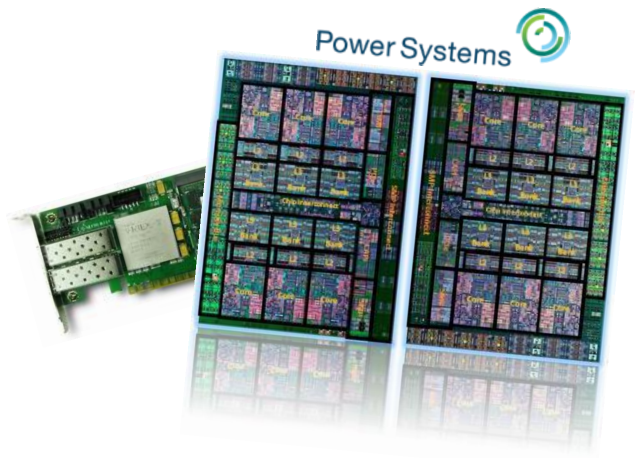
CAPI SNAP Education Series:
User Guide

CAPI SNAP Education
hls_latency_eval : howto?
V2.0



May 31th, 2018

Architecture of the SNAP git files

Compilation option :
export ACTION_ROOT=$SNAP_ROOT/actions/hls_latency_eval
export SDRAM_USED=FALSE **(➔ "Enable SDRAM" must be unset in Kconfig menu)**

SNAP

actions
- hls_latency_eval
  - **sw**
    - **snap_latency_eval.c**
    - **action_lowercase.c**
  - **hw**
    - **action_uppercase.cpp**
    - **action_uppercase.H**
  - include
    - **action_changecase.h**
  - tests
  - doc

Main ***application*** program

"CPU executed" ***action*** program

"FPGA executed" ***action*** program

Specific constants + "data exchanged" structures used by this action (i.e. Action_Release_Level)

Specific constants + "data exchanged" structures used by sw action <u>and</u> hw action + application (i.e. Action_Type)

Regression test used to show how to run this hls_example

hardware
- sim
- Build/Images

Sim directory used for simulation (inside which we can run ./run_sim to simulate the current action)

Location of bitstream files used to "burn" FPGA when current action is ready to be used in actual hardware

software
- include
  - snap_types.h
- tools
  - snap_maint
  - snap_find_card

Common constants + structures (i.e.snap_addr) used by all hls actions <u>and</u> applications

programs used to "discover" the actions, do basic settings and attach / detach actions

Action overview

Purpose: Provide to SNAP user a simple example to let him optimize the data exchanges between an application and an action with a minimum of latency.
Access to external interfaces are :

- Host memory server

When to use it:

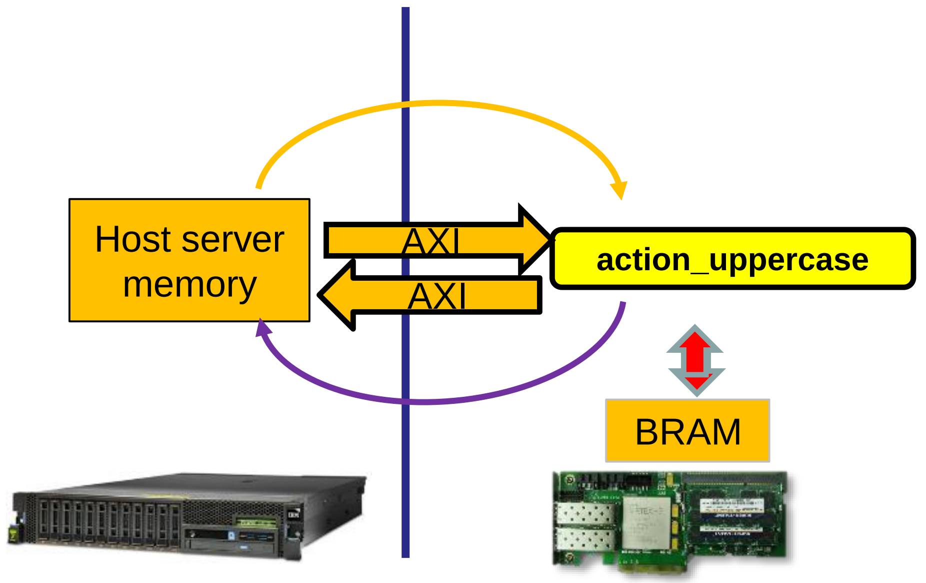
- Understand how to optimize latencies access
- Measure latency from application to application

Memory management:

- Application is managing address of Host memory
- Data are read 64B words one after the other

Known limitations:

- HLS requires transfers to be 64 byte aligned and a size of multiples of 64 bytes



CAPI SNAP Enabled Card

Action usage

Usage:

```
./snap_latency_eval [-h] [-v, --verbose] [-V, --version]
  -C, --card <cardno> can be (0...3)
  -t, --timeout             timeout in sec to wait for done.
  -T, --Action timeout      Number max of reads done by the action * 0xF.
  -n, --Number of iterations Number of iterations done to calculate the access time average
  -v, --verbose             verbose mode displays text sent and received
  -N, --no-irq              disable Interrupts (=> polling status)
```

Example :

```
export SNAP_TRACE=0x0
snap_maint -v

snap_latency_eval            // default parameters are 100 iterations / Action timeout 16777215 (0xFFFFFF)reads
snap_latency_eval -T 10      //The action will send a timeout sequence and exit after 10*15 reads
snap_latency_eval -n 2000    //Calculates the access time average on 2000 access
snap_latency_eval -n 200 v   //Calculates the access time average on 200 access and display the text sent and
                                received by the application
```

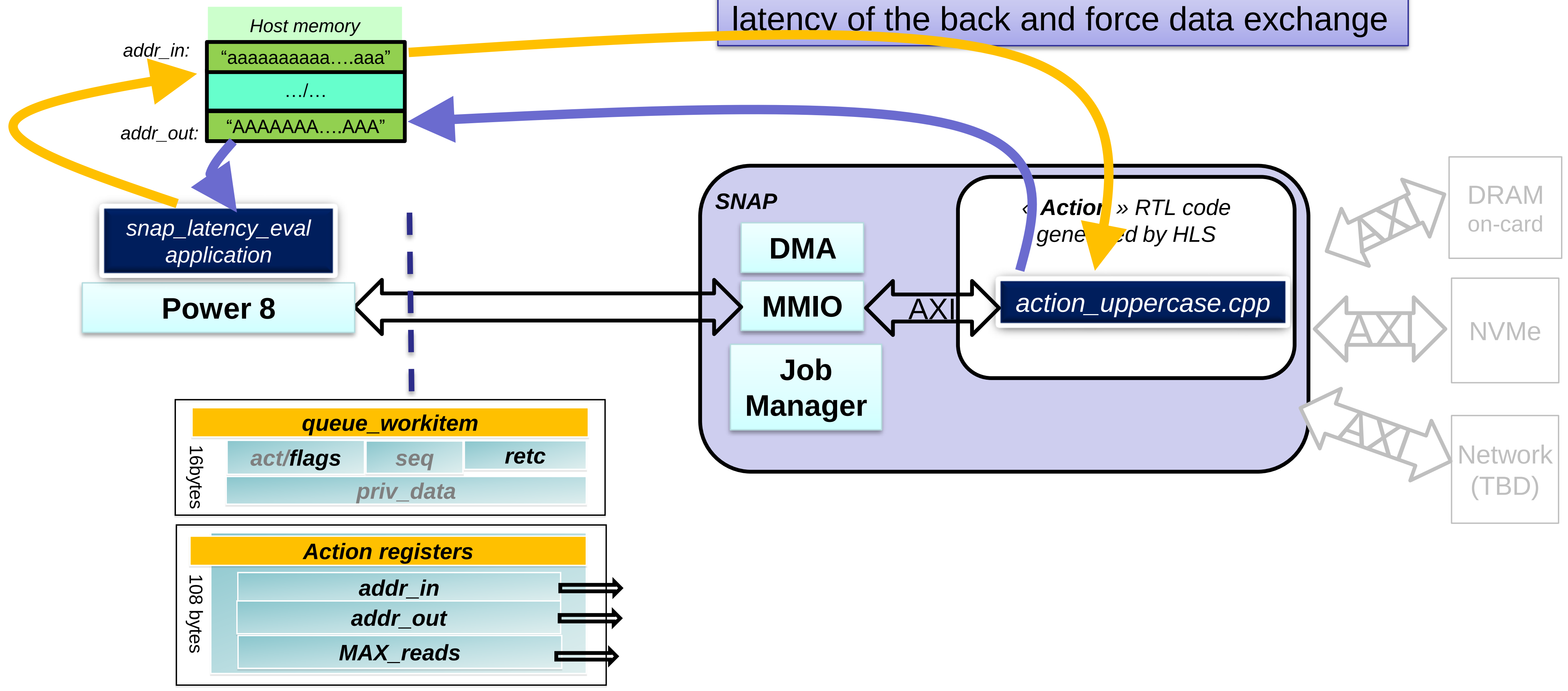
```
$SNAP_TRACE=0xF snap_latency_eval -n 50
```

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

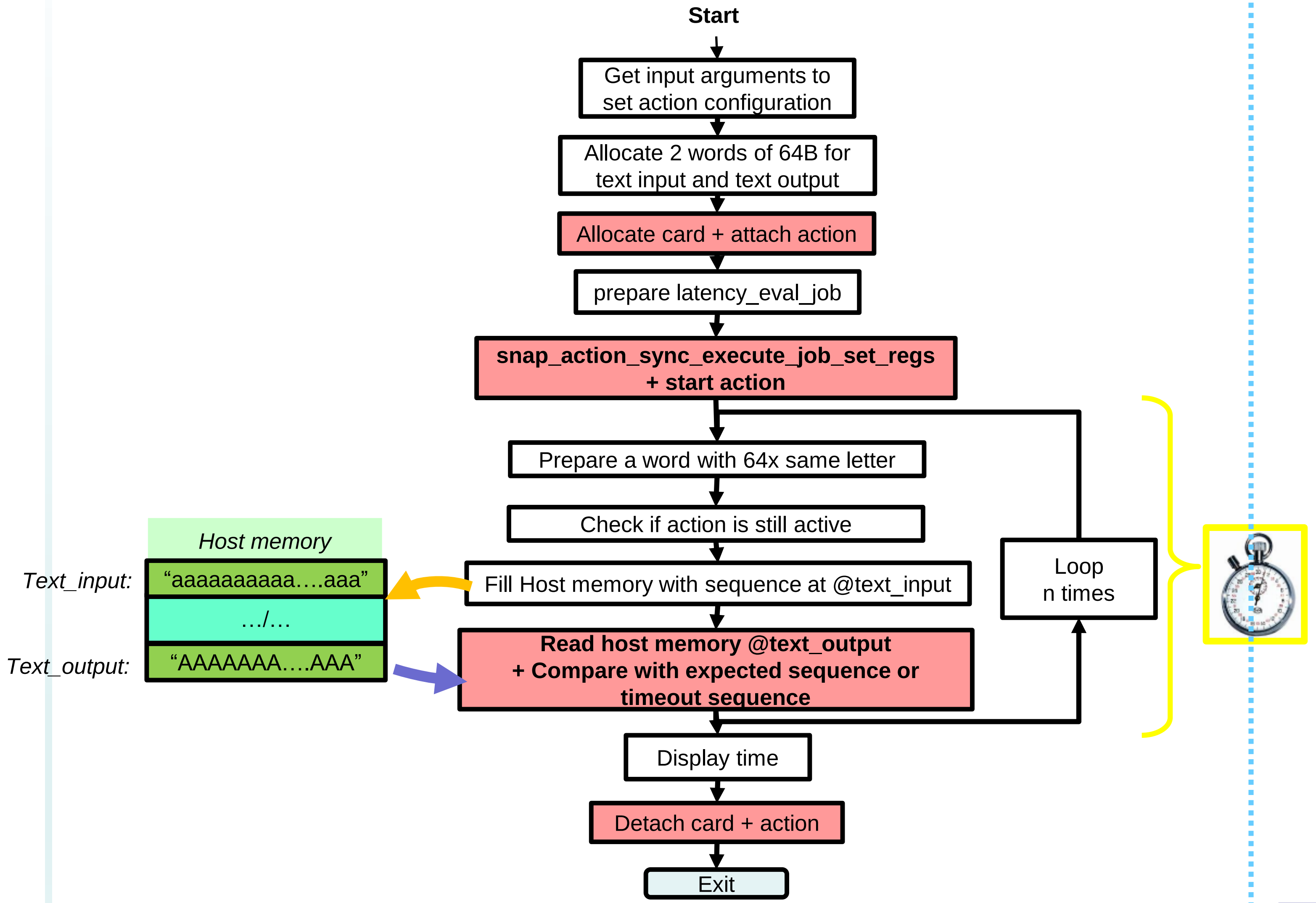
SNAP_TRACE=0x0 ➔	no debug trace
SNAP_TRACE=0xF ➔	full debug trace
SNAP_CONFIG=FPGA ➔	hardware execution
SNAP_CONFIG=CPU ➔	software execution

latency_eval registers

Purpose: code is written to minimize the latency of the back and force data exchange



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

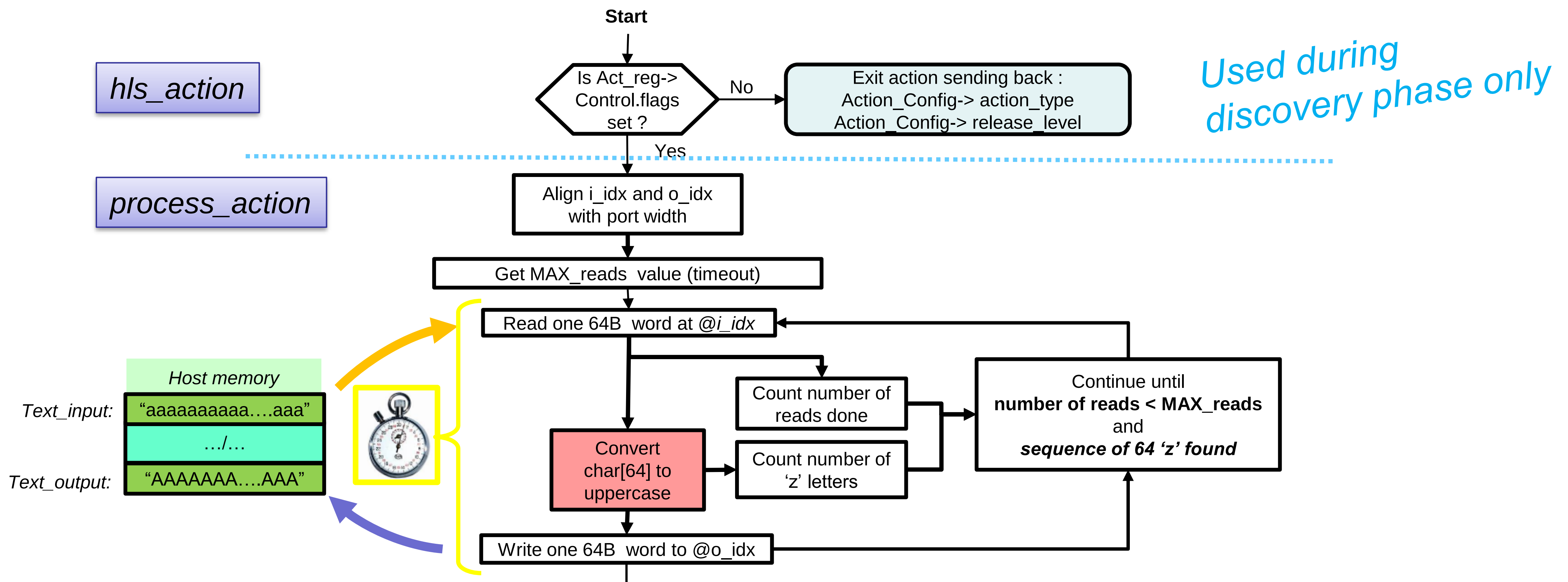

Start
Get input arguments to set action configuration
Allocate 2 words of 64B for text input and text output
Allocate card + attach action
prepare latency_eval_job
snap_action_sync_execute_job_set_regs + start action
Prepare a word with 64x same letter
Check if action is still active
Fill Host memory with sequence at @text_input
Read host memory @text_output + Compare with expected sequence or timeout sequence
Loop n times
Display time
Detach card + action
Exit
Host memory
Text_input:
"aaaaaaaaaa….aaa"
…/…
Text_output:
"AAAAAAA….AAA"

Application calling the software action:
latency_eval processing code is not relevant here since this software action should be coded as an independent and parallel thread to show the same effect.
The purpose here is to show how coding the interaction between the application and the hardware action.

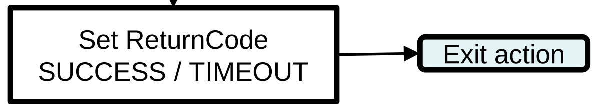
Application: snap_latency_eval.c

CPU executed action: action_lowercase.c

Hardware action Code : what's in it?



hls_action
process_action
Start
Is Act_reg-> Control.flags set ?
No
Yes
Exit action sending back : Action_Config-> action_type Action_Config-> release_level
Used during discovery phase only
Align i_idx and o_idx with port width
Get MAX_reads value (timeout)
Read one 64B word at @*i_idx*
Count number of reads done
Continue until **number of reads < MAX_reads** and ***sequence of 64 'z' found***
Convert char[64] to uppercase
Count number of 'z' letters
Write one 64B word to @o_idx
Host memory
Text_input:
"aaaaaaaaaa….aaa"
…/…
Text_output:
"AAAAAAA….AAA"



End of action:
- Specific sequence found (64 'z')
- Max reads value reached (timeout)
  ➔ specific timeout sequence written (64 '!')

FPGA executed Action: action_uppercase.cpp

Constants - Ports

Constants: ➔ $ACTION_ROOT = snap/actions/hls_helloworld

Constant name	Value	Type	Definition location	Usage
LATENCY_EVAL_ACTION_TYPE	0x10141009	Fixed	$ACTION_ROOT/include/action_changecase.h	latency_eval ID - list is in snap/ActionTypes.md
RELEASE_LEVEL	0x00000020	Variable	$ACTION_ROOT/hw/action_uppercase.**H**	release level – user defined

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	NOT used
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 intersect_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	[snap_addr]**in**.addr (LSB)							
0x3C45	0x114	0x194	[snap_addr]**in**.addr (MSB)							
0x3C46	0x118	0x198	[snap_addr]**in**.size							
0x3C47	0x11C	0x19C	[snap_addr]**in**.flags (SRC, DST, ...)		[snap_addr]**in**.type (DRAM, NVME,..)					
0x3C48	0x120	0x1A0	[snap_addr]**src_result**.addr (LSB)							
0x3C49	0x124	0x1A4	[snap_addr]**src_result**.addr (MSB)							
0x3C4A	0x128	0x1A8	[snap_addr]**src_result**.size							
0x3C4B	0x12C	0x1AC	[snap_addr]**src_result**.flags (SRC, DST, ...)		[snap_addr]**src_result**.type (DRAM, NVME,..)					
0x3C4C	0x130	0x1B0	**MAX_reads (LSB)**							
0x3C4D	0x134	0x1B4	**MAX_reads (LSB)**							

$ACTION_ROOT/hw/action_uppercase.H

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

$ACTION_ROOT/include/action_changecase.h

```
typedef struct latency_eval_job {
        struct snap_addr in;     /* input data */
        struct snap_addr out;   /* offset table */
        uint64_t MAX_reads;     /* setting MAX number of reads (timeout)*/
} latency_eval_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 a POWER8 and POWER9 servers

hls_latency_eval	POWER8 (S822LC - CAPI1.0) + N250S (PCIe Gen3x8)	POWER9 (AC922 - CAPI2.0) + RCXVUP (PCIe Gen3x16)
Average latency for 10,000 access	**2.496 µs**	**1.096 µs**

```
$ ./snap_latency_eval -n 10000
PARAMETERS:
  type_in:      0 HOST_DRAM
  addr_in:      0000010004da0000
  type_out:     0 HOST_DRAM
  addr_out:     0000010004db0000
  size_in/out:  00000040
  prepare latency_eval job of 40 bytes size
Action Timeout: MAX reads set to: 16777215
SNAP registers set + action start took 6 usec
SNAP action processing for 10000 iteration is 1.099570 usec
SUCCESS
SNAP latency_eval closing action took 11 usec
```

To run these performances, run the following:

```
$ snap_maint -v
$ snap_latency_eval -n 10000
```

What do we measure?

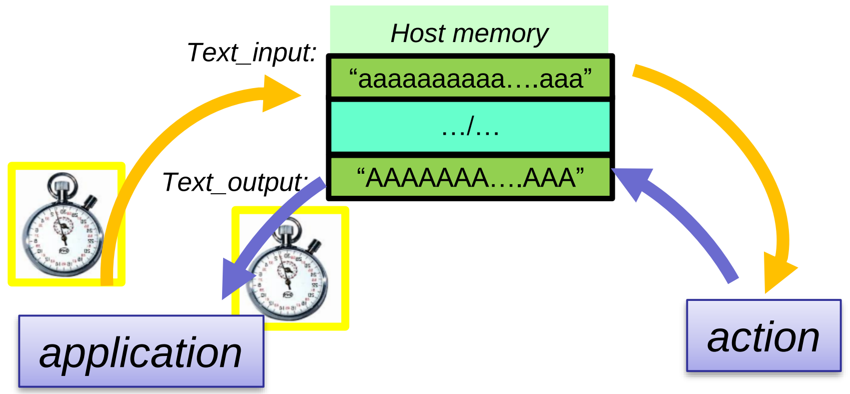
These numbers are the measurements results of the following sequence time:

START TIME MEASUREMENT

- The **application writes** a 64B word to host memory @in
- The **action reads** (continuously) the host memory address @in
- The **action process** the 64B word read to uppercase letters
- The **action writes** back the 64B word result to the host memory at @out
- The **application reads** continuously the host memory at @out and compares it to the expected word until it matches (or get action timeout sequence)

STOP TIME MEASUREMENT

This measurement is done 10,000 times to evaluate a good average time



Path of improvements

History of this document and of the action release level

V2.0: initial document