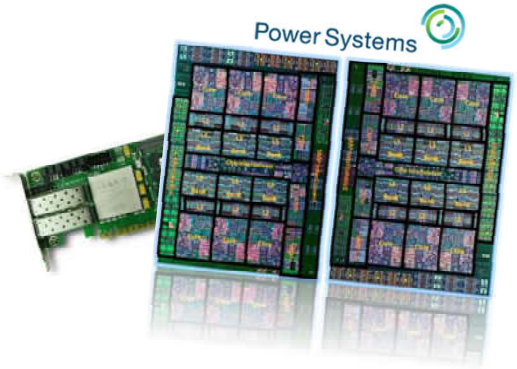
IBM Power Systems

CAPI SNAP Education Series:
User Guide

CAPI SNAP Education
hls_search : howto?
V2.2

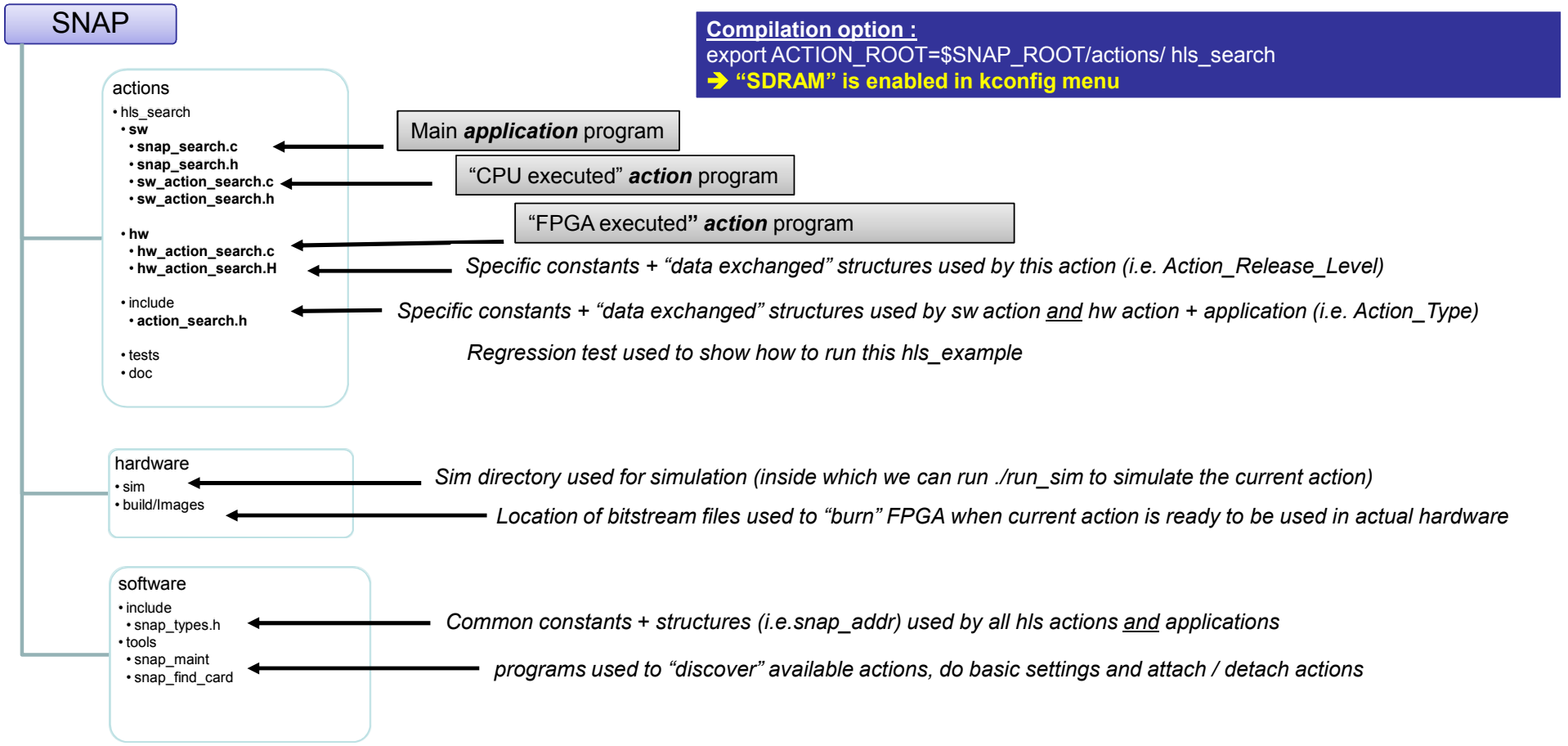


CAPI Technology

February 26th, 2018

SNAP Framework built on Power™ CAPI technology

Architecture of the SNAP git files



Action overview

Purpose: Search a Pattern in a text and count number of occurrences

- Pattern is in host memory, Text is in DDR (or NVMe soon)
- **Compare HW "data access + processing" vs SW "data access + processing" times**

When to use it:

- Understand basic access to different interfaces
- Processing data in array vs data in a streaming flow
- Understand insertion of code in the SNAP environment
- Multiple calls of action to perform different code

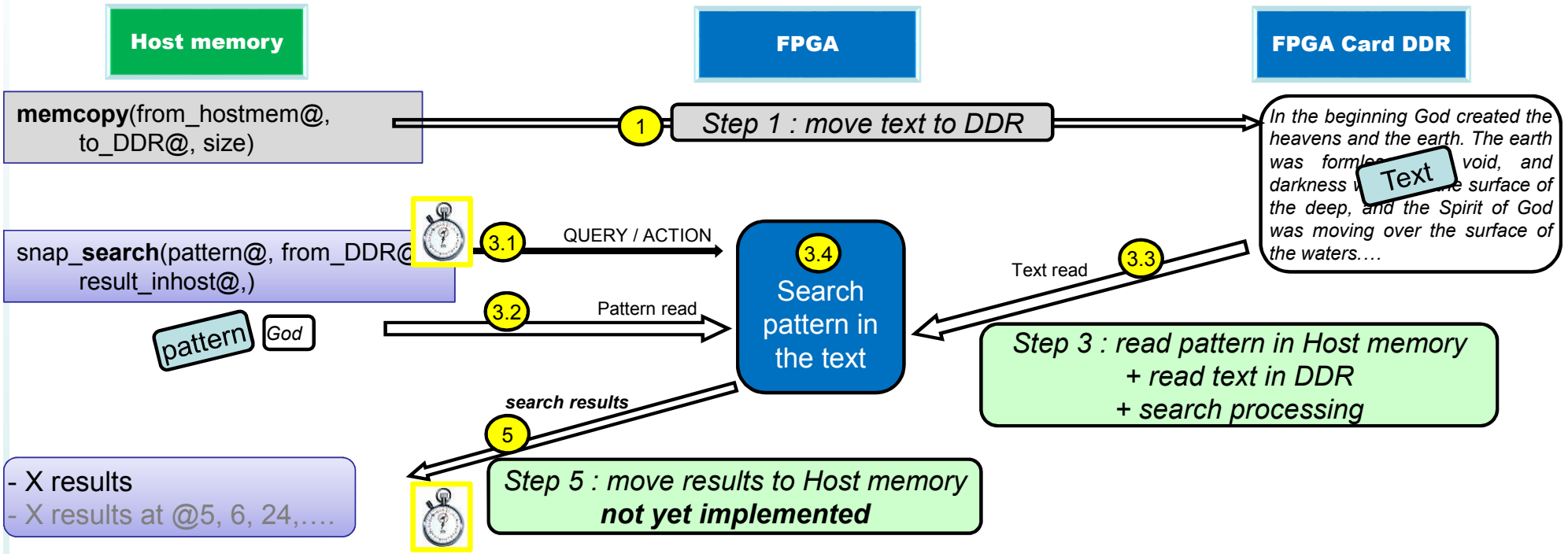
Memory management:

- Step1: (HW+SW) Action transfers Text from Host memory to DDR
- Step2: (SW only) Action transfers Text from DDR to Host memory
- Step3: (HW only) Search is done in hardware action – Result is written in register
- Step4: (SW only) Search is done in software application
- Step5: (HW only) (soon) Action reports pattern positions from DDR to Host memory

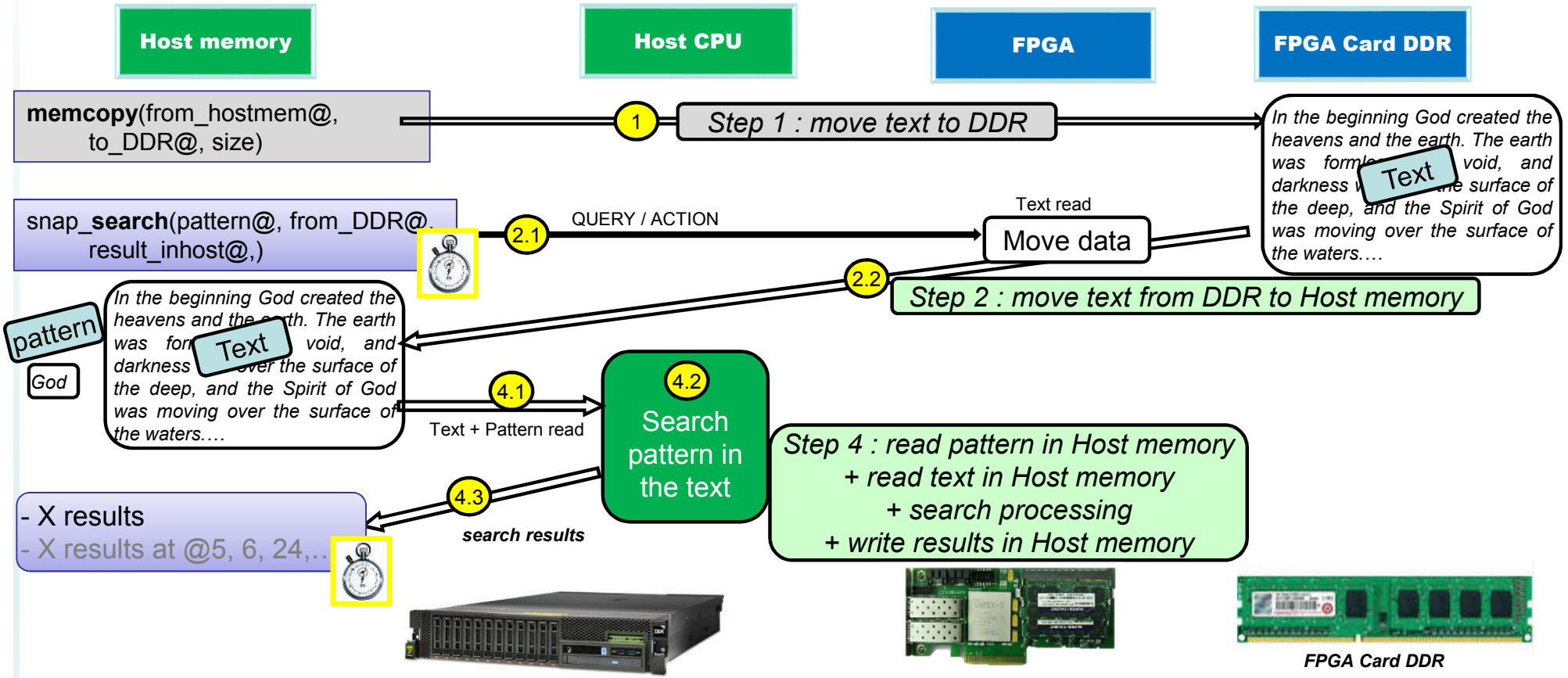
Known limitations:

- 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)

Hardware Search processing : FPGA pattern search in a text located in DDR



Software Search processing : CPU pattern search in a text located in DDR



Action usage

Usage:

```
./snap_search [-h] [-v, --verbose] [-V, --version]
          -C, --card <cardno> can be (0...3)
          -s, --software         Test the software flow (Step 2 + 4)
          -m, --method           Can be (0,1,2) different method search (1=Naïve – 2=KMP, 0=Streaming)
          -i, --input <data.bin> Input data.
          -I, --items <items>    Max items to find.
          -p, --pattern <str>    Pattern to search for
          -E, --expected <num>   Expected # of patterns to find
          -t, --timeout <num>    timeout in sec (default 10 sec)
          -N, --no irq             Disable IRQs
```

Options:

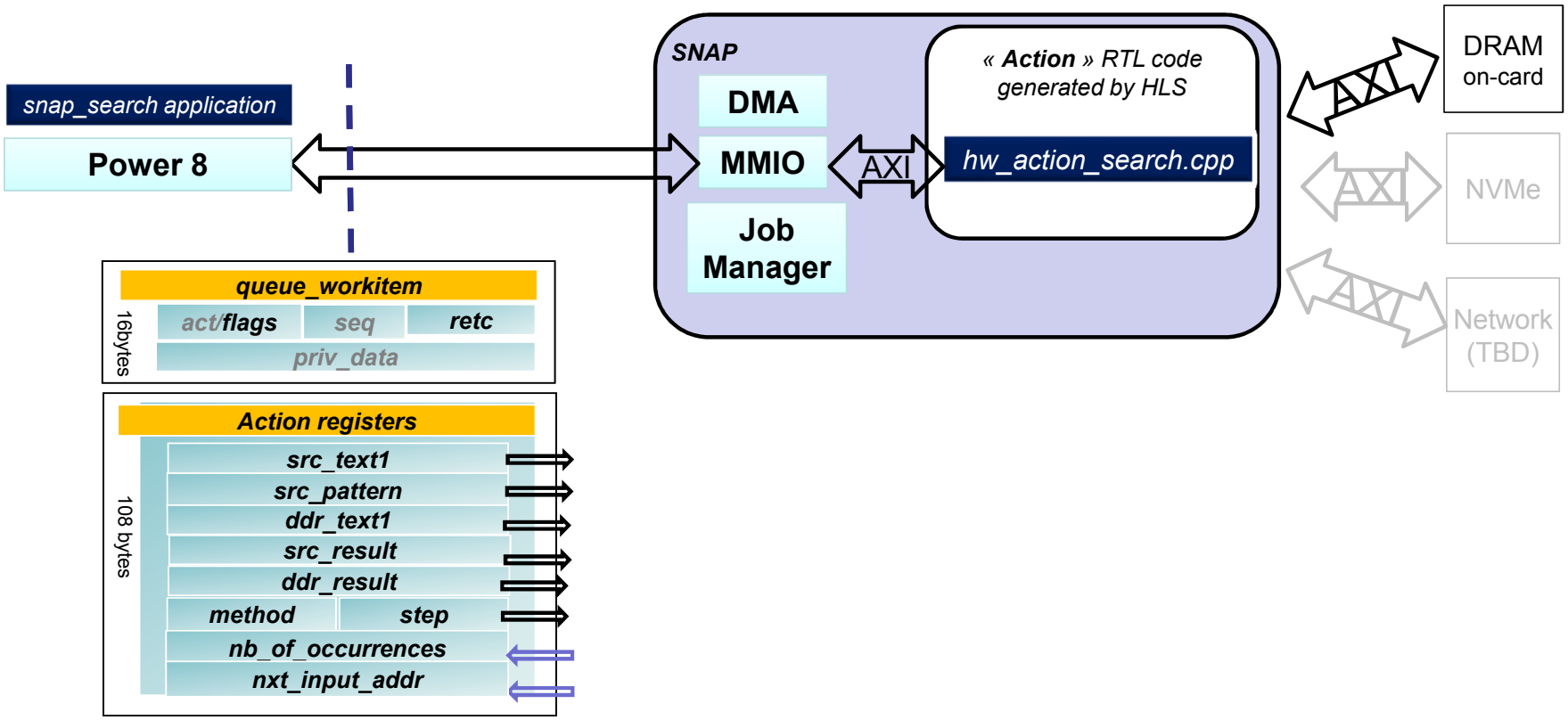
```
 SNAP_TRACE = 0x0 ➔ no debug trace
 SNAP_TRACE = 0xF ➔ full debug trace
SNAP_CONFIG = FPGA ➔ hardware execution
SNAP_CONFIG = CPU ➔ software execution
```

Example :

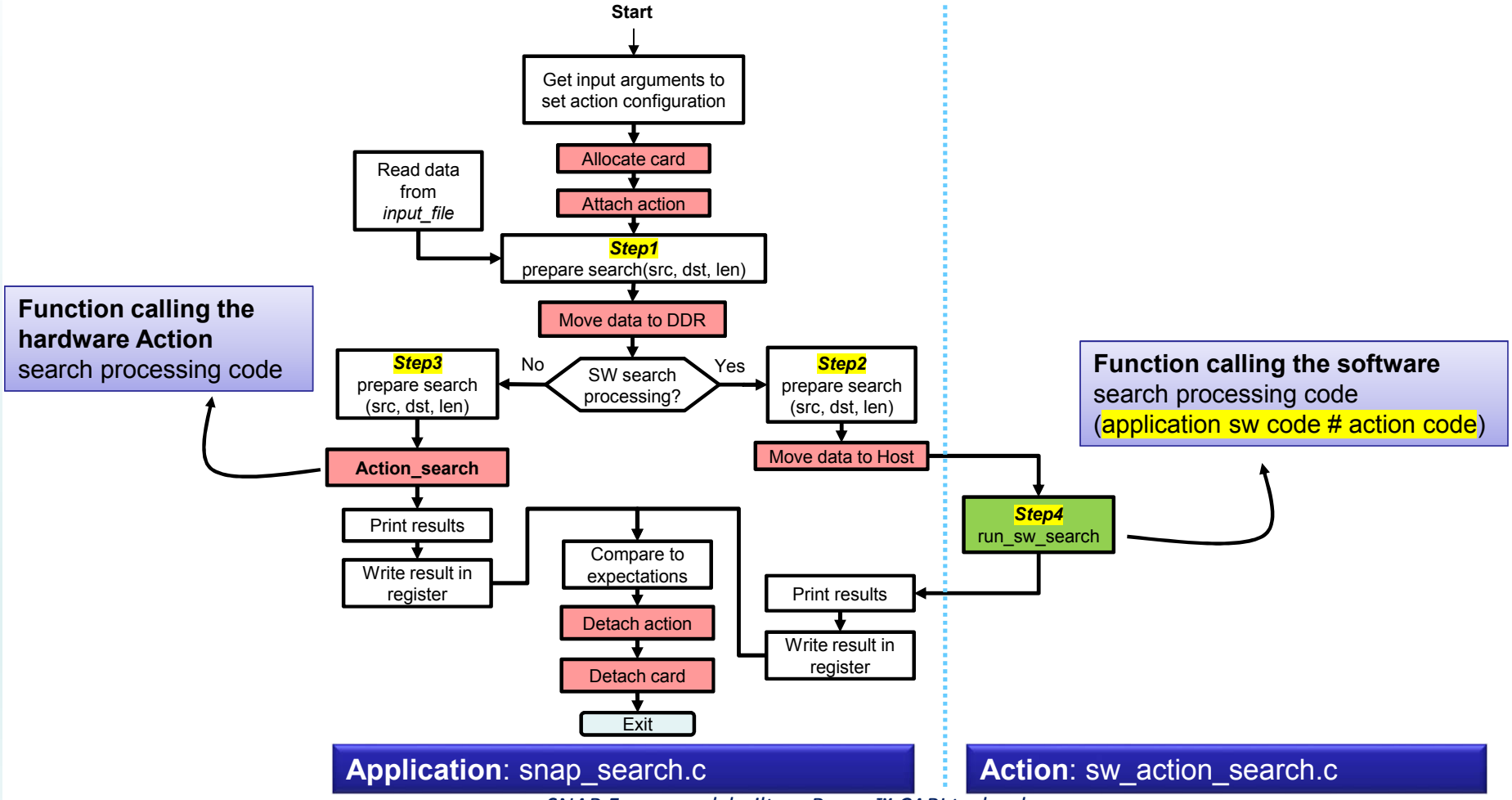
```
export SNAP_TRACE=0x0
cd $SNAP_ROOT && export ACTION_ROOT=$SNAP_ROOT/actions/hls_hashjoin
source snap_path.sh
snap_maint -vv

echo \"Hardware\" search pattern looking for the word \"include\" in ~snap/software/examples/search.txt
$SNAP_ROOT/software/examples/snap_search -t5000 -m1 -X -E88 -i ../../../../actions/hls_search/sw/snap_search.txt -p include

echo \"Software\" search pattern looking for the word \"include\" in ~snap/software/examples/search.txt
$SNAP_ROOT/software/examples/snap_search -t5000 -m1 -X -E88 -s      -i ../../../../ actions/hls_search/sw/snap_search.txt
-p include
```

Software data move + search (Step2+4)

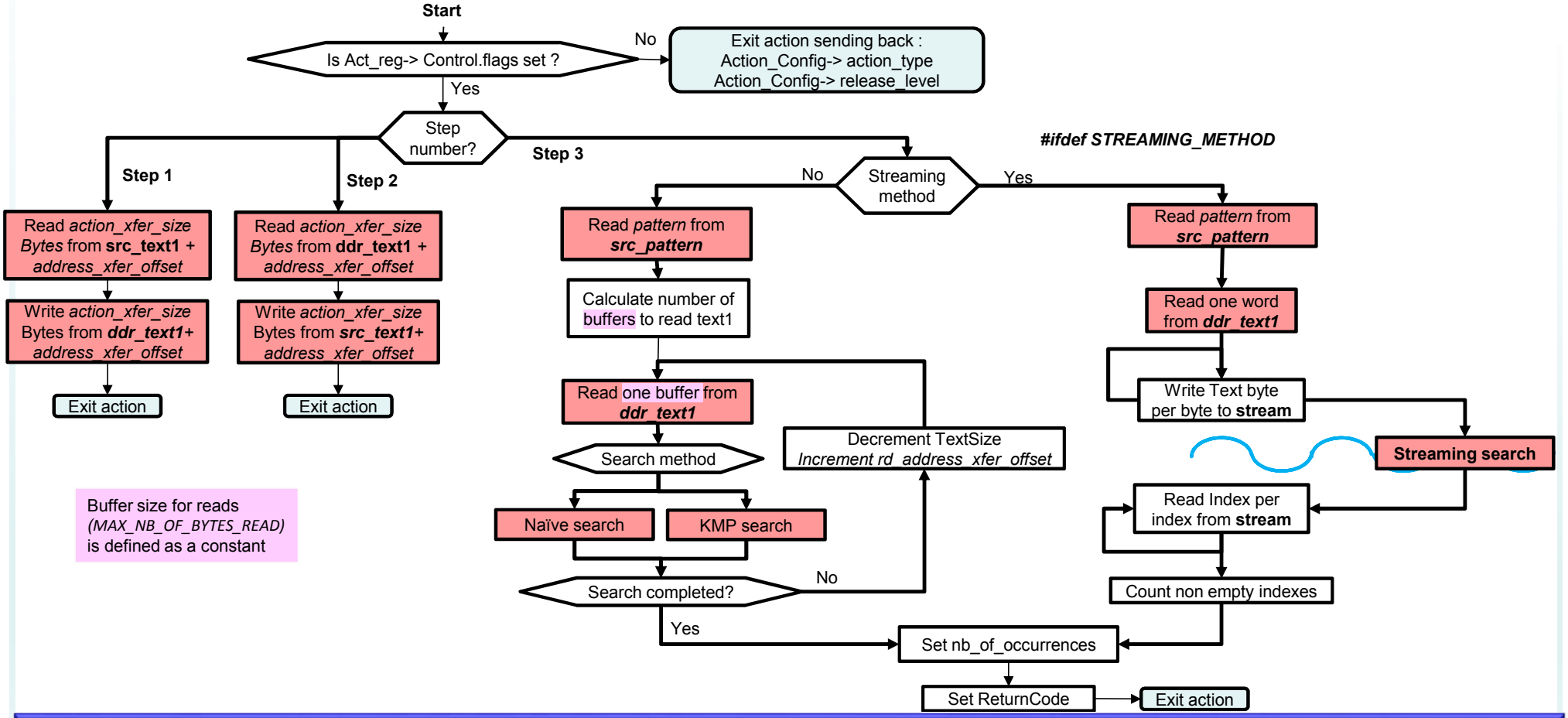
Search registers



Application Code : what's in it?



Action search Code : what's in it?



Action: hw_action_search.cpp

Constants - Ports

Constants:

Constant name	Value	Type	Definition location	Usage
SEARCH_ACTION_TYPE	0x10141003	Fixed	$ACTION_ROOT/include/action_search.h	Search ID - list is in snap/ActionTypes.md
RELEASE_LEVEL	0x00000022	Variable	$ACTION_ROOT/hw/hw_action_search.**H**	release level – user defined
MAX_NB_OF_BYTES_READ	(4 * 1024)	Variable	$ACTION_ROOT/hw/hw_action_search.**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_search.**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_search.**H**	Max size of the DDR – *unused here*
PATTERN_SIZE	BPERDW	Variable	$ACTION_ROOT/hw/hw_action_search.**H**	Max size of the pattern – Limiting to 64 Bytes so that reading pattern is done in a single word access.
TEXT_SIZE	4096 * MAX_NB_OF_BYTES_READ	Variable	$ACTION_ROOT/hw/hw_action_search.**H**	Max size of the text – used for streaming mode only

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*

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 search_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
0x3C44	0x110	0x190	[snap_addr]**src_text1**.addr (LSB)				
0x3C45	0x114	0x194	[snap_addr]**src_text1**.addr (MSB)				
0x3C46	0x118	0x198	[snap_addr]**src_text1**.size				
0x3C47	0x11C	0x19C	[snap_addr]**src_text1**.flags (SRC, DST, ...)		[snap_addr]**src_text1**.type (DRAM, NVME,..)		
0x3C48	0x120	0x1A0	[snap_addr]**src_pattern**.addr (LSB)				
0x3C49	0x124	0x1A4	[snap_addr]**src_pattern**.addr (MSB)				
0x3C4A	0x128	0x1A8	[snap_addr]**src_pattern**.size				
0x3C4B	0x12C	0x1AC	[snap_addr]**src_pattern**.flags (SRC, DST, ...)		[snap_addr]**src_pattern**.type (DRAM, NVME,..)		
0x3C4C	0x130	0x1B0	[snap_addr]**ddr_text1**.addr (LSB)				
0x3C4D	0x134	0x1B4	[snap_addr]**ddr_text1**.addr (MSB)				
0x3C4E	0x138	0x1B8	[snap_addr]**ddr_text1**.size				
0x3C4F	0x13C	0x1BC	[snap_addr]**ddr_text1**.flags (SRC, DST, ...)		[snap_addr]**ddr_text1**.type (DRAM, NVME,..)		
0x3C50	0x140	0x1C0	[snap_addr]**src_result**.addr (LSB)				
0x3C51	0x144	0x1C4	[snap_addr]**src_result**.addr (MSB)				
0x3C52	0x148	0x1C8	[snap_addr]**src_result**.size				
0x3C53	0x14C	0x1CC	[snap_addr]**src_result**.flags (SRC, DST, ...)		[snap_addr]**src_result**.type (DRAM, NVME,..)		
0x3C54	0x150	0x1D0	[snap_addr]**ddr_result**.addr (LSB)				
0x3C55	0x154	0x1D4	[snap_addr]**ddr_result**.addr (MSB)				
0x3C56	0x158	0x1D8	[snap_addr]**ddr_result**.size				
0x3C57	0x15C	0x1DC	[snap_addr]**ddr_result**.flags (SRC, DST, ...)		[snap_addr]**ddr_result**.type (DRAM, NVME		
0x3C58	0x160	0x1E0	**method**		**step**		
0x3C59	0x164	0x1E4	**nb_of_occurrences**				
0x3C5A	0x168	0x1E8	**next_input_addr** (LSB)				
0x3C5B	0x16C	0x1EC	**next_input_addr** (MSB)				

$ACTION_ROOT/hw/hw_action_search.H

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

$ACTION_ROOT/include/action_search.h

```
typedef struct search_job {
    struct snap_addr src_text1;     /* input text in HOST: 128 bits*/
    struct snap_addr src_pattern;   /* input pattern in HOST: 128 bits*/
    struct snap_addr ddr_text1;     /* input text in DDR : 128 bits*/
    struct snap_addr src_result;    /* output result in HOST : 128 bits*/
    struct snap_addr ddr_result;    /* output result in DDR : 128 bits*/
    uint16_t step;
    uint16_t method;
    uint32_t nb_of_occurrences;
    uint64_t next_input_addr;
} search_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

hls_search KU3 board					Step 1	Step 3		HW total	Step 2	Step 4			SW Total	SW Total
SNAP_CONFIG = 0x0		Size	Size		Host to DDR	**HW Naive**	**HW KMP**	best HW total	DDR to Host	SW Naive processing	SW KMP processing	**SW Naive**	**SW KMP**
text	**pattern**	**Text (Bytes)**	**Pattern**	Matches	(µs)	**(µs)**	**(µs)**	(µs)	(µs)	(µs)	(µs)	**(µs)**	**(µs)**
random (4096*4k)	pr	16,777,216	2	#290	10,542	**311,092**	**578,440**	**311,092**	10,027	29,360	72,515	***39,387***	***82,542***
random (8192*4k)	pr	33,554,432	2	#520	20,641	**622,127**	**1,156,712**	**622,127**	21,245	58,403	145,071	***79,648***	***166,316***

These performances almost shows performances differences between the algorithm used.
No optimization was done in C code yet to try and get the best performances!

What else ?

Path of improvement ?

1. HLS should allow generation of 64 access per burst so that we can read 4KiB in one request
2. DMA pipelining which could increase bandwidth sending the next Host memoruy access request before having all results from the initial request.
3. Improving data types cast
4. Optimize code to get better results

History of this document and of the action release level

V2.0: initial document
V2.1: new files directory structure applied
V2.2: cleaning code