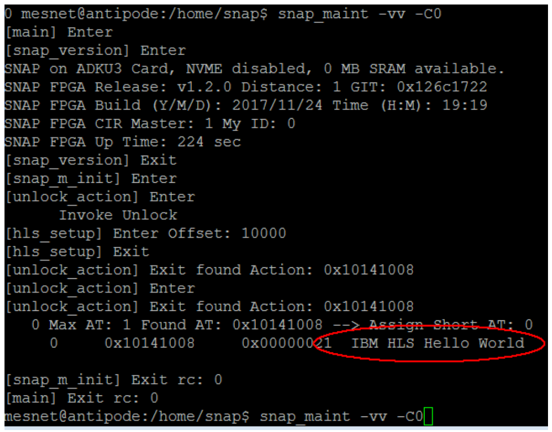
7) Execute the command source ./snap_path.sh to add all the paths you will need to PATH variable.
8) Find which card are available in the Power8 server you are connected to:
   snap_find_card -v -A ALL



```
[redacted]@antipode:/home/snap$ snap_find_card -v -A ALL
-------------------------------------------------------------------------
Gzip Cards:
-------------------------------------------------------------------------
ADKU3 card:
An acceleration card has been detected in card position 1
 PSL Revision is                                              : 0x3006
 Device ID    is                                              : 0x0632
 Sub device   is                                              : 0x0605
 Image loaded is                                              : user
 Next image to be loaded at next reset (load_image_on_perst) is : user
-------------------------------------------------------------------------
S121B card:
-------------------------------------------------------------------------
N250S card:
An acceleration card has been detected in card position 0
 PSL Revision is                                              : 0x4002
 Device ID    is                                              : 0x0632
 Sub device   is                                              : 0x060a
 Image loaded is                                              : factory
 Next image to be loaded at next reset (load_image_on_perst) is : factory
-------------------------------------------------------------------------
```

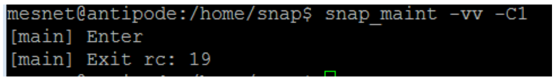
If you have 2 cards ADKU3, then the 2 cards slots are displayed. Using C0 or C1 will help you chosing the right one.

9) Run the discovery mode to locate on which FPGA card your action has been copied to by typing:
   snap_maint -vv -C0 or snap_maint -vv -C1 depending on the slots reported previously
   Only one of these commands should answer you



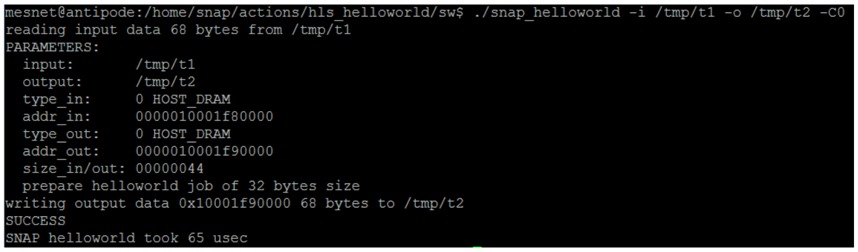
```
0 mesnet@antipode:/home/snap$ snap_maint -vv -C0
[main] Enter
[snap_version] Enter
SNAP on ADKU3 Card, NVME disabled, 0 MB SRAM available.
SNAP FPGA Release: v1.2.0 Distance: 1 GIT: 0x126c1722
SNAP FPGA Build (Y/M/D): 2017/11/24 Time (H:M): 19:19
SNAP FPGA CIR Master: 1 My ID: 0
SNAP FPGA Up Time: 224 sec
[snap_version] Exit
[snap_m_init] Enter
[unlock_action] Enter
      Invoke Unlock
[hls_setup] Enter Offset: 10000
[hls_setup] Exit
[unlock_action] Exit found Action: 0x10141008
[unlock_action] Enter
[unlock_action] Exit found Action: 0x10141008
   0 Max AT: 1 Found AT: 0x10141008 --> Assign Short AT: 0
     0      0x10141008      0x00000021  IBM HLS Hello World

[snap_m_init] Exit rc: 0
[main] Exit rc: 0
mesnet@antipode:/home/snap$ snap_maint -vv -C0
```

For your information the other slot will not answer:



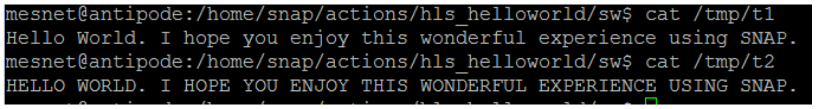
```
mesnet@antipode:/home/snap$ snap_maint -vv -C1
[main] Enter
[main] Exit rc: 19
```

10) Create a text file to be processed by the FPGA. Use a mix of lower and upper case to see the difference when using the "hardware" action which will change all characters in Upper case and the "software" action which will change all characters in Lower case.
    echo " Hello World. I hope you enjoy this wonderful experience using SNAP." > /tmp/t1
11) Go to the application you want to run typing
    cd /home/snap/actions/hls_helloworld/sw
12) Run the application ./snap_helloworld -i /tmp/t1 -o /tmp/t2 -C0 (or -C1)



```
mesnet@antipode:/home/snap/actions/hls_helloworld/sw$ ./snap_helloworld -i /tmp/t1 -o /tmp/t2 -C0
reading input data 68 bytes from /tmp/t1
PARAMETERS:
  input:       /tmp/t1
  output:      /tmp/t2
  type_in:     0 HOST_DRAM
  addr_in:     0000010001f80000
  type_out:    0 HOST_DRAM
  addr_out:    0000010001f90000
  size_in/out: 00000044
  prepare helloworld job of 32 bytes size
writing output data 0x10001f90000 68 bytes to /tmp/t2
SUCCESS
SNAP helloworld took 65 usec
```

13) Display the 2 files :



```
mesnet@antipode:/home/snap/actions/hls_helloworld/sw$ cat /tmp/t1
Hello World. I hope you enjoy this wonderful experience using SNAP.
mesnet@antipode:/home/snap/actions/hls_helloworld/sw$ cat /tmp/t2
HELLO WORLD. I HOPE YOU ENJOY THIS WONDERFUL EXPERIENCE USING SNAP.
```

This confirms that the “upper case” processing has been correctly done – using the “hardware” action code.

You can try other options before exiting

- Run the software code: `SNAP_CONFIG=CPU snap_helloworld -i /tmp/t1 -o /tmp/t2`
- Run the debug mode on FPGA code: `SNAP_TRACE=0xF snap_helloworld -i /tmp/t1 -o /tmp/t2`

12. Conclusion

So far you have been able to run a complete simple example on an FPGA through CAPI SNAP.

You have seen in detail the mecanism that allows submitting a simple task to the FPGA.

You have all the necessary files to undertand how the initially CPU executed task has been submitted to FPGA hardware.

Once this mecanism is understood, you can experiment further.

You can use some other provided examples to launch a large section of memory copy (hls_memcopy example) from host memory to the DDR of the FPGA. Then you can make your own FPGA calculation and get the result back with a second hls_memcopy as a last step.

It should give you a taste of the power of CAPI acceleration using POWER CAPI Technology.