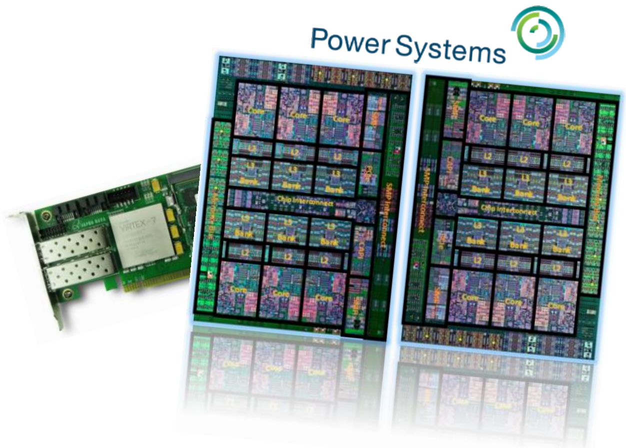
CAPI SNAP Education

hls_nvme_memcopy : howto?

V1.0



Generalities

1. **NVMe** stands for **non-Volatile-Memory expres**s. It is an open logical device interface specification for accessing non-volatile storage media attached via a PCI Express (PCIe) bus.
2. NVMe is supported on Nallatech N250S (with POWER8) and N250S+ (with POWER9) cards.
3. OpenPOWER CAPI SNAP NVMe hardware is based on a mechanism that's using SDRAM (DDR4 on FPGA board is used as a buffer) to handle data transfers.
4. Hardware bridge allows data transfers to or from the NVMe attached SSD devices from or to the SDRAM memory.
5. From there, the proposed application (***snap_nvme_memcopy***) demonstrates different kinds of transfers to and from :
   - Host memory (server memory)
   - SDRAM (on board DDR4)
   - NVMe devices
6. When Host memory is involved, a 2 steps transfer is performed :
   - step 1 from Host to SDRAM
   - step 2 from SDRAM to NVME (same process in the other way)
7. When a transfer is desired between the 2 NVMe devices, it requires to call ***snap_nvme_memcopy*** twice :
   - first to transfer from device #1 to SDRAM,
   - second to transfer from SDRAM to device #2)
8. There is a need for initialisation before using the NVMe attached devices.
9. Note that for simulation, it is required to have the DENALI models of the memories (use Cadence irun simulator)
10. Have a look at https://github.com/open-power/snap/blob/master/hardware/doc/NVMe.md

Architecture of the SNAP git files

SNAP

Compilation option :
export ACTION_ROOT=$SNAP_ROOT/actions/hls_nvme_memcopy
export SDRAM_USED=TRUE **(➔ "Enable SDRAM" must be set in Kconfig menu)**
export NVME_USED=TRUE **(➔ "Enable NVME" must be set in Kconfig menu)**

actions

- hls_memcopy
  - **sw**
    - **snap_nvme_memcopy.c** ← Main *application* program
    - **sw_action_nvme_memcopy.c** ← "CPU executed" *action* program
  - **hw**
    - **hls_action_nvme_memcopy.cpp** ← "FPGA executed" *action* program
    - **hls_action_nvme_memcopy.H** ← *Specific constants + "data exchanged" structures used by this action (i.e. Action_Release_Level)*
  - include
    - **action_memcopy.h** ← *Specific constants + "data exchanged" structures used by sw action <u>and</u> hw action + application (i.e. Action_Type)*
  - tests
  - doc ← *Regression test used to show how to run this hls_example*

hardware

- sim ← *Sim directory used for simulation (inside which we can run ./run_sim to simulate the current action)*
- build/Images ← *Location of bitstream files used to "burn" FPGA when current action is ready to be used in actual hardware*

software

- include
  - snap_types.h ← *Common constants + structures (i.e.snap_addr) used by all hls actions <u>and</u> applications*
- tools
  - snap_maint
  - snap_find_card ← *programs used to "discover" available actions, do basic settings and attach / detach actions*

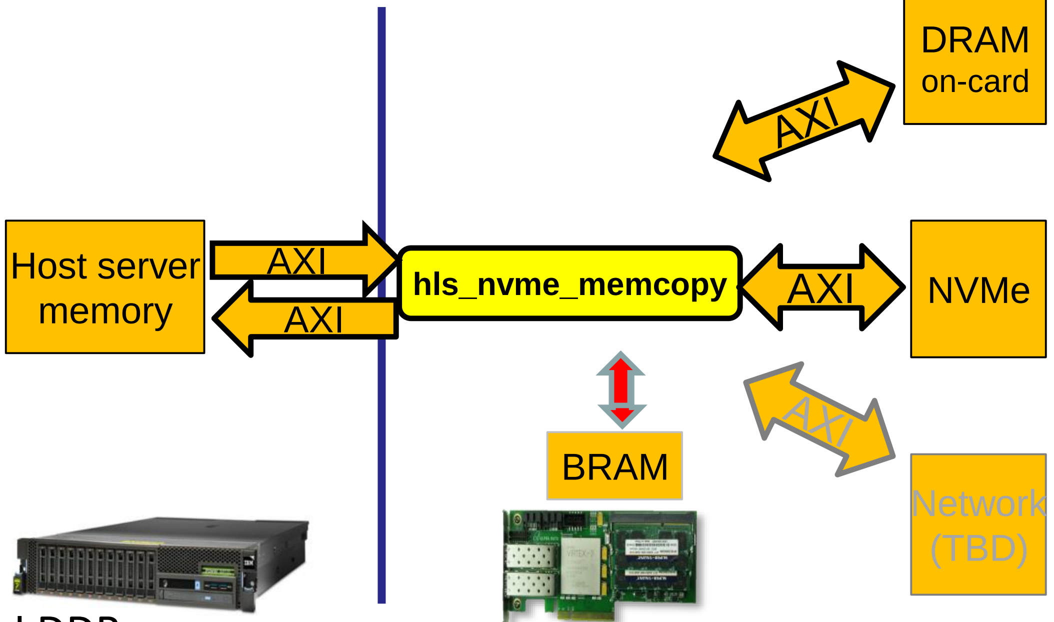
Action overview

Purpose: Transferring data between different resources :

- host memory,
- DDR,
- NVMe

When to use it:

- Understand Basic access to different interfaces
- Memcopy benchmarking



CAPI SNAP Enabled Card

Memory management:

- Application is managing address of Host memory and DDR
- Action is testing if size of transfer is greater than DRAM size (see constants)
- Size of buffer (BRAM) used to copy data can be configured (see constants)

Known limitations:

- HLS requires transfers to be 64 byte aligned and a size of multiples of 64 bytes
- DDR simulation model reads will return wrong values if non 64 bytes words or non initialized words are read (this is due to the simulation model only)
- If Source or Destination is NVME_SSD, size must be multiples of 512 (0x200)

Action usage (1/2)

Usage:

```
./snap_nvme_memcopy [-h] [-v, --verbose] [-V, --version]
 Usage: ./snap_nvme_memcopy [-h] [-v, --verbose] [-V, --version]
   -C, --card <cardno> can be (0...3)
   -i, --input  <file.bin>   input file  (HOST).
   -o, --output <file.bin>   output file (HOST).
   -A, --type-in  <NVME_SSD, HOST_DRAM, CARD_DRAM>.
   -a, --addr-in  <addr>     byte address in CARD_DRAM or NVME_SSD.
   -D, --type-out <NVME_SSD, HOST_DRAM, CARD_DRAM>.
   -d, --addr-out <addr>     byte address in CARD_DRAM or NVME_SSD.
   -n, --drv-id   <0/1>      drive_id if NVME_SSD is used (default: 0)
   -s, --size <size>         size of data (in bytes).
   -m, --mode <mode>         mode flags.
   -t, --timeout             Timeout in sec to wait for done. (10 sec default)
   -X, --verify              verify result if possible
   -N, --no_irq                 Disable Interrupts
```

Example :

```
export SNAP_TRACE=0x0
snap_maint -vv -C0
snap_nvme_init -vv -C0
…
echo move 4kB from Host to DDR@0x0 and back from DDR@0x0 to Host
rm t2; dd if=/dev/urandom of=in4k bs=1K count=4
./snap_nvme_memcopy -A HOST_DRAM -D NVME_SSD  -i in4k.bin -d 0x0
echo 4kout.bin collected from address 0x0 of SSD1 in 8 blocs of 512 (size 0x1000)
./snap_nvme_memcopy -A NVME_SSD -D HOST_DRAM -a 0x0 -o out4k.bin -s 0x1000

diff in4k.bin out4k.bin
 if diff in4k.bin out4k.bin >/dev/null;then echo "RC=$rc file_diff ok";else
   echo -e "$t RC=$rc file_diff is wrong\n$del";exit 1;
 fi
```

Options: *(default option in **bold**)*

SNAP_TRACE = 0x0 ➔ no debug trace
SNAP_TRACE = 0xF ➔ full debug trace

SNAP_CONFIG = FPGA ➔ hardware execution
SNAP_CONFIG = CPU ➔ software execution

Action usage (2/2)

Different cases that can be run

```
 WARNING : All data transfers to and from NVME_SSDs are buffered in CARD_DRAM :
 Check #define DRAM_ADDR_TO_SSD  0x00000000 and #define DRAM_ADDR_FROM_SSD 0x80000000
 in $ACTION_ROOT/hw/hw_action_nvme_memcopy.H
 Usage Examples:
Before using NVME following command must be run :
 ${SNAP_ROOT}/software/tools/snap_maint -Cn #n is card number to attach your action !
 ${SNAP_ROOT}/software/tools/snap_nvme_init prior to use NVME memory driver !

  echo create a 128kB file with random data ...wait...
  dd if=/dev/urandom of=in.bin bs=1k count=128
  echo create a 512MB file with random data ...wait...
  dd if=/dev/urandom of=in.bin bs=1M count=512
  snap_nvme_memcopy -A HOST_DRAM -D HOST_DRAM -i in.bin -o out.bin ...
  snap_nvme_memcopy -A HOST_DRAM -D CARD_DRAM -i in.bin -d 0xD000 ...
  snap_nvme_memcopy -A HOST_DRAM -D NVME_SSD  -i in.bin -d 0xE000 ...

  snap_nvme_memcopy -A CARD_DRAM -D HOST_DRAM -a 0xD000 -o out.bin -s 0x200 ...
  snap_nvme_memcopy -A CARD_DRAM -D NVME_SSD  -a 0xD000 -d 0xE000 -s 0x200 ...
  snap_nvme_memcopy -A CARD_DRAM -D CARD_DRAM -a 0xD000 -d 0xD200 -s 0x200 ...

  snap_nvme_memcopy -A NVME_SSD -D CARD_DRAM -a 0xE000 -d 0xD000 -s 0x200 ...
  snap_nvme_memcopy -A NVME_SSD -D HOST_DRAM -a 0xE000 -o out.bin -s 0x200 ...

 1) In Above examples, all addresses are byte address.
    CARD_DRAM address limit is 0x1_0000_0000  (  4294967296 Bytes =   4GB)
    NVME_SSD  address limit is 0xDF_9035_6000 (960197124096 Bytes = 960GB) for one drive.
    If Source or Destination is NVME_SSD, size must be multiples of 512 (0x200)
 2) NVME to NVME is not directly supported,
    but can be done by calling snap_nvme_memcopy twice.
 3) HOST to and from NVME is actually performed using 2 hardware steps with a SDRAM buffer in the middle,
    !! See WARNING ABOVE !!
```

Take in account that running on a simulator is far more slow than an execution on a FPGA:
➔ moving 512MB with a simulator is a HUGE challenge. May be just trying 4K should be sufficient !

Simple transfer tests

Default SSD is 0 if not specified

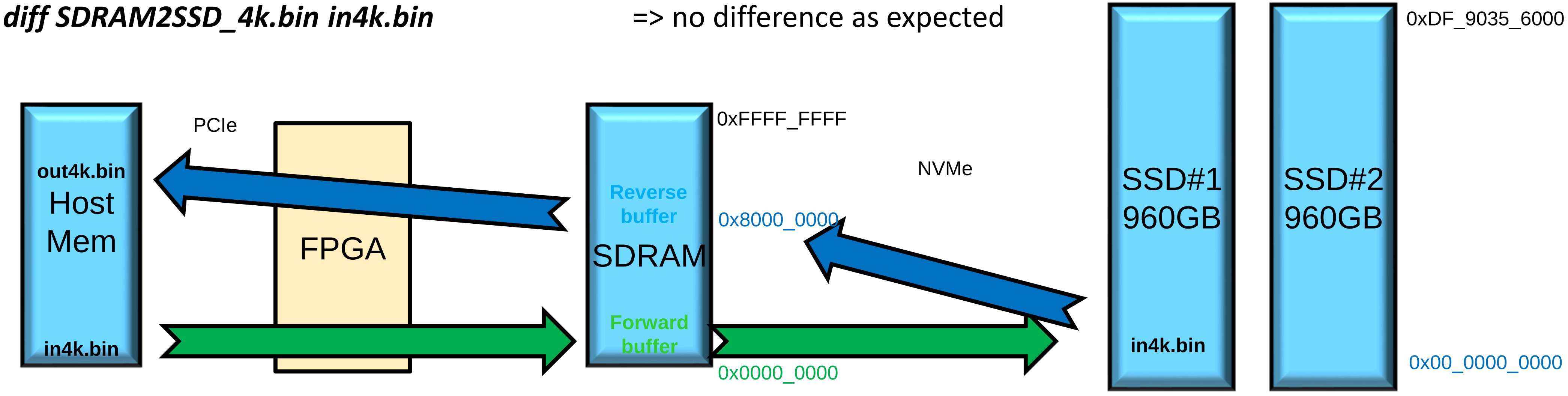
Purpose: Transferring 4kB data from host file to NVMe and get it back for comparison:

- File creation : ***dd if=/dev/urandom of=in4k.bin bs=1k count=4***
- in4k.bin file copied into address 0x0 of SSD 1
- ***./snap_nvme_memcopy -A HOST_DRAM -D NVME_SSD -i in4k.bin -d 0x0***
- 4kout.bin collected from address 0x0 of SSD1 in 8 blocs of 512 (size 0x1000)
- ***./snap_nvme_memcopy -A NVME_SSD -D HOST_DRAM -a 0x0 -o out4k.bin -s 0x1000***
- ***diff in4k.bin out4k.bin*** => no difference as expected

Default buffers locations, see :
$ACTION_ROOT/hw/hw_action_nvme_memcopy.H

Check SDRAM (used as buffer) content :

- ***./snap_nvme_memcopy -A CARD_DRAM -D HOST_DRAM -a 0x00000000 -o SDRAM2SSD_4k.bin -s 0x1000***
- ***./snap_nvme_memcopy -A CARD_DRAM -D HOST_DRAM -a 0x80000000 -o SSD2SDRAM_4k.bin -s 0x1000***
- ***diff SDRAM2SSD_4k.bin SSD2SDRAM_4k.bin*** => no difference as expected
- ***diff SDRAM2SSD_4k.bin in4k.bin*** => no difference as expected



Simple transfer tests

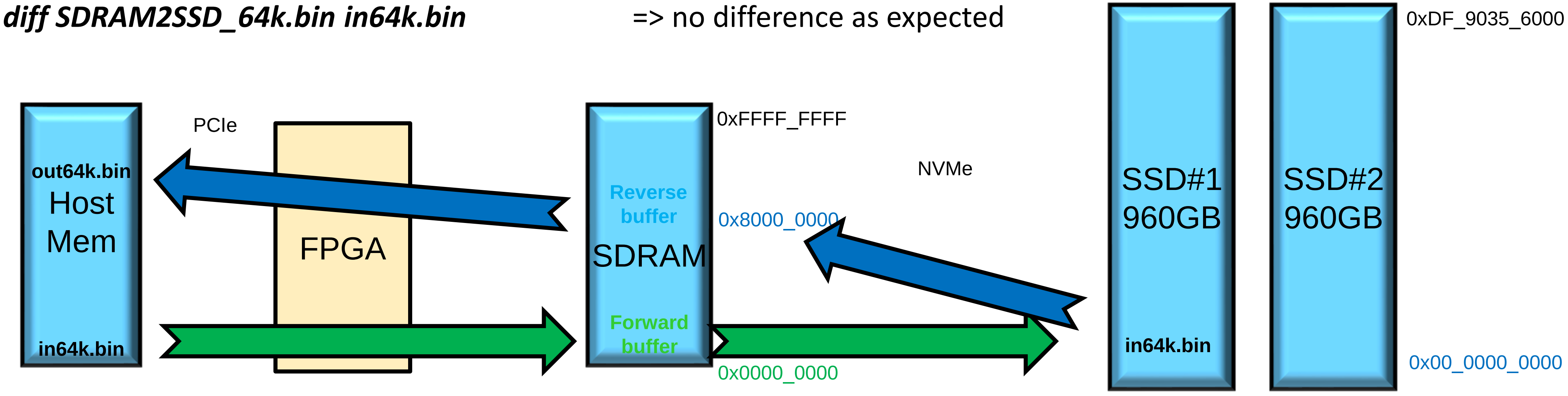
Purpose: Transferring 64kB data from host file to NVMe and get it back for comparison:

- File creation : ***dd if=/dev/urandom of=in64k.bin bs=1k count=64***
- in64k.bin file copied into address 0x0 of SSD 1
- ***./snap_nvme_memcopy -A HOST_DRAM -D NVME_SSD -i in64k.bin -d 0x0***
- 4kout.bin collected from address 0x0 of SSD1 in 128 blocs of 512 (size 0x10000)
- ***./snap_nvme_memcopy -A NVME_SSD -D HOST_DRAM -a 0x0 -o out64k.bin -s 0x10000***
- ***diff in64k.bin out64k.bin*** => no difference as expected

Default buffers locations, see :
$ACTION_ROOT/hw/hw_action_nvme_memcopy.H

Check SDRAM (used as buffer) content :

- ***./snap_nvme_memcopy -A CARD_DRAM -D HOST_DRAM -a 0x00000000 -o SDRAM2SSD_64k.bin -s 0x10000***
- ***./snap_nvme_memcopy -A CARD_DRAM -D HOST_DRAM -a 0x80000000 -o SSD2SDRAM_64k.bin -s 0x10000***
- ***diff SDRAM2SSD_64k.bin SSD2SDRAM_64k.bin*** => no difference as expected
- ***diff SDRAM2SSD_64k.bin in64k.bin*** => no difference as expected



Simple transfer tests

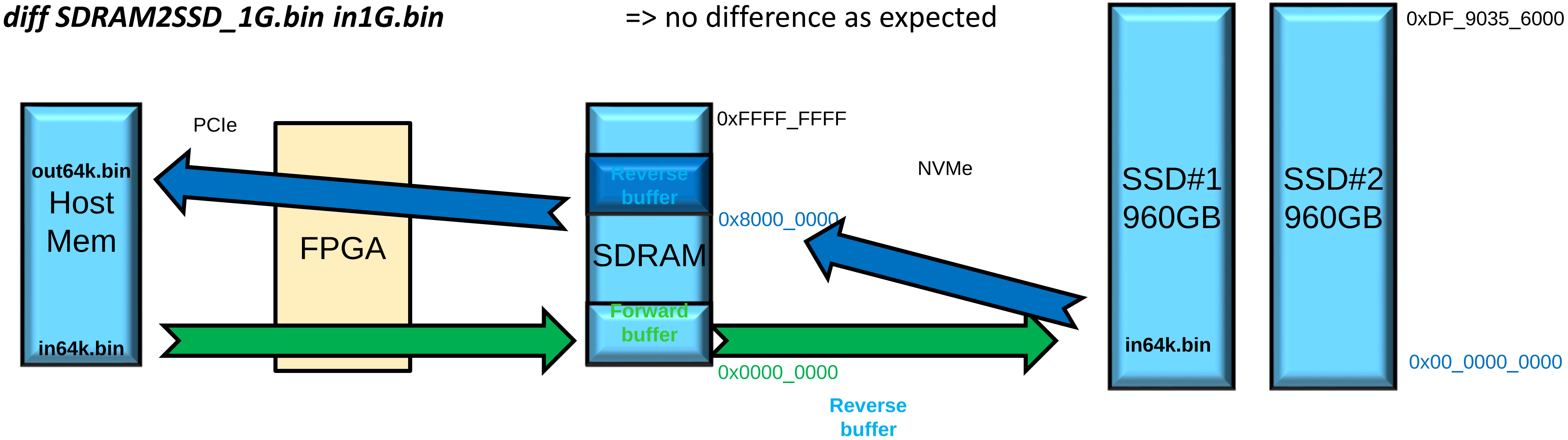
Purpose: Transferring 1GB data from host file to NVMe and get it back for comparison:

- File creation : ***dd if=/dev/urandom of=in1G.bin bs=1M count=1024***
- in64k.bin file copied into address 0x0 of SSD 1
- *./snap_nvme_memcopy -A HOST_DRAM -D NVME_SSD -i in1G.bin -d 0x0*
- 4kout.bin collected from address 0x0 of SSD1 in XXX blocs of 512 (size 0x4000_0000)
- *./snap_nvme_memcopy -A NVME_SSD -D HOST_DRAM -a 0x0 -o out1G.bin -s 0x40000000*
- ***diff in1G.bin out1G.bin*** => no difference as expected

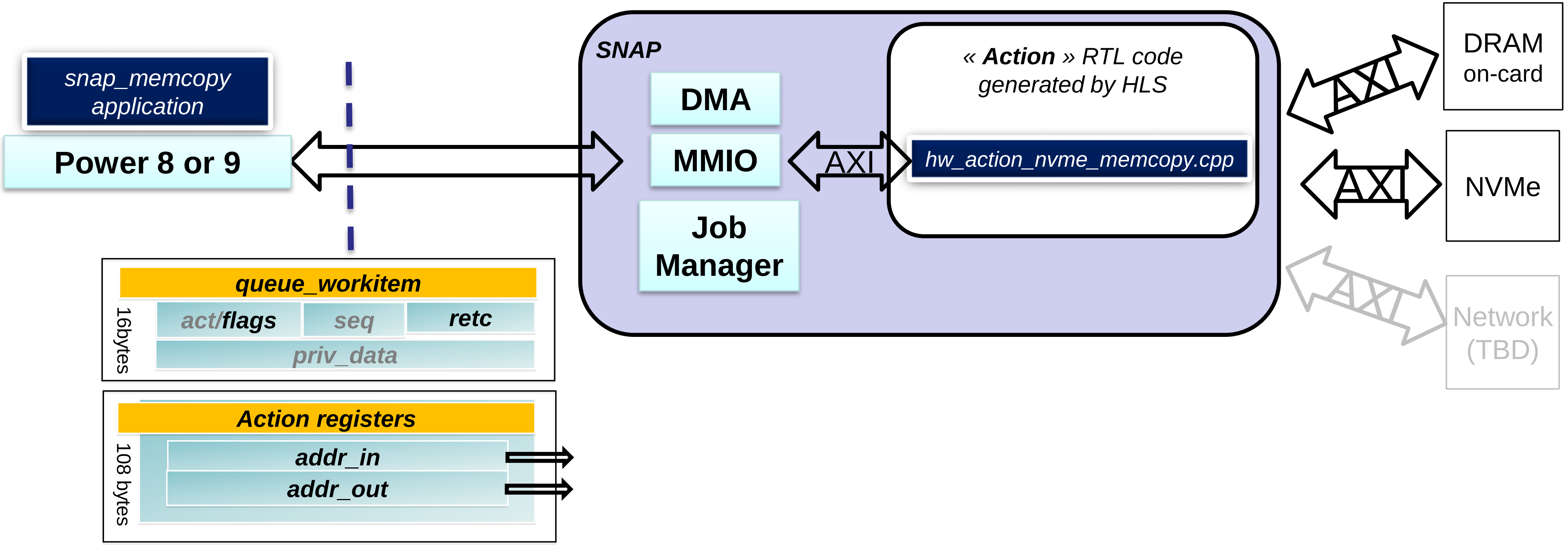
Check SDRAM (used as buffer) content :

- ***./snap_nvme_memcopy -A CARD_DRAM -D HOST_DRAM -a 0x00000000 -o SDRAM2SSD_1G.bin -s 0x40000000***
- ***./snap_nvme_memcopy -A CARD_DRAM -D HOST_DRAM -a 0x80000000 -o SSD2SDRAM_1G.bin -s 0x40000000***
- ***diff SDRAM2SSD_1G.bin SSD2SDRAM_1G.bin*** => no difference as expected
- ***diff SDRAM2SSD_1G.bin in1G.bin*** => no difference as expected

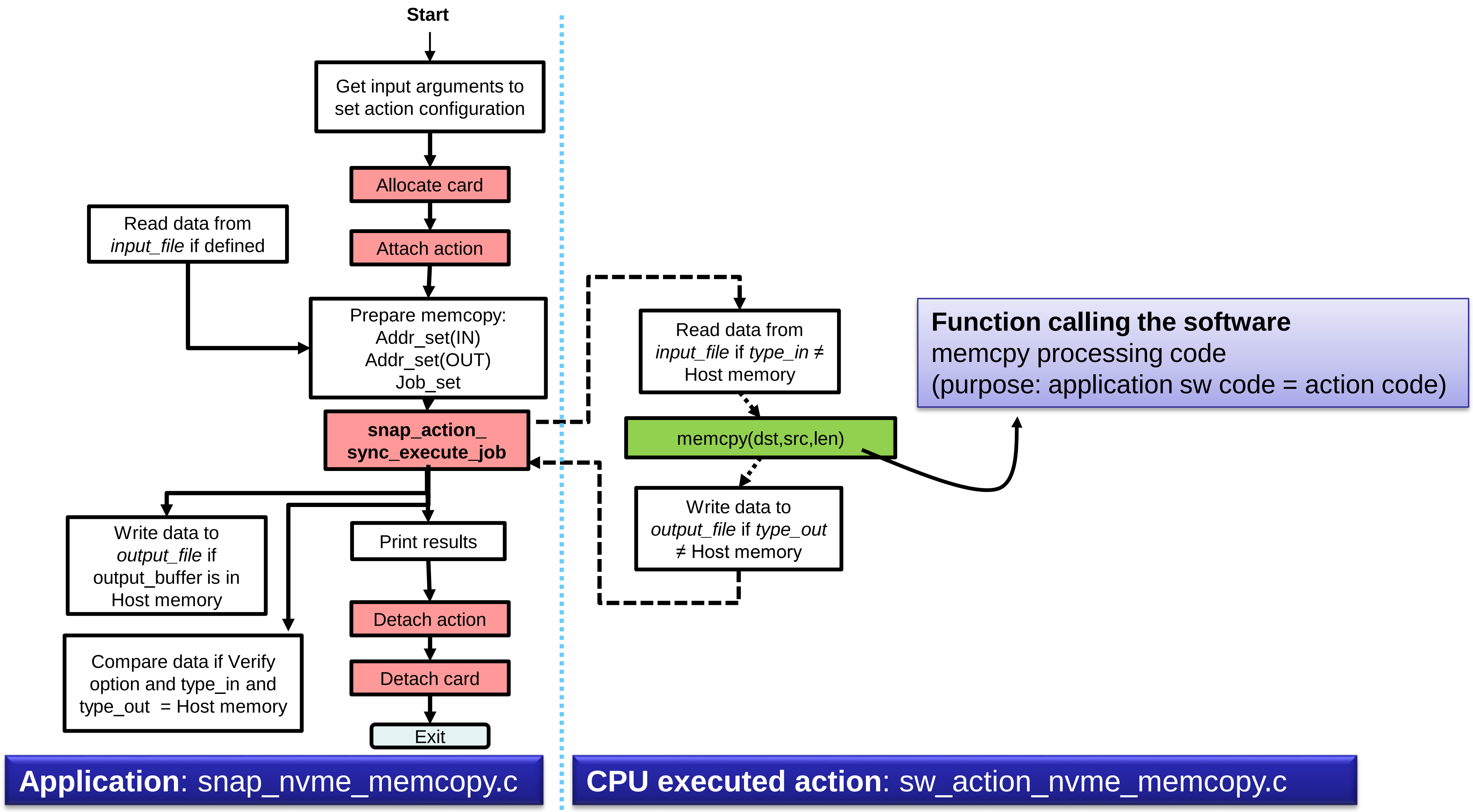
Default buffers locations, see :
$ACTION_ROOT/hw/hw_action_nvme_memcopy.H



nvme_memcopy registers



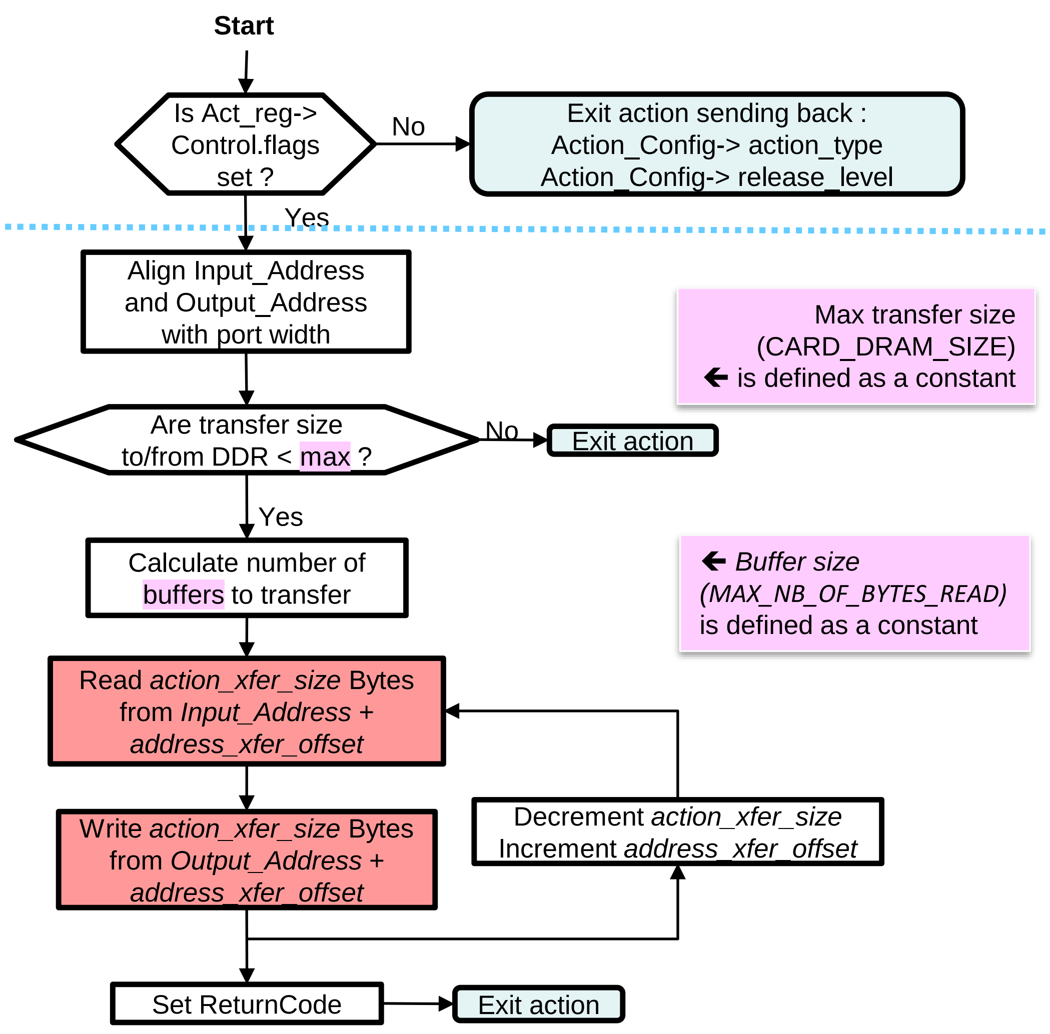
Application Code + software action code : what's in it?



Hardware action Code : what's in it?

hls_action

process_action



Used during discovery phase only

Max transfer size (CARD_DRAM_SIZE) ← is defined as a constant

← *Buffer size (MAX_NB_OF_BYTES_READ)* is defined as a constant

FPGA executed Action: hls_action_nvme_memcopy.cpp

Constants - Ports

IBM Power Systems

Constants: ➔ $ACTION_ROOT = snap/actions/hls_nvme_memcopy

Constant name	Value	Type	Definition location	Usage
MEMCOPY_ACTION_TYPE	0x10141000	Fixed	$ACTION_ROOT/include/action_nvme_memcopy.h	memcopy ID - list is in snap/ActionTypes.md
RELEASE_LEVEL	0x00000001ll ../h	Variable	$ACTION_ROOT/hw/hw_action_nvme_memcopy.**H**	release level – user defined
MAX_NB_OF_BYTES_READ	(256 * 1024)	Variable	$ACTION_ROOT/hw/hw_action_nvme_memcopy.**H**	Max size in Bytes of the buffer for read/write access
MAX_NB_OF_WORDS_READ	(MAX_NB_OF_BYTES_READ/BPERDW)	*Operation*	$ACTION_ROOT/hw/hw_action_nvme_memcopy.**H**	Max size in 64B words of the buffer for read/write access
CARD_DRAM_SIZE	(4 * 1024 *1024 * 1024)	Variable	$ACTION_ROOT/hw/hw_action_nvme_memcopy.**H**	Max size of the DDR - prevents from moving data with a size larger than this value

Ports used:

Ports name	Description	Enabled
din_gmem	Host memory data bus input Addr : 64bits - Data : 512bits	Yes
dout_gmem	Host memory data bus output Addr : 64bits - Data : 512bits	Yes
d_ddrmem	DDR3 - DDR4 data bus in/out Addr : 33bits - Data : 512bits	Yes
nvme	NVMe data bus in/out Addr : 32bits - Data : 32bits	Yes

MMIO Registers

IBM Power Systems

0x3C42	0x108	0x188	Private Data				c0febabe	
0x3C43	0x10C	0x18C	Private Data				deadbeef	
action_reg.Data		***Action specific - user defined - need to stay in 108 Bytes***						
memcopy_job_t		***This is the way for application and action to exchange information through this set of registers***						
	Write@	Read@	3	2	1	0	Typical Write value	Typical Read value
0x3C44	0x110	0x190	snap_addr.**addr_in** (LSB)					
0x3C45	0x114	0x194	snap_addr.**addr_in** (MSB)					
0x3C46	0x118	0x198	snap_addr_in.**size**					
0x3C47	0x11C	0x19C	snap.addr_in.**flags** (SRC, DST, ...)		snap.addr_in.**type** (HOST, DRAM, NVME,..)			
0x3C48	0x120	0x1A0	snap_addr.**addr_out** (LSB)					
0x3C49	0x124	0x1A4	snap_addr.**addr_out** (MSB)					
0x3C4A	0x128	0x1A8	snap.addr_out.**size**					
0x3C4B	0x12C	0x1AC	snap.addr_out.**flags** (SRC, DST, ...)		snap.addr_out.**type** (HOST, DRAM, NVME,..)			
	0x130	0x1B0						
	0x134	0x1B4						
	0x138	0x1B8						
	0x13C	0x1BC						
	0x140	0x1C0						
	0x144	0x1C4						

$ACTION_ROOT/hw/hw_action_nvme_memcopy.H

```
typedef struct {
        CONTROL Control;        /*  16 bytes */
        memcopy_job_t Data;     /* 108 bytes */
        uint8_t padding[SNAP_HLS_JOBSIZE - sizeof(memcopy_job_t)];
} action_reg;
```

$SNAP_ROOT/actions/include/hls_snap.H

```
typedef struct {
        snapu8_t sat; // short action type
        snapu8_t flags;
        snapu16_t seq;
        snapu32_t Retc;
        snapu64_t Reserved; // Priv_data
} CONTROL;
```

$ACTION_ROOT/include/action_memcopy.h

```
typedef struct memcopy_job {
        struct snap_addr in;    /* input data */
        struct snap_addr out;   /* output data */
} memcopy_job_t;
```

$SNAP_ROOT/software/include/snap_types.h

```
typedef struct snap_addr {
        uint64_t addr;
        uint32_t size;
        snap_addrtype_t type;           /* DRAM, NVME, ... */
        snap_addrflag_t flags;          /* SRC, DST, EXT, ... */
} snap_addr_t;
```

Performances measurements

Measurements on N250S card

hls_nvme_memcopy / N250S board	1-direction access, 1GB data going from or to SSD			
256KBytes buffer - 64 access/burst	Read from Host	Write to Host	Read from DDR4	Write to DDR4
Bytes transfered	**BW (MBps)**	**BW (MBps)**	**BW (GBps)**	**BW (GBps)**
1GB memory area transfer	**498**	**705**	**624**	**973**

Latency to access DDR4 memory:

- Read : from HLS_action request to data in HLS : 184ns
- Write : from HLS_action request to data in DDR : 105ns

Performances measurements

To run these performances, run the following :

```
snap_find_card -v -AN250S
A N250S card has been detected in card position 0
 PSL Revision is                                         : 0x3007
 Device ID    is                                         : 0x0632
 Sub device   is                                         : 0x060a
 Image loaded is self defined as                         : user
 Next image to be loaded at next reset (load_image_on_perst) is : user
snap_maint -vv
[main] Enter
[snap_version] Enter
SNAP on N250S Card, NVME enabled, 4096 MB DRAM available.
SNAP FPGA Release: v1.3.5 Distance: 43 GIT: 0xe7036da5
SNAP FPGA Build (Y/M/D): 2018/03/21 Time (H:M): 17:04
SNAP FPGA CIR Master: 1 My ID: 0
SNAP FPGA Up Time: 226 sec
[snap_version] Exit
[snap_m_init] Enter
SNAP FPGA Exploration already done (MSAT: 1 MAID: 1)

   Short |  Action Type |   Level    |
   ------+--------------+-----------+-----------
     0     0x10141007     0x00000001  IBM HLS NVMe memcopy

[snap_m_init] Exit rc: 0
[main] Exit rc: 0
```

Performances measurements

To run these performances, run the following:

```
 snap_nvme_memcopy -A HOST_DRAM -D NVME_SSD  -i in1G.bin  -d 0x0
reading input data 1073741824 bytes from in1G.bin
PARAMETERS:
  input:        in1G.bin
  output:       unknown
  type_in:      0 HOST_DRAM
  addr_in:      00003fff73b70000
  type_out:     2 NVME_SSD
  addr_out:     0000000000000000
  drive_id:     0
  size_in/out:  40000000
  mode:         00000000
  prepare nvme_memcopy job of 40 bytes size
  This is the register information exchanged between host and fpga
 00000000: 00 00 b7 73 ff 3f 00 00 00 00 00 40 00 00 12 00 | ...s............
 00000010: 00 00 00 00 00 00 00 00 00 00 00 40 02 00 23 00 | ................
 00000020: 00 00 00 00 00 00 00 00                         | ........

      get starting time
Action is running ....      got end of exec. time
SUCCESS
memcopy of 1073741824 bytes took 2157638 usec @ 497.647 MiB/sec
This represents the register transfer time + memcopy action time
```

Performances measurements

To run these performances, run the following:

```
 snap_nvme_memcopy -A NVME_SSD -D HOST_DRAM -a 0xE000 -o out1G.bin -s 0x40000000
PARAMETERS:
  input:        unknown
  output:       out1G.bin
  type_in:      2 NVME_SSD
  addr_in:      000000000000e000
  type_out:     0 HOST_DRAM
  addr_out:     00003fff58120000
  drive_id:     0
  size_in/out: 40000000
  mode:         00000000
  prepare nvme_memcopy job of 40 bytes size
  This is the register information exchanged between host and fpga
 00000000: 00 e0 00 00 00 00 00 00 00 00 00 40 02 00 12 00 | ................
 00000010: 00 00 12 58 ff 3f 00 00 00 00 00 40 00 00 23 00 | ...X............
 00000020: 00 00 00 00 00 00 00 00                         | ........

      get starting time
Action is running ....      got end of exec. time
writing output data 0x3fff58120000 1073741824 bytes to out1G.bin
SUCCESS
memcopy of 1073741824 bytes took 1522240 usec @ 705.370 MiB/sec
This represents the register transfer time + memcopy action time
```

Performances measurements

To run these performances, run the following:

```
 snap_nvme_memcopy -A CARD_DRAM -D NVME_SSD  -a 0x000 -d 0x000 -s 0x40000000
PARAMETERS:
  input:        unknown
  output:       unknown
  type_in:      1 CARD_DRAM
  addr_in:      0000000000000000
  type_out:     2 NVME_SSD
  addr_out:     0000000000000000
  drive_id:     0
  size_in/out: 40000000
  mode:         00000000
  prepare nvme_memcopy job of 40 bytes size
  This is the register information exchanged between host and fpga
 00000000: 00 00 00 00 00 00 00 00 00 00 00 40 01 00 12 00 | ................
 00000010: 00 00 00 00 00 00 00 00 00 00 00 40 02 00 23 00 | ................
 00000020: 00 00 00 00 00 00 00 00                         | ........

      get starting time
Action is running ....      got end of exec. time
SUCCESS
memcopy of 1073741824 bytes took 1721294 usec @ 623.799 MiB/sec
This represents the register transfer time + memcopy action time
```

Performances measurements

To run these performances, run the following:

```
snap_nvme_memcopy -A NVME_SSD -D CARD_DRAM -a 0x0 -d 0x0 -s 0x40000000
PARAMETERS:
  input:        unknown
  output:       unknown
  type_in:      2 NVME_SSD
  addr_in:      0000000000000000
  type_out:     1 CARD_DRAM
  addr_out:     0000000000000000
  drive_id:     0
  size_in/out: 40000000
  mode:         00000000
  prepare nvme_memcopy job of 40 bytes size
  This is the register information exchanged between host and fpga
 00000000: 00 00 00 00 00 00 00 00 00 00 00 40 02 00 12 00 | ................
 00000010: 00 00 00 00 00 00 00 00 00 00 00 40 01 00 23 00 | ................
 00000020: 00 00 00 00 00 00 00 00                         | ........

      get starting time
Action is running ....      got end of exec. time
SUCCESS
memcopy of 1073741824 bytes took 1104054 usec @ 972.545 MiB/sec
This represents the register transfer time + memcopy action time
```

Path of improvements

1. HLS memcpy function waits for the end of the request before starting a new one. Being able to parallelize reads with writes since both ports are independent would increase performance since the DMA is able to pipeline requests.

History of this document and of the action release level

V1.0: initial document