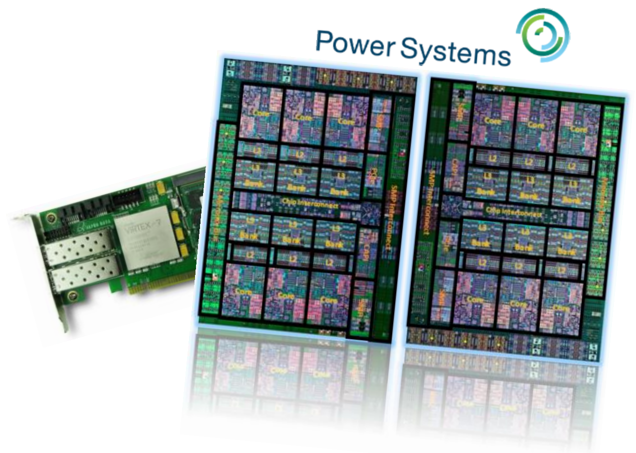
CAPI SNAP Education Series:
User Guide

CAPI SNAP Education
hls_memcopy : howto?
V2.2

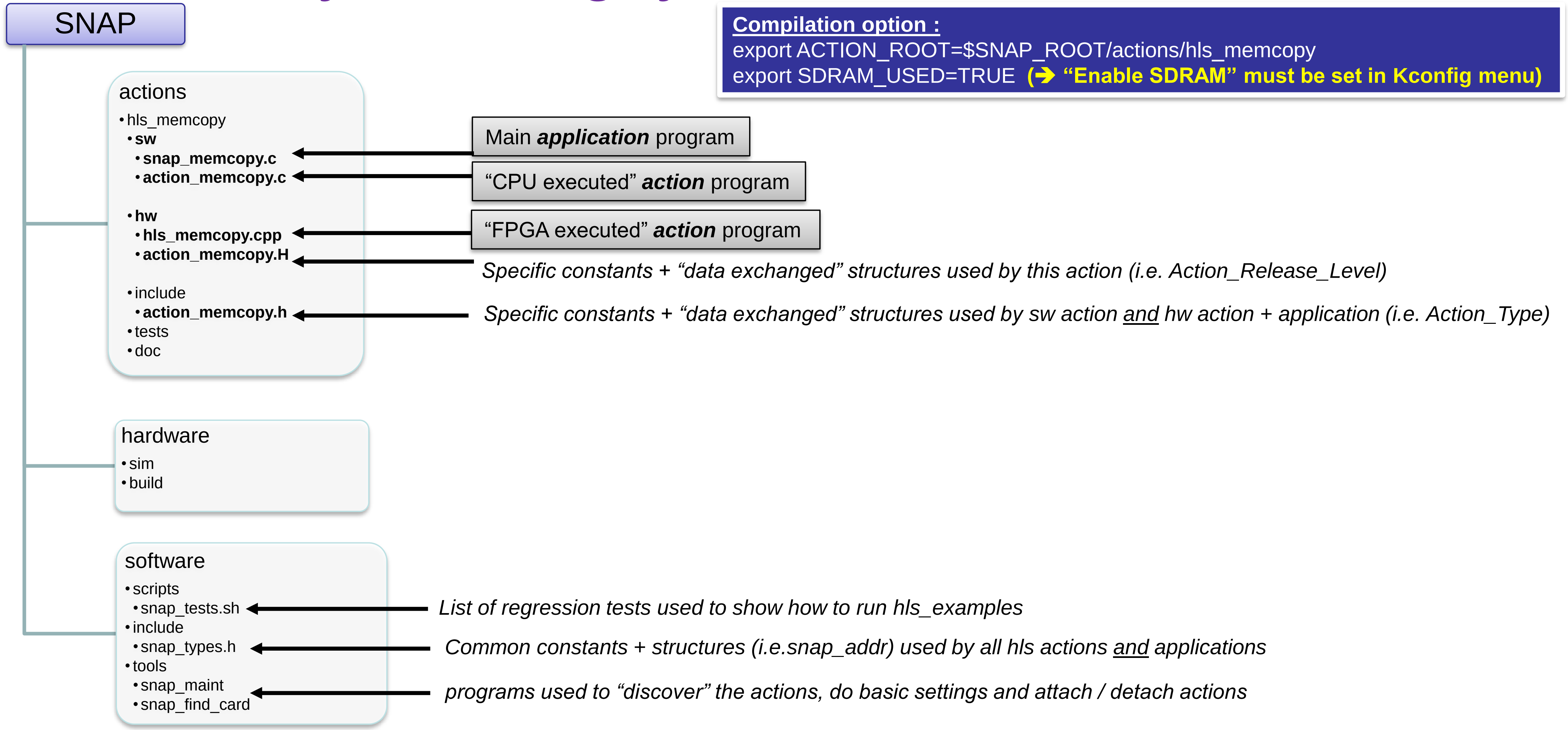


September 20th, 2017

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
    - **action_memcopy.c** ← "CPU executed" ***action*** program
  - **hw**
    - **hls_memcopy.cpp** ← "FPGA executed" ***action*** program
    - **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
  - doc

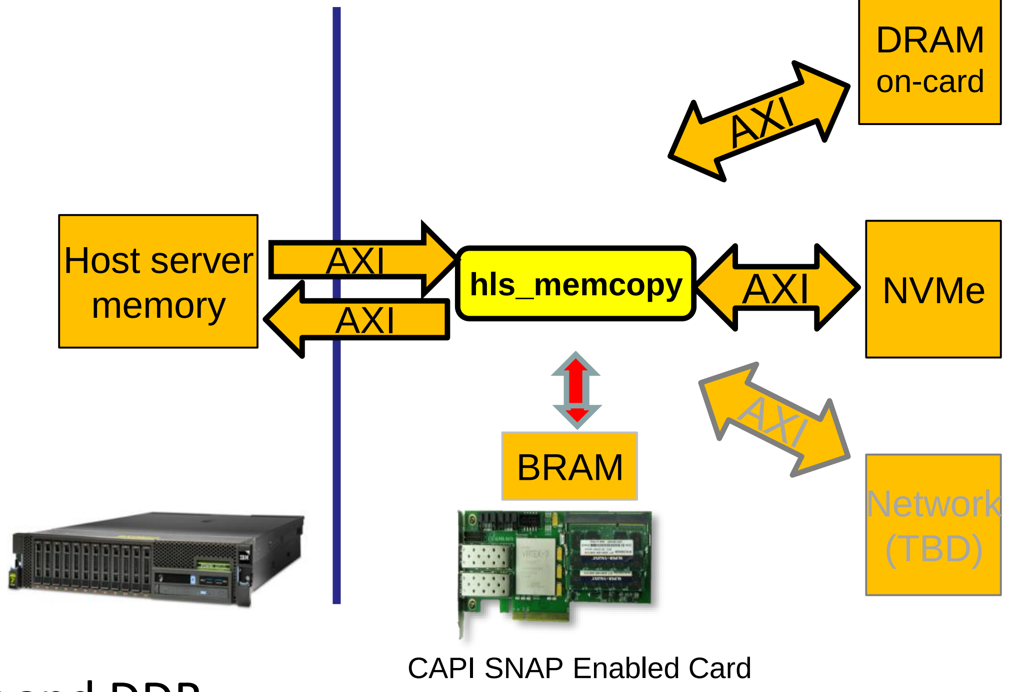
hardware
- sim
- build

software
- scripts
  - snap_tests.sh ← *List of regression tests used to show how to run hls_examples*
- 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" the actions, do basic settings and attach / detach actions*

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_ROOT/software/tools/snap_maint -vvv

echo move 512kB from Host to DDR@0x0 and back from DDR@0x0 to Host
rm t2; dd if=/dev/urandom of=t1 bs=1K count=512
SNAP_CONFIG=0x0 $ACTION_ROOT/sw/snap_memcopy -i t1 -D CARD_DRAM -d 0x0
SNAP_CONFIG=0x0 $ACTION_ROOT/sw/snap_memcopy -o t2 -A CARD_DRAM -a 0x0 -s0x00001000

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

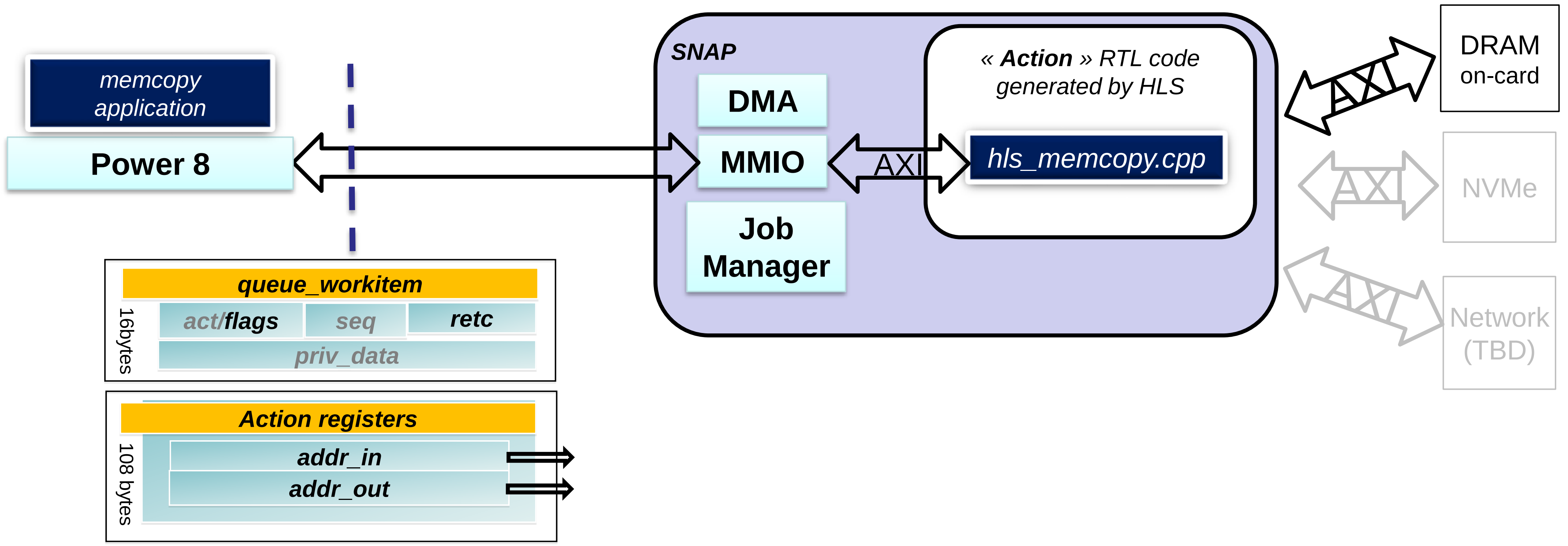
SNAP_CONFIG = 0x0 ➔ hardware execution
SNAP_CONFIG = 0x1 ➔ software execution

Action usage (2/2)

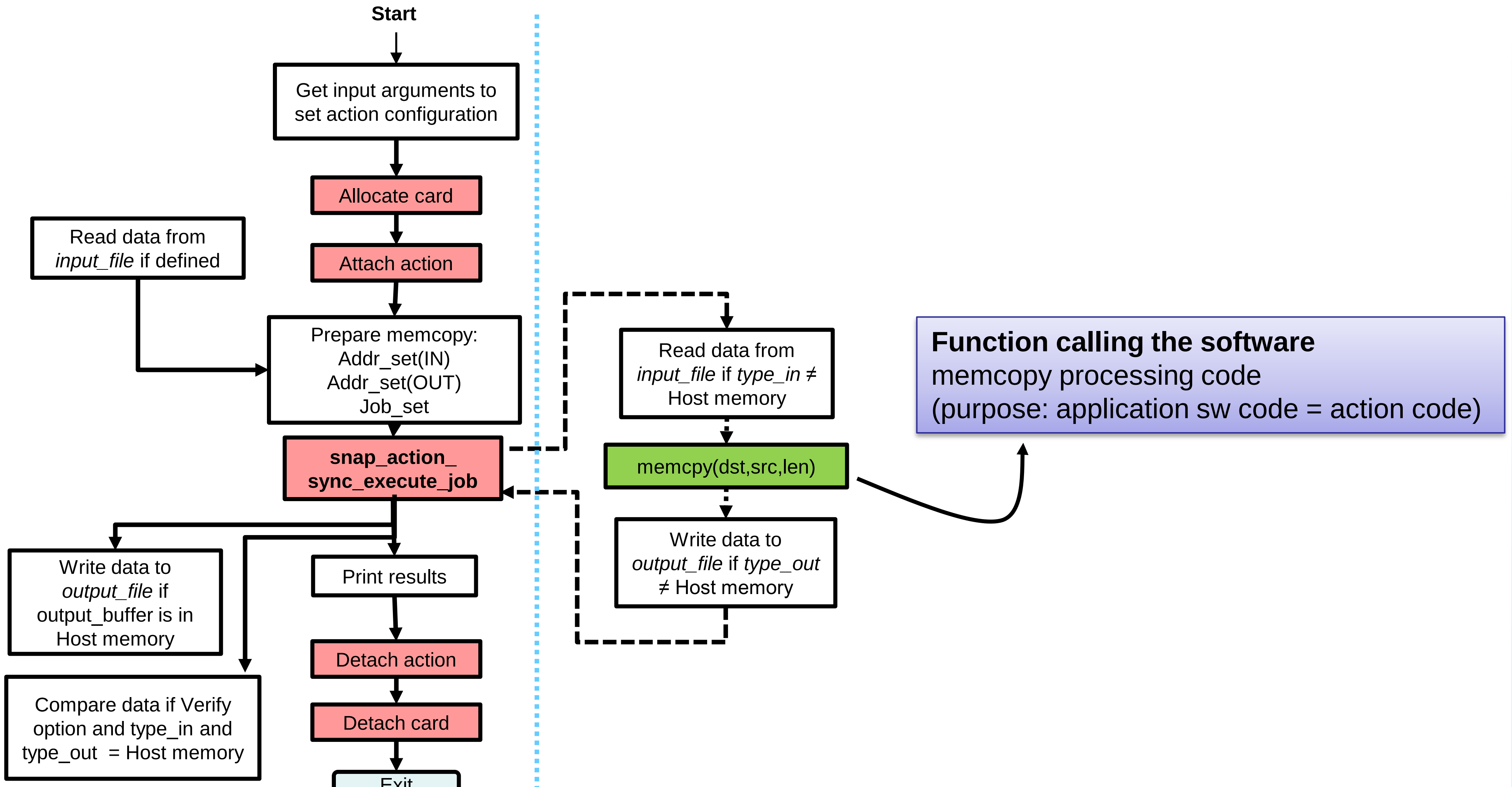
Different cases that can be run

```
$SNAP_ROOT/software/tools/snap_maint -vvv -C0

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

echo READ 512MB from Host - one direction
$ACTION_ROOT/sw/snap_memcopy -C0 -i t1

echo WRITE 512MB to Host - one direction
$ACTION_ROOT/sw/snap_memcopy -C0 -o t2 -s0x20000000

echo READ 512MB from DDR - one direction
$ACTION_ROOT/sw/snap_memcopy -C0 -s0x20000000 -ACARD_DRAM -a0x0

echo WRITE 512MB to DDR - one direction
$ACTION_ROOT/sw/snap_memcopy -C0 -s0x20000000 -DCARD_DRAM -d0x0

echo READ file t1 from host memory THEN write it at @0x0 in card
SNAP_CONFIG=0x0 $ACTION_ROOT/sw/snap_memcopy -i t1 -D CARD_DRAM -d 0x0

echo READ 4KB from card DDR at @0x0 THEN write them to Host and file t2
SNAP_CONFIG=0x0 $ACTION_ROOT/sw/snap_memcopy -o t2 -A CARD_DRAM -a 0x0 -s0x1000
echo same test using polling instead of IRQ waiting for the result
SNAP_CONFIG=0x0 $ACTION_ROOT/sw/snap_memcopy -o t2 -A CARD_DRAM -a 0x0 -s0x1000 -N
```

memcopy registers



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



Start
Get input arguments to set action configuration
Allocate card
Attach action
Read data from *input_file* if defined
Prepare memcopy: Addr_set(IN) Addr_set(OUT) Job_set
snap_action_ sync_execute_job
Write data to *output_file* if output_buffer is in Host memory
Print results
Compare data if Verify option and type_in and type_out = Host memory
Detach action
Detach card
Exit
Read data from *input_file* if *type_in* ≠ Host memory
memcpy(dst,src,len)
Write data to *output_file* if *type_out* ≠ Host memory

Function calling the software memcopy processing code (purpose: application sw code = action code)

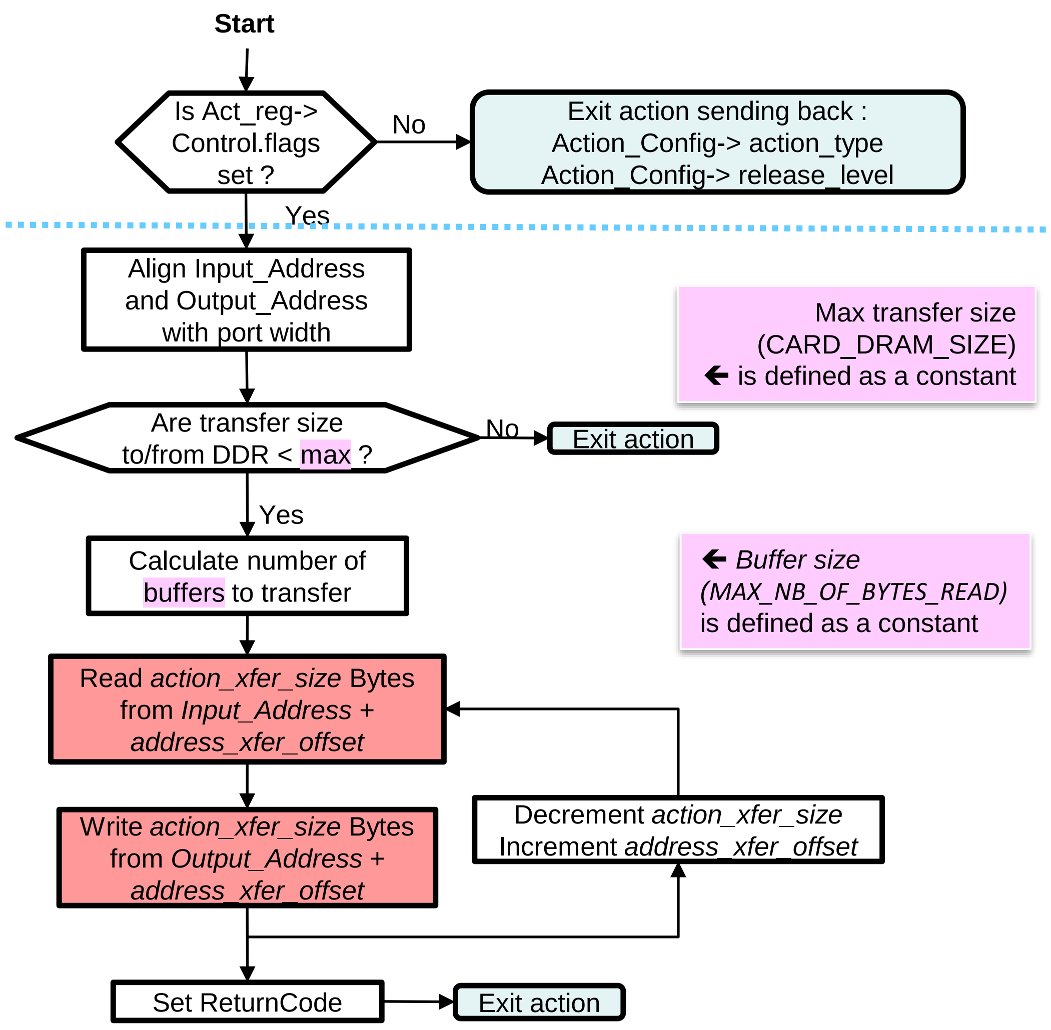
Application: snap_memcopy.c

CPU executed action: action_memcopy.c

Hardware action Code : what's in it?

hls_action

process_action



Start
Is Act_reg-> Control.flags set ?
No
Exit action sending back : Action_Config-> action_type Action_Config-> release_level
Yes
Align Input_Address and Output_Address with port width
Are transfer size to/from DDR < max ?
No
Exit action
Max transfer size (CARD_DRAM_SIZE) ← is defined as a constant
Yes
Calculate number of buffers to transfer
← Buffer size (MAX_NB_OF_BYTES_READ) is defined as a constant
Read action_xfer_size Bytes from Input_Address + address_xfer_offset
Write action_xfer_size Bytes from Output_Address + address_xfer_offset
Decrement action_xfer_size Increment address_xfer_offset
Set ReturnCode
Exit action

Used during discovery phase only

FPGA executed Action: hls_memcopy.cpp

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	0x00000022	Variable	$ACTION_ROOT/hw/action_memcopy.H	release level – user defined
MAX_NB_OF_BYTES_READ	(256 * 1024)	Variable	$ACTION_ROOT/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/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/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							
	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,..)					

$ACTION_ROOT/hw/action_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;
```

$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 KU3 card

hls_memcopy / KU3 board	1-direction access			
256KBytes buffer - 64 access/burst	Read from Host	Write to Host	Read from DDR3	Write to DDR3
Bytes transfered	BW (GBps)	BW (GBps)	BW (GBps)	BW (GBps)
512MB memory area transfer	**3.337**	**3.305**	**10.336**	**9.584**

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

To run these performances, run the following:

```
$SNAP_ROOT/software/tools/snap_find_card -A KU3
1
$SNAP_ROOT/software/tools/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
$ACTION_ROOT/sw/snap_memcopy -C1 -i t1

echo WRITE 512MB to Host
$ACTION_ROOT/sw/snap_memcopy -C1 -o t2 -s0x20000000

echo READ 512MB from DDR
$ACTION_ROOT/sw/snap_memcopy -C1 -s0x20000000 -ACARD_DRAM -a0x0

echo WRITE 512MB to DDR
$ACTION_ROOT/sw/snap_memcopy -C1 -s0x20000000 -DCARD_DRAM -d0x0
```

Performances measurements

Measurements on FGT card

hls_memcopy / FGT board	1-direction access			
256KBytes buffer - 64 access/burst	Read from Host	Write to Host	Read from DDR4	Write to DDR4
Bytes transfered	**BW (GBps)**	**BW (GBps)**	**BW (GBps)**	**BW (GBps)**
512MB memory area transfer	**3.166**	**3.569**	**14.854**	**13.524**

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:

```
$SNAP_ROOT/software/tools/snap_find_card -A FGT
0
$SNAP_ROOT/software/tools/snap_maint -vvv -C0

echo create a 512MB file …wait…
dd if=/dev/urandom of=t1 bs=1M count=512

echo READ 512MB from Host
$ACTION_ROOT/sw/snap_memcopy -C0 -i t1

echo WRITE 512MB to Host
$ACTION_ROOT/sw/snap_memcopy -C0 -o t2 -s0x20000000

echo READ 512MB from DDR
$ACTION_ROOT/sw/snap_memcopy -C0 -s0x20000000 -ACARD_DRAM -a0x0

echo WRITE 512MB to DDR
$ACTION_ROOT/sw/snap_memcopy -C0 -s0x20000000 -DCARD_DRAM -d0x0
```

Path of improvements

1. HLS 2017.1 memcpy prevents from using maximum AXI bandwidth limiting burst transfers to 1KB. A patch to Issue #320 has been provided that gives the maximum 4KB burst but requires a complex pipelined loop instead of a simple memcopy. This is announced to be corrected in Vivado HLS 2017.4
2. HLS memcpy function waits for the end of the request before starting a new one. Being able to parallelize reads with writes 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