CAPI SNAP Education Series:
User Guide

CAPI SNAP Education
hls_memcopy : howto?
V2.3

Architecture of the SNAP git files

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

SNAP

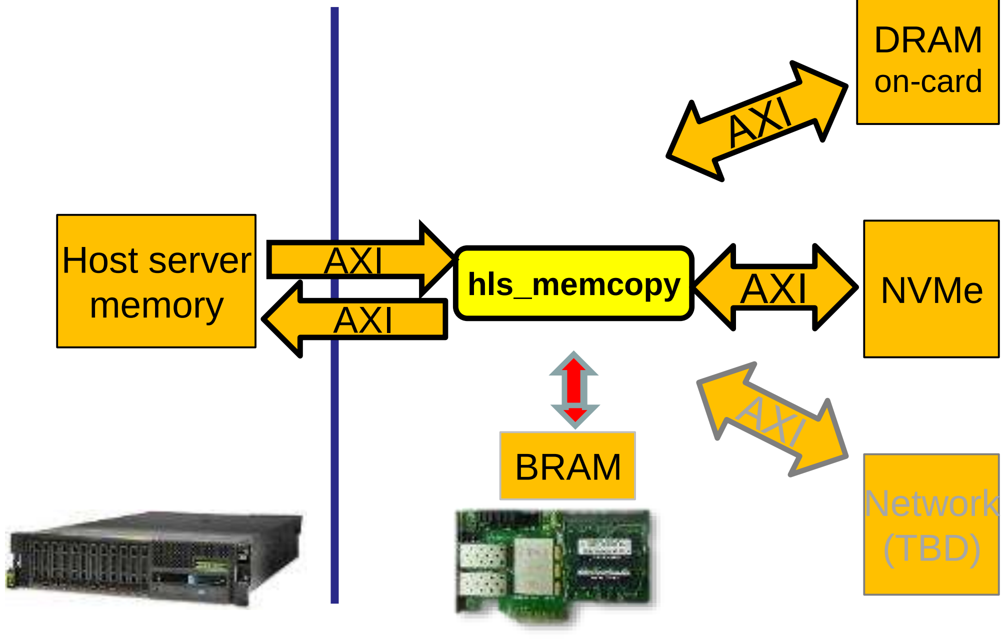
actions
- hls_memcopy
  - **sw**
    - **snap_memcopy.c** ← Main ***application*** program
    - **sw_action_memcopy.c** ← "CPU executed" ***action*** program
  - **hw**
    - **hw_action_memcopy.cpp** ← "FPGA executed" ***action*** program
    - **hw_action_memcopy.H** ← *Specific constants + "data 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 ← *Regression test used to show how to run this hls_example*
  - doc

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 ← *programs used to "discover" available actions, do basic settings and attach / detach actions*
  - snap_find_card

Action overview

Purpose: Transferring data between different resources :
- host memory,
- DDR,
- NVMe (soon)



CAPI SNAP Enabled Card

When to use it:
- Understand Basic access to different interfaces
- Memcopy benchmarking

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)

Action usage (1/2)

Usage:

```
./snap_memcopy [-h] [-v, --verbose] [-V, --version]
    -C, --card <cardno> can be (0...3)
    -i, --input <file.bin>    input file.
    -o, --output <file.bin>   output file.
    -A, --type-in <CARD_DRAM, HOST_DRAM, ...>.
    -a, --addr-in <addr>      address e.g. in CARD_RAM.
    -D, --type-out <CARD_DRAM, HOST_DRAM, ...>.
    -d, --addr-out <addr>     address e.g. in CARD_RAM.
    -s, --size <size>         size of data.
    -t, --timeout             Timeout in sec to wait for done. (10 sec default)
    -X, --verify              verify result if possible (only CARD_DRAM)
    -N, --no irq              Disable IRQs
```

Example :

```
export SNAP_TRACE=0x0
snap_maint -vv

echo move 4kB from Host to DDR@0x0 and back from DDR@0x0 to Host
rm t2; dd if=/dev/urandom of=t1 bs=1K count=4
SNAP_CONFIG=FPGA snap_memcopy -i t1 -D CARD_DRAM -d 0x0
SNAP_CONFIG=FPGA snap_memcopy -o t2 -A CARD_DRAM -a 0x0 -s0x1000

diff t1 t2
 if diff t1 t2 >/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

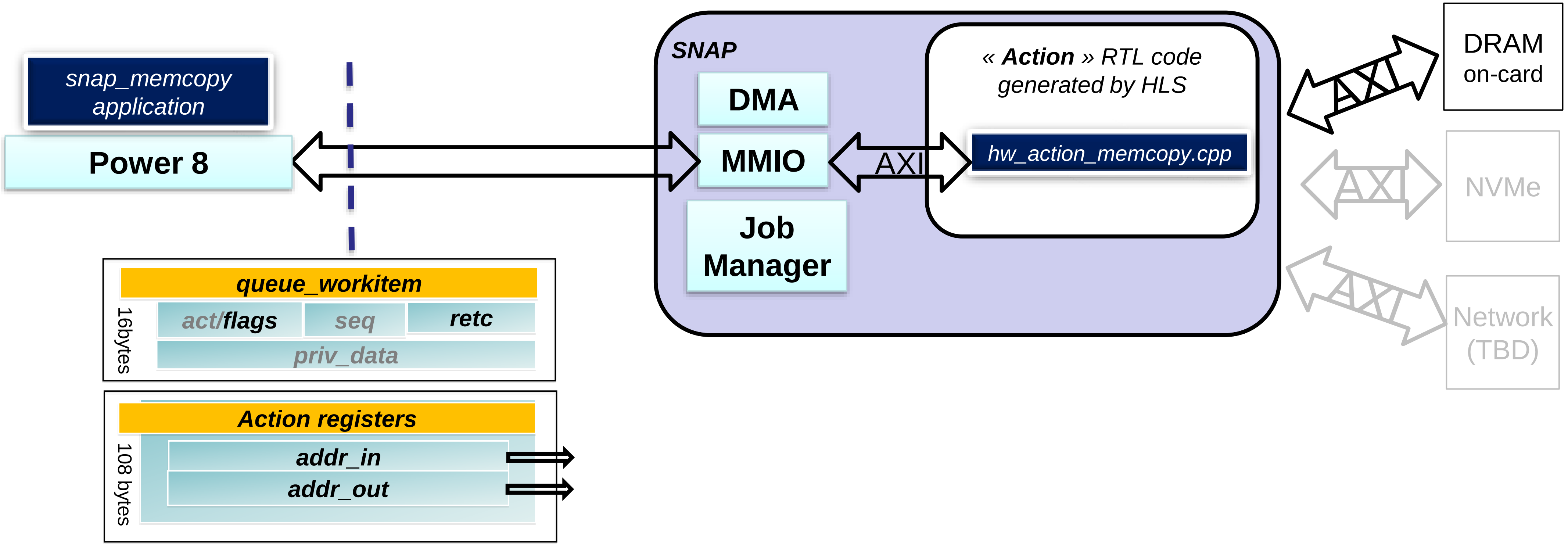
Action usage (2/2)

Different cases that can be run

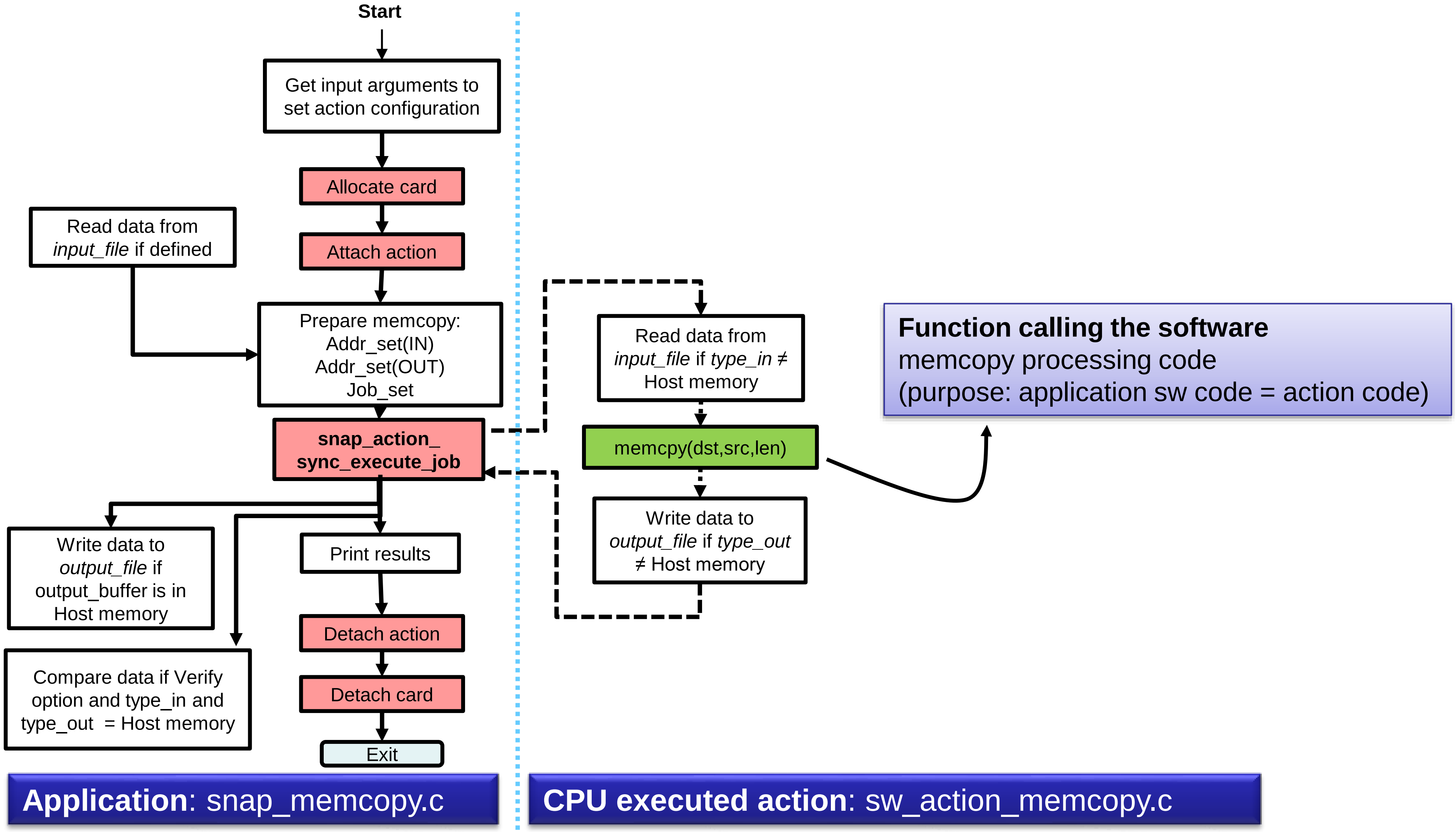
```
snap_maint -vv -C0

echo create a 512MB file with random data …wait…
rm t2; dd if=/dev/urandom of=t1 bs=1M count=512

echo READ 512MB from Host - one direction
snap_memcopy -C0 -i t1
echo WRITE 512MB to Host - one direction - (t1!=t2 since buffer is 256KB)
snap_memcopy -C0 -o t2 -s0x2000_0000

echo READ 512MB from DDR - one direction
snap_memcopy -C0 -s0x2000_0000 -ACARD_DRAM -a0x0
echo WRITE 512MB to DDR - one direction
snap_memcopy -C0 -s0x2000_0000 -DCARD_DRAM -d0x0

Move 4KB from Host to DDR and back to Host and compare
rm t2; dd if=/dev/urandom of=t1 bs=1K count=4
snap_memcopy -i t1 -D CARD_DRAM -d 0x0
snap_memcopy -o t2 -A CARD_DRAM -a 0x0 -s0x1000
diff t1 t2

echo same test using polling instead of IRQ waiting for the result
snap_memcopy -o t2 -A CARD_DRAM -a 0x0 -s0x1000 -N
```

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 !

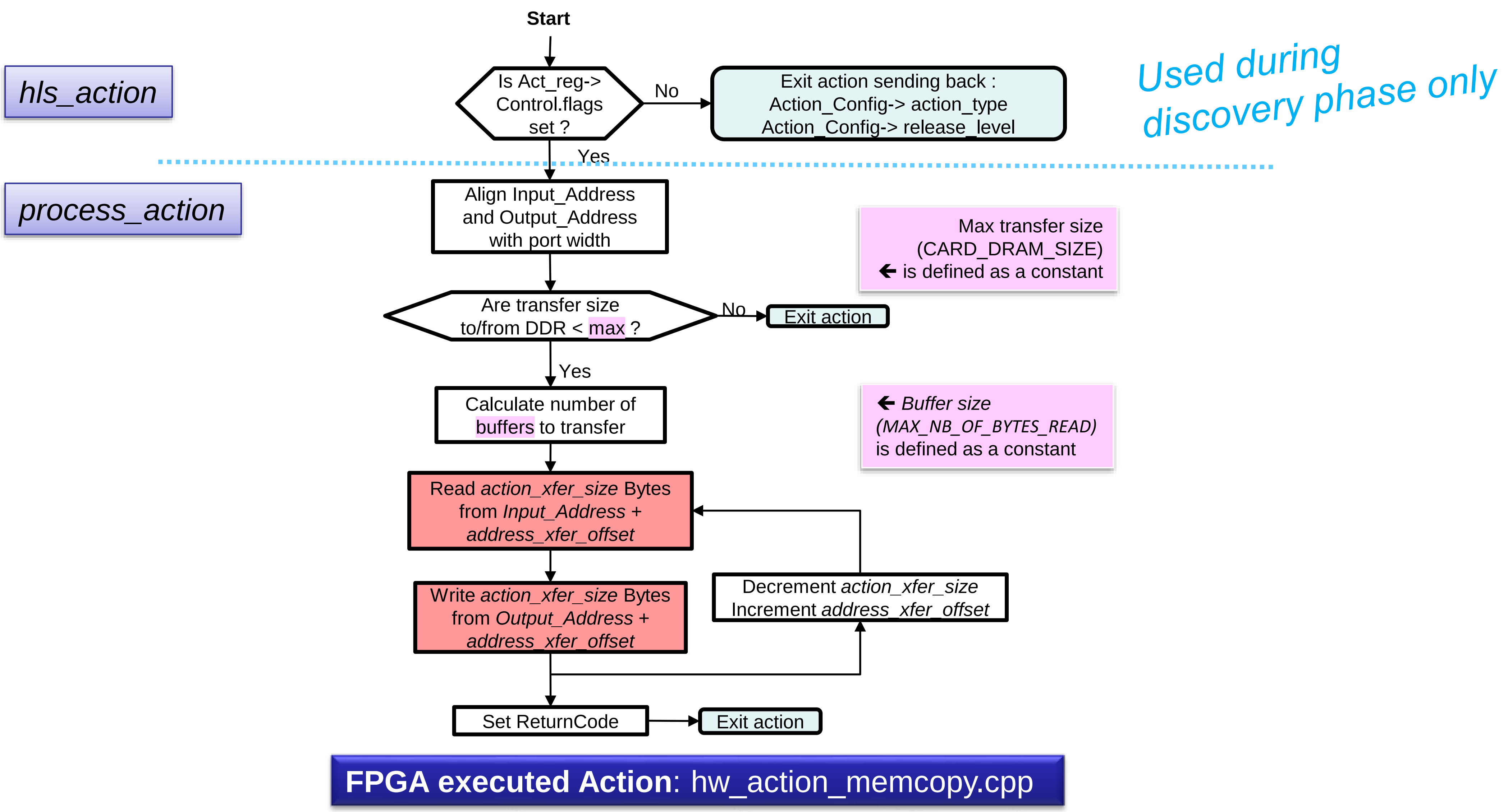
memcopy registers



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



Hardware action Code : what's in it?



Constants - Ports

Constants: ➔ $ACTION_ROOT = snap/actions/hls_memcopy

Constant name	Value	Type	Definition location	Usage
MEMCOPY_ACTION_TYPE	0x10141000	Fixed	$ACTION_ROOT/include/action_memcopy.h	memcopy ID - list is in snap/ActionTypes.md
RELEASE_LEVEL	0x00000023	Variable	$ACTION_ROOT/hw/hw_action_memcopy.**H**	release level – user defined
MAX_NB_OF_BYTES_READ	(256 * 1024)	Variable	$ACTION_ROOT/hw/hw_action_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_memcopy.**H**	Max size in 64B words of the buffer for read/write access
CARD_DRAM_SIZE	(1 * 1024 *1024 * 1024)	Variable	$ACTION_ROOT/hw/hw_action_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	*No (soon)*

MMIO Registers

Read and Write are considered from the application / software side

act_reg.Control ***CONTROL***		***This header is initialized by the SNAP job manager. The action will update the Return code and read the flags value. If the flags value is 0, then action sends only the action_RO_config_reg value and exit the action, otherwise it will process the action***						
Simu - WR	Write@	Read@	3	2	1	0	Typical Write value	Typical Read value
0x3C40	0x100	0x180	sequence		**flags**	short action type	f001_01_00	
0x3C41	0x104	0x184	**Retc** (return code 0x102/0x104)				0	0x102 - 0x104 SUCCESS/FAILURE
0x3C42	0x108	0x188	Private Data				c0febabe	
0x3C43	0x10C	0x18C	Private Data				deadbeef	

action_reg.Data ***memcopy_job_t***		***Action specific - user defined - need to stay in 108 Bytes This is the way for application and action to exchange information through this set of registers***						
Simu - WR	Write@	Read@	3	2	1	0	Typical Write value	Typical Read value
0x3C44	0x110	0x190	**in.addr** (LSB)					
0x3C45	0x114	0x194	**in.addr** (MSB)					
0x3C46	0x118	0x198	in.**size**					
0x3C47	0x11C	0x19C	in.**flags** (SRC, DST, …)		in.**type** (HOST, DRAM, NVME,..)			
0x3C48	0x120	0x1A0	**out.addr** (LSB)					
0x3C49	0x124	0x1A4	**out.addr** (MSB)					
0x3C4A	0x128	0x1A8	out.**size**					
0x3C4B	0x12C	0x1AC	out.**flags** (SRC, DST, …)		out.**type** (HOST, DRAM, NVME,..)			
	0x130	0x1B0						

$ACTION_ROOT/hw/hw_action_memcopy.H

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

$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/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;
```

$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 cards (512MB memory area transfer)

hls_memcopy		1-direction access			
256KBytes buffer - 64 access/burst		Read from Host	Write to Host	Read from DDR	Write to DDR
	Bytes transferred	BW (GBps)	BW (GBps)	BW (GBps)	BW (GBps)
CAPI1.0:	*POWER8 + ADKU3*	**3.337**	**3.305**	*DDR3:* **10.336**	*DDR3:* **9.584**
CAPI1.0:	*POWER8 + N250S*	**3.166**	**3.569**	*DDR4:* **14.854**	*DDR4:* **13.524**
CAPI2.0 *:*	*POWER9 + FX609*	**12.321**	**14.201**	*DDR4:* **14.926**	*DDR4:* **14.820**
CAPI2.0*:*	*POWER9 + RCXVUP*	**12.288**	**13.120**	*DDR4:* **13.520**	*DDR4:* **13.487**

Latency to access DDR3 memory:
- Read : from HLS_action request to data in HLS : 232ns
- Write : from HLS_action request to data in DDR : 226ns

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

To run these performances, run the following:

```
    from snap installation directory, get the card slot number
    snap_find_card -A <name_of_the_card>
    0 or 1
    ./actions/hls_memcopy/tests/test_0x10141000_throughput.sh -C1 -dLONG
or
    snap_maint -vvv -C1
    echo create a 512MB file …wait…
    dd if=/dev/urandom of=t1 bs=1M count=512
    echo READ 512MB from Host
    snap_memcopy -C1 -i t1
    echo WRITE 512MB to Host
    snap_memcopy -C1 -o t2 -s0x2000_0000
    echo READ 512MB from DDR
    snap_memcopy -C1 -s0x2000_0000 -ACARD_DRAM -a0x0
    echo WRITE 512MB to DDR
    snap_memcopy -C1 -s0x2000_0000 -DCARD_DRAM -d0x0
```

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

V2.0: initial document
V2.1: new files directory structure applied
V2.2: changes to have one direction access to get real performances
V2.3: simplification of paths thanks to new SNAP features - updates in documentation – Issue#320 circumvention removed