CAPI SNAP Education Series:
User Guide

CAPI SNAP Education
hls_sponge : howto?
V2.2

Architecture of the SNAP git files

Compilation options (as a result of make snap_config menu) :
export ACTION_ROOT= =$SNAP_ROOT/actions / hls_sponge/
export SDRAM_USED=FALSE ➔ **"Enable SDRAM" must be unset in Kconfig menu)**

SNAP

actions

- hls_sponge
  - **sw**
    - **snap_checksum.c** — Main ***application*** program
    - **action_checksum.c** — "CPU executed" ***action*** program
  - **hw**
    - **action_sponge.cpp** — "FPGA executed" ***action*** program
    - action_sponge.H — *Specific constants + "data exchanged" structures used by this action (i.e. Action_Release_Level)*
  - include
    - **action_checksum.h** — *Specific constants + "data exchanged" structures used by sw action and hw action + application (i.e. Action_Type)*
  - tests — *Regression test used to show how to run this hls_example*
  - doc — *Location of this document*

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 and applications*
- tools
  - snap_maint
  - snap_find_card — *programs used to "discover" available actions, do basic settings and attach / detach actions*

Action overview

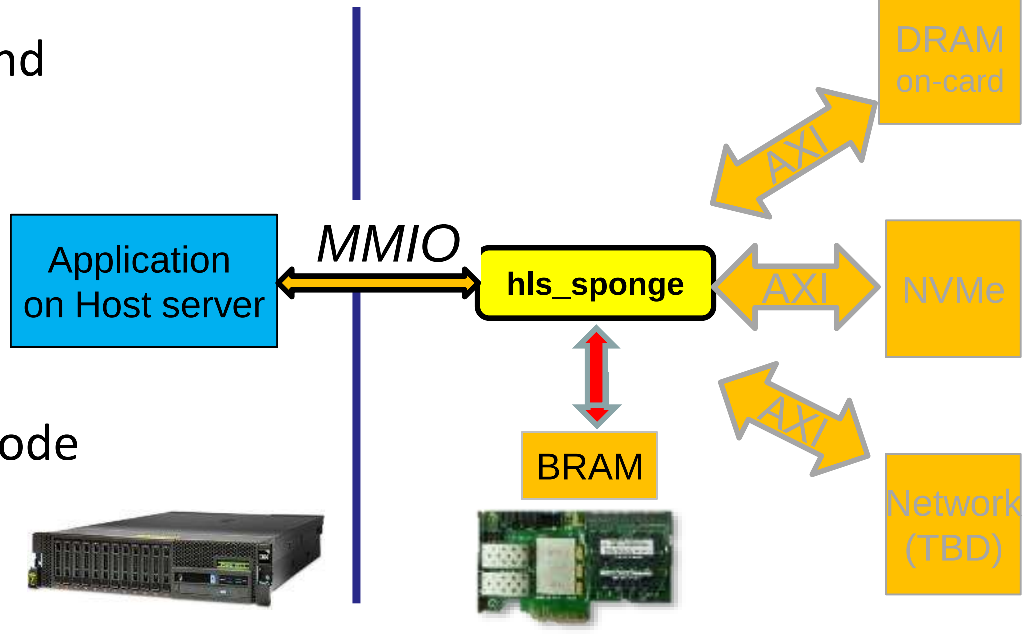
<u>Purpose:</u> Port a pure mathematical function written in C and see how much performance HLS can reach with it.

- **Measure development time to port code**
- **Compare CPU and FPGA performances**

  ➔ Multi-threading for CPU and for FPGA

<u>When to use it:</u>

- Understand HLS constraints when porting standard C code
- Understand HLS basic pragmas that can improve code performance.



CAPI SNAP Enabled Card

<u>Memory management:</u>

- No memory access done since data are generated and checked by the code

<u>Known limitations:</u>

- Only test_speed was optimized for HLS. The "key" calculation functions ***test_sha3*** and ***test_shake*** are functional but not optimized

The SHA3 “test_speed” program structure:

➔ 2 parameters : NB_TEST_RUNS, NB_ROUNDS

As measuring time with HLS is not obvious, the “time” loop was modified so that parallelism could be done. The goal stays to execute the maximum times the keccakf algorithm per second.

Code used was downloaded from: https://github.com/mjosaarinen/tiny_sha3

```
main() {
  for(run_number = 0; run_number < NB_TEST_RUNS; run_number++)
  {
       if(nb_elmts > (run_number % freq))
         checksum ^= test_speed(run_number);
  }
}
```

NB_TEST_RUNS = 65,536

Parallel loops

Recursive loops

```
uint64_t test_speed (const uint64_t run_number)
{
for( i=0; i < 25; i++ )
      st[i] = i + run_number;
bg = clock;
do {
     for( i=0; I < NB_ROUNDS; i++ )
          sha3_keccakf(st, st);
} while ((clock - bg) < 3 * CLOCKS_PER_SEC);
for( i=0; i < 25; i++ )
      x += st[i];
return x;
}
```

NB_ROUNDS=65,536

Math function

```
void sha3_keccakf(uint64_t st_in[25], uint64_t st_out[25])
{
   for (round = 0; round < KECCAKF_ROUNDS; round++)
       processing Theta + Rho Pi + Chi
}
```

KECCAKF_ROUNDS = 24 → 24 calls calling the algorithm process

Application usage

Usage:

```
Usage: ./snap_checksum [-h] [-v, --verbose] [-V, --version]
  -C, --card <cardno> can be (0...3)
  -x, --threads <threads>   depends on the available CPUs.
  -i, --input <file.bin>    input file.
  -S, --start-value <checksum_start> checksum start value.
  -A, --type-in <CARD_RAM, HOST_RAM, ...>.
  -a, --addr-in <addr>      address e.g. in CARD_RAM.
  -s, --size <size>         size of data.
  -c, --choice <SPEED,SHA3,SHAKE,SHA3_SHAKE>  sponge specific input.
  -n, --number of elements <nb_elmts> sponge specific input.
  -f, --frequency <freq>        sponge specific input.(up to 65536)
  -m, --mode <CRC32|ADLER32|SPONGE> mode flags.
  -t, --timeout             Timeout in sec (default 3600 sec).
  -N, --no irq              Disable IRQs
```

Example :

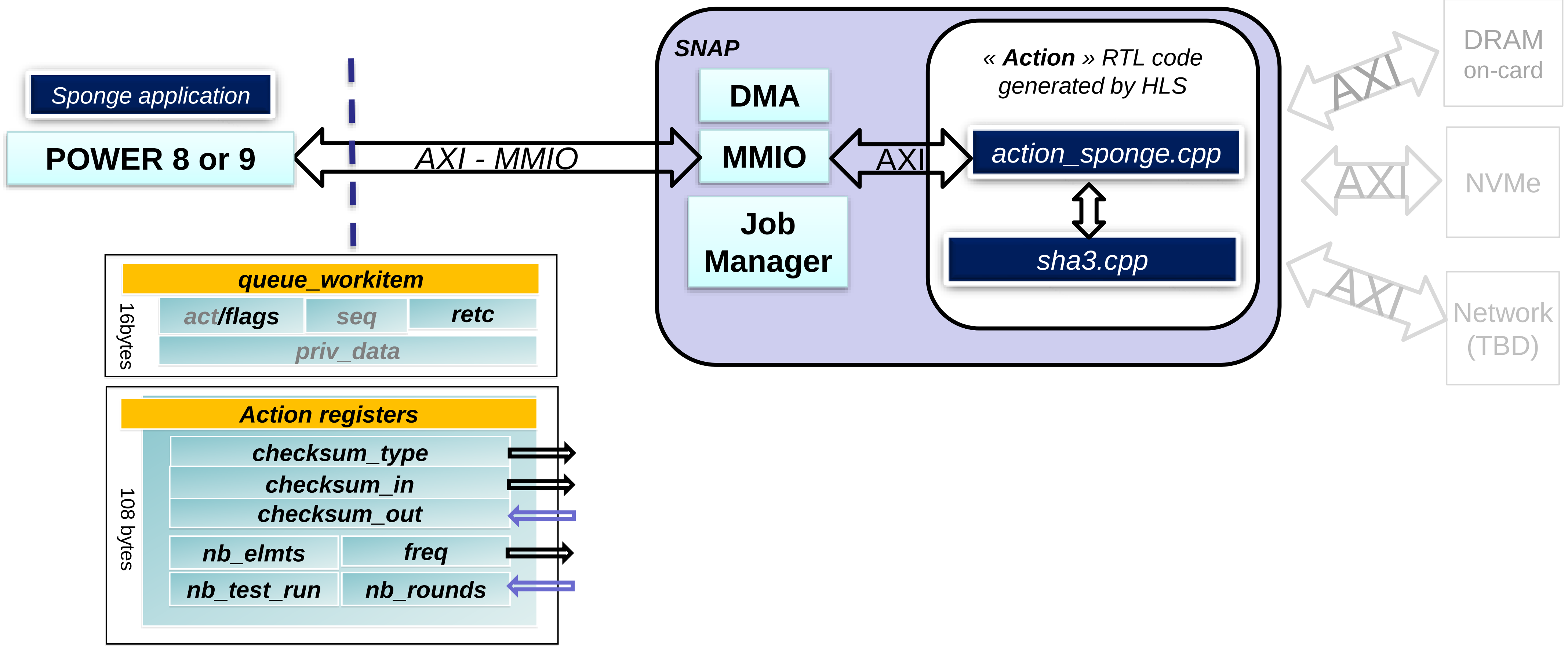
```
export SNAP_TRACE=0x0
$SNAP_ROOT/software/tools/snap_maint
#echo Generation of 65536*2/65536 = 2 calls
SNAP_CONFIG=FPGA ./snap_checksum -C1 -vv -t2500 -mSPONGE -I -cSPEED -n1 -f65536
SNAP_CONFIG=FPGA ./snap_checksum -C1 -vv -t2500 -mSPONGE -I -cSPEED -n128 -f65536
SNAP_CONFIG=FPGA ./snap_checksum -C1 -vv -t2500 -mSPONGE -I -cSPEED -n4096 -f65536
#echo Generation of 65536*1/4 = 16384 calls
SNAP_CONFIG=FPGA ./snap_checksum -C1 -vv -t2500 -mSPONGE -I -cSPEED -n1 -f4

#echo to run tests SHA3 or/and SHAKE
SNAP_CONFIG=FPGA ./snap_checksum -mSPONGE -I -t800 -cSHA3
SNAP_CONFIG=FPGA ./snap_checksum -mSPONGE -I -t800 -cSHAKE
SNAP_CONFIG=FPGA ./snap_checksum -mSPONGE -I -t800 -cSHA3_SHAKE
```

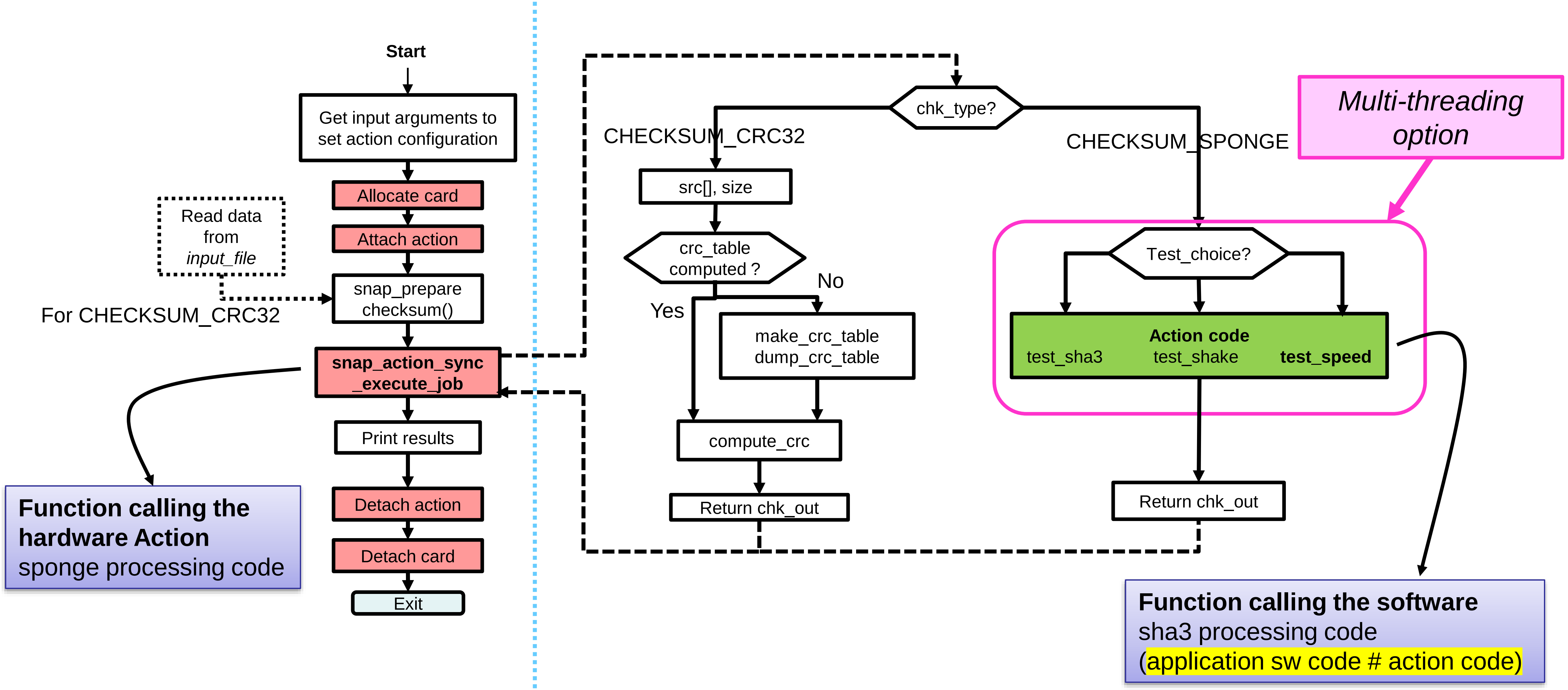
Options:

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

Sponge/checksum registers



Application Code calling action code : reorganized

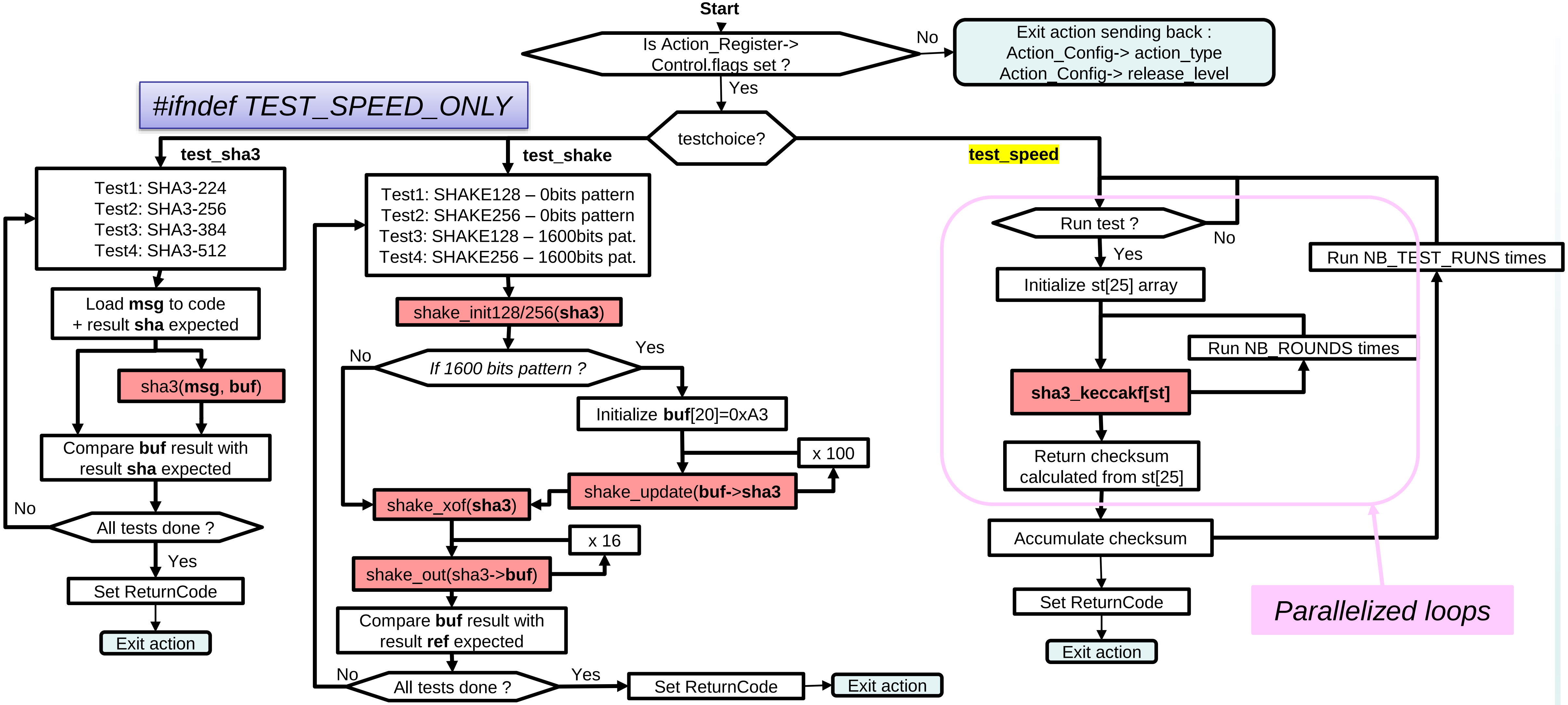


Application: snap_checksum.c

Action : action_checksum.c

Subroutine : sha3.c (see following slides)

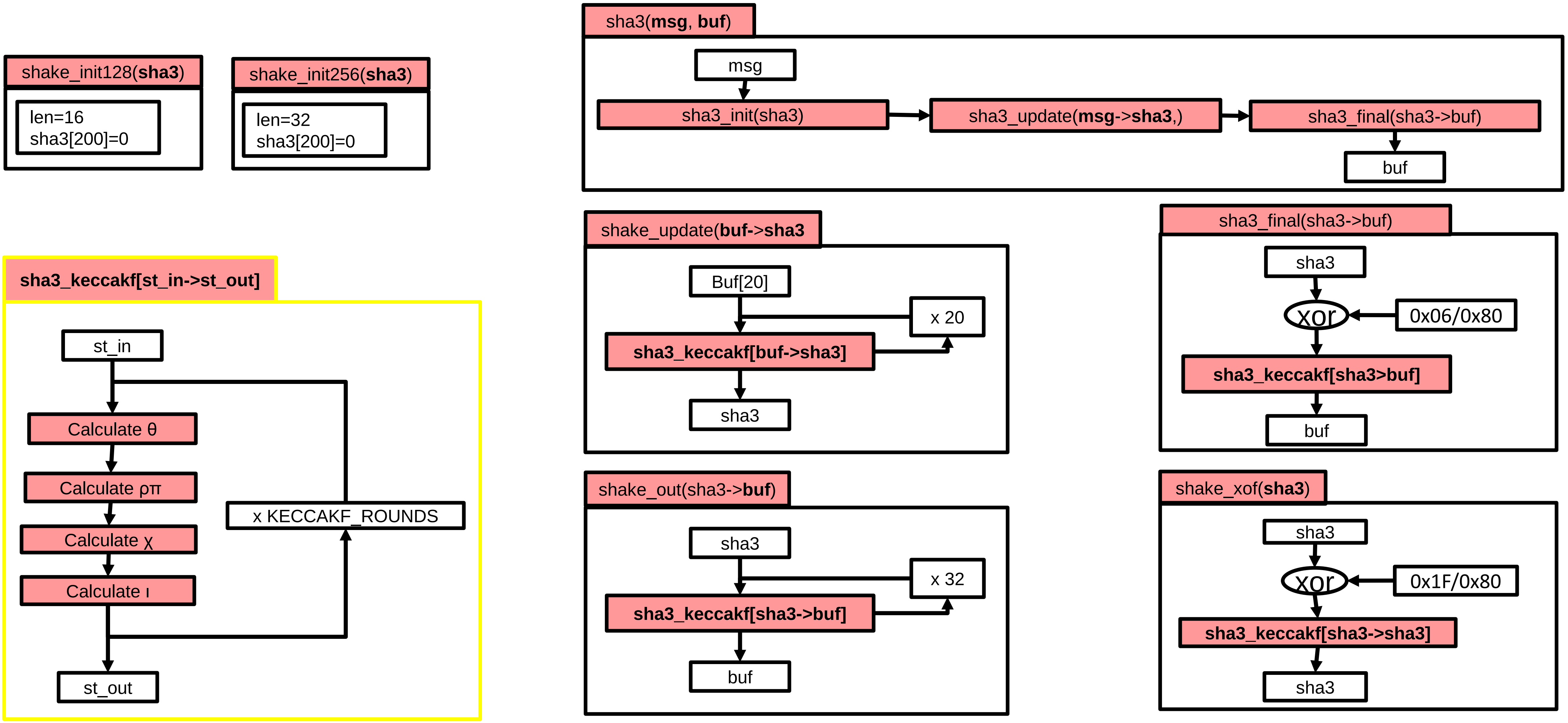
Action checksum Code : what's in it?



Action: action_sponge.cpp

Application-Action checksum Code : what's in it?

IBM Power Systems



Action: sha3.cpp = **Application: sha3.c**

Constants - Ports

Constants:

Constant name	Value	Type	Definition location	Usage
CHECKSUM_ACTION_TYPE	0x10141001	Fixed	$ACTION_ROOT/include/action_checksum.h	Checksum ID - list is in snap/ActionTypes.md
RELEASE_LEVEL	0x00000021	Variable	$ACTION_ROOT/hw/action_checksum.**H**	release level – user defined
NB_ROUNDS	65536	Variable	$ACTION_ROOT/hw/action_checksum.**H**	Number of recursive loops done in test_speed function
NB_TEST_RUNS	65536	Variable	$ACTION_ROOT/hw/action_checksum.**H**	Number of parallel loops done in test_speed function
KECCAKF_ROUNDS	24	Variable	$ACTION_ROOT/hw/sha3.H	Number of loops done in keccakf function

For simulation, reduce these numbers to very low values (i.e. 8 or 16) or simulation will be VERY long

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	**NO**
nvme	NVMe data bus in/out Addr : 32bits - Data : 32bits	*No (soon)*

export SDRAM_USED=FALSE

MMIO Registers

Read and Write are considered from the application / software side

act_reg.Control			*This header is initialized by the SNAP job manager. The action will update the Return code and read the flags value.*							
CONTROL			*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			*Action specific - user defined - need to stay in 108 Bytes(padding done in $ACTION_ROOT/hw/action_sponge.H)*			
checksum_job_t			*This is the way for application and action to exchange information through this set of registers*			
Simu - WR	Write@	Read@	3	2	1	0
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	chk_in (LSB)			
0x3C49	0x124	0x1A4	chk_in (MSB)			
0x3C4A	0x128	0x1A8	chk_out (LSB)			
0x3C4B	0x12C	0x1AC	chk_out (MSB)			
0x3C4C	0x130	0x1B0	chk_type			
0x3C4D	0x134	0x1B4	test_choice			
0x3C4E	0x138	0x1B8	nb_elmts			
0x3C4F	0x13C	0x1BC	freq			
0x3C50	0x140	0x1C0	nb_test_runs			
0x3C51	0x144	0x1C4	nb_rounds			

$ACTION_ROOT/hw/action_sponge.H

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

$ACTION_ROOT/include/action_checksum.h

```
typedef struct checksum_job {
          struct snap_addr in;    /* in:  input data */
          uint64_t chk_in;        /* in:  checksum input */
          uint64_t chk_out;       /* out: checksum output */
          uint32_t chk_type;      /* in:  CRC32, ADDLER32 */
          uint32_t test_choice;   /* in:  special parameter for sponge */
          uint32_t nb_elmts;      /* in:  special parameter for sponge */
          uint32_t freq;          /* in:  special parameter for sponge */
          uint32_t nb_test_runs;  /* out: special parameter for sponge */
          uint32_t nb_rounds;     /* out: special parameter for sponge */
} checksum_job_t;
```

$SNAP_ROOT/software/include/snap_types.h

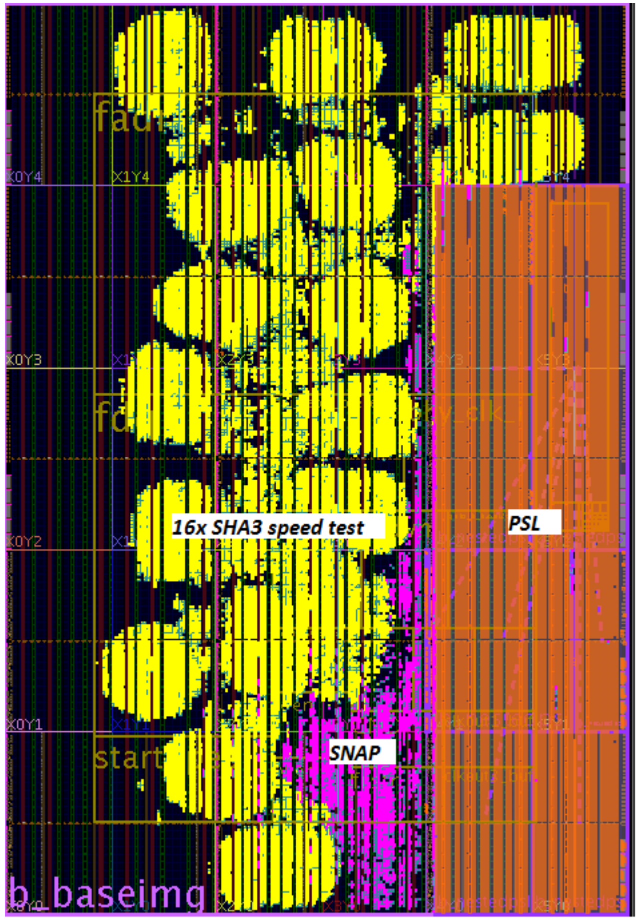
```
typedef struct snap_addr {
        uint64_t addr;
        uint32_t size;
        snap_addrtype_t type;          /* DRAM, NVME, ... */
        snap_addrflag_t flags;        /* SRC, DST, EXT, ... */
} snap_addr_t;
```

$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;
```

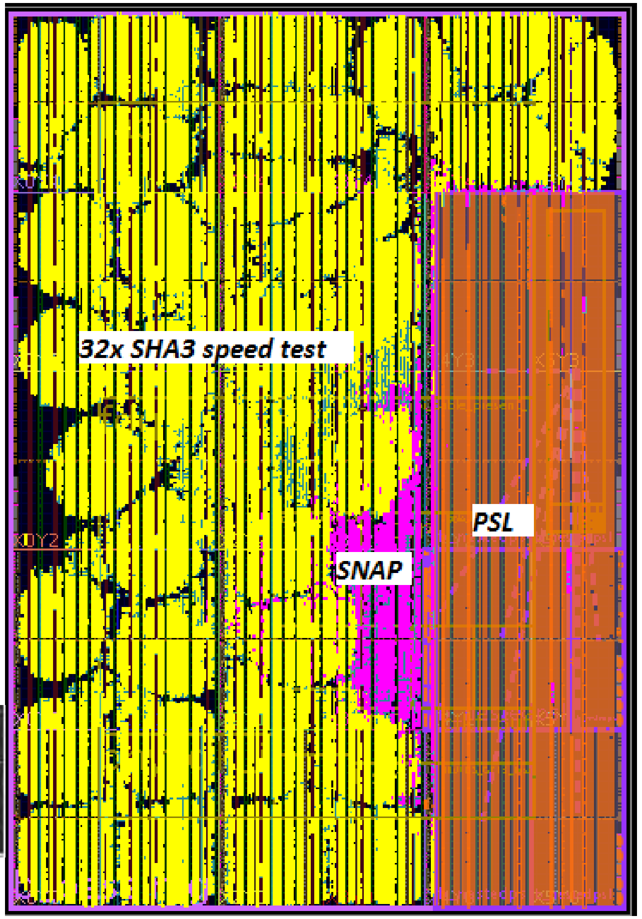
FPGA area used by the design

16 test_speed functions in parallel:
HLS_SYN_CLOCK=2.827000,HLS_SYN_LAT=2713646082,
HLS_SYN_MEM=96,HLS_SYN_DSP=0,HLS_SYN_FF=74689,HLS_SYN_LUT=171,112



Site Type	Used	Fixed	Available	Util%
CLB LUTs	151842	69756	331680	45.78
LUT as Logic	137137	55073	331680	41.35
LUT as Memory	14705	14683	146880	10.01

32 test_speed functions in parallel:
HLS_SYN_CLOCK=2.827000,HLS_SYN_MEM=192,
HLS_SYN_FF=142929,HLS_SYN_LUT=337,640



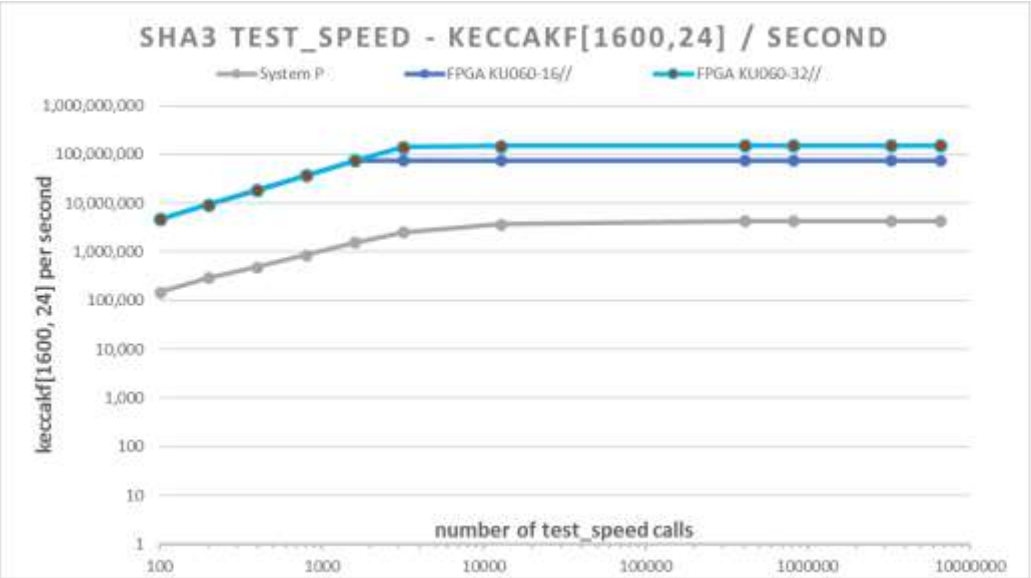
To fill at much as possible the FPGA for the speed_test, set:
In include/action_checksum.h
➔ *#define TEST_SPEED_ONLY*
In hw/hls_checksum.cpp line 355:
➔ *#pragma HLS UNROLL factor=32 (or more if FPGA is larger than a KU060)*

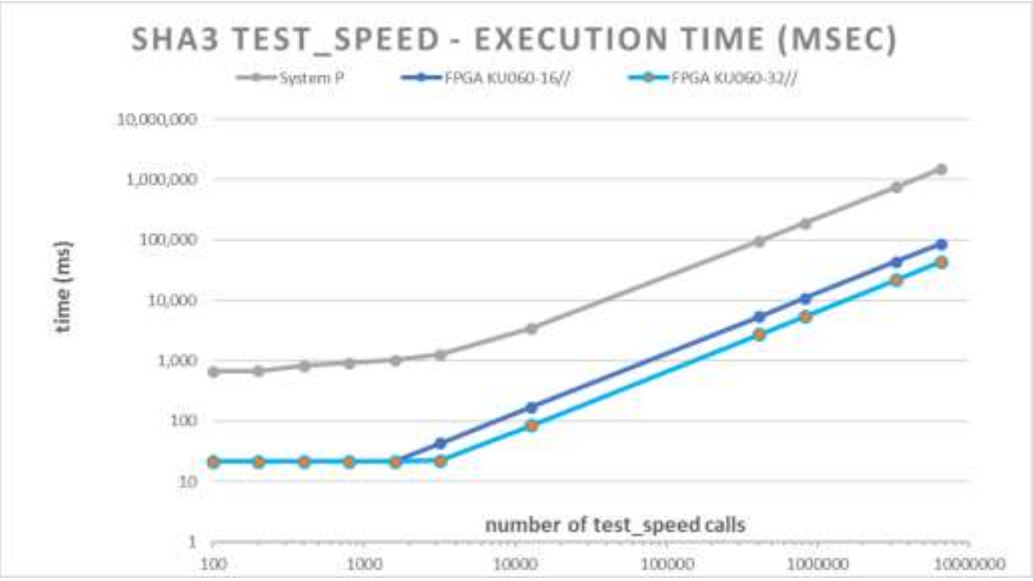
Site Type	Used	Fixed	Available	Util%
CLB LUTs	225387	69756	331680	67.95
LUT as Logic	210666	55073	331680	63.51
LUT as Memory	14721	14683	146880	10.02

➔ Vivado HLS estimation is **very pessimistic** and Vivado doing a **very good optimization** of resources!

SHA3 speed_test benchmark : FPGA is 35x faster than CPU

						slices/16 FPGA KU060-16//	slices/16 FPGA KU060-16//	slices/32 FPGA KU060-32//	slices/32 FPGA KU060-32//	CPU (antipode) 16 cores - 160 threads System P	CPU (antipode) 16 cores - 160 threads System P
NB_ROUNDS	**NB_TEST_RUNS**	**nb_elmts**	**freq**	**test_speed calls**	Checksum	(keccak per sec)	(msec)	**(keccak per sec)**	**(msec)**	(keccak per sec)	(msec)
100,000	65,536	1	65,536	100,000	3e05f34be7cc0386	4,624,491	22	**4,666,573**	21	**149,575**	669
100,000	65,536	2	65,536	200,000	2ccef6d61b67ad2f	9,248,983	22	**9,334,453**	21	**295,786**	676
100,000	65,536	4	65,536	400,000	0796ca863ac8273f	18,498,821	22	**18,668,036**	21	**488,441**	819
100,000	65,536	8	65,536	800,000	0018c0972c9227d2	36,990,799	22	**37,330,845**	21	**865,289**	925
100,000	65,536	16	65,536	1,600,000	5bd139d5bf8dad3a	73,995,283	22	**74,672,143**	21	**1,572,084**	1,018
100,000	65,536	32	65,536	3,200,000	a0c267468cf1e051	74,722,709	43	**143,568,576**	22	**2,539,064**	1,260
100,000	65,536	128	65,536	12,800,000	05c290e99ff8b7ae	75,279,062	170	**149,900,457**	85	**3,699,211**	3,460
100,000	65,536	4,096	65,536	409,600,000	ed3ff1c664125abb	75,465,691	5,428	**150,837,950**	2,715	**4,267,759**	95,975
100,000	65,536	8,192	65,536	819,200,000	cfd69627069b3e3e	75,468,917	10,855	**150,900,077**	5,429	**4,303,717**	190,347
100,000	65,536	32,767	65,536	3,276,700,000	eb4c1384fa60e252	75,468,889	43,418	**150,937,573**	21,709	**4,344,618**	754,198
100,000	65,536	65,536	65,536	6,553,600,000	38c7143fc6c46500	75,471,578	86,835	**150,941,821**	43,418	**4,352,266**	1,505,790





What else ?

Path of improvement ?

1. Improving data types cast
2. Modify the code to replace the typecasting done to circumvent the union so that **test_sha3** and **test_shake** functions can get normal/good performances. Up to now, adaptation to HLS has been done but not optimized for these 2 functions.

History of this document and of the action release level

V2.0: initial document
V2.1: new files directory structure applied
V2.2 : minor corrections