

*CAPI SNAP Education Series:  
User Guide*

*CAPI SNAP Education  
HLS table intersection: howto ?  
V2.2*



April 3<sup>rd</sup>, 2018

SNAP Framework built on Power™ CAPI technology

# Architecture of the SNAP git files

SNAP hierarchy	Files	
<ul style="list-style-type: none"> <li>SNAP           <ul style="list-style-type: none"> <li>actions               <ul style="list-style-type: none"> <li>hls_intersect</li> </ul> </li> </ul> </li> </ul>		
<ul style="list-style-type: none"> <li> <ul style="list-style-type: none"> <li>hw_h</li> </ul> </li> </ul>	action_intersect_h.H action_intersect_h.cpp	Specific constants and data types used by this HLS action
<ul style="list-style-type: none"> <li> <ul style="list-style-type: none"> <li>hw_s</li> </ul> </li> </ul>	action_intersect_s.H action_intersect_s.cpp	<div>“FPGA executed” action program</div> <div>“CPU executed” action program</div>
<ul style="list-style-type: none"> <li> <ul style="list-style-type: none"> <li>sw</li> </ul> </li> </ul>	action_intersect.c snap_intersect.c	<div>Main application program</div>
<ul style="list-style-type: none"> <li> <ul style="list-style-type: none"> <li>include</li> </ul> </li> </ul>	action_intersect.h action_intersect_common.h	Specific constants and data types used by “CPU” action Common constants and data types used by both “FPGA” and “CPU” action
<ul style="list-style-type: none"> <li> <ul style="list-style-type: none"> <li>doc</li> </ul> </li> </ul>	This Document	
<ul style="list-style-type: none"> <li> <ul style="list-style-type: none"> <li>tests</li> </ul> </li> </ul>	test_0x10141005.sh test_0x10141006.sh other scripts	test script for “Hash” method implementation test script for “Sort” method implementation
<ul style="list-style-type: none"> <li>hardware</li> </ul>		
<ul style="list-style-type: none"> <li> <ul style="list-style-type: none"> <li>sim</li> </ul> </li> </ul>		Sim directory used for simulation (“make sim” to simulation the “FPGA” action)
<ul style="list-style-type: none"> <li> <ul style="list-style-type: none"> <li>build/Images</li> </ul> </li> </ul>		Location of bitstream files used to program FPGA for actual hardware
<ul style="list-style-type: none"> <li>software</li> </ul>		
<ul style="list-style-type: none"> <li> <ul style="list-style-type: none"> <li>include</li> </ul> </li> </ul>	snap_types.h	Common constants + structures (i.e, snap_addr) used by all hls actions and applications
<ul style="list-style-type: none"> <li> <ul style="list-style-type: none"> <li>tools</li> </ul> </li> </ul>	snap_maint snap_find_card	programs used to “discover” available actions, do basic settings and attach / detach actions

## Compilation option :

```
export ACTION_ROOT=~snap/hardware/actions/hls_intersect/vhdl
export SDRAM_USED=TRUE
```

## Action overview

**Purpose:** Given two **unsorted** arrays, write a function that returns the **third array** which is the **intersection** of the two arrays. This means that the resulting array should only contain elements that appear in both input arrays. Order of elements in the resulting array is irrelevant.

- Evaluate how tables can be managed with HLS
- Compare CPU and FPGA performances

### When to use it:

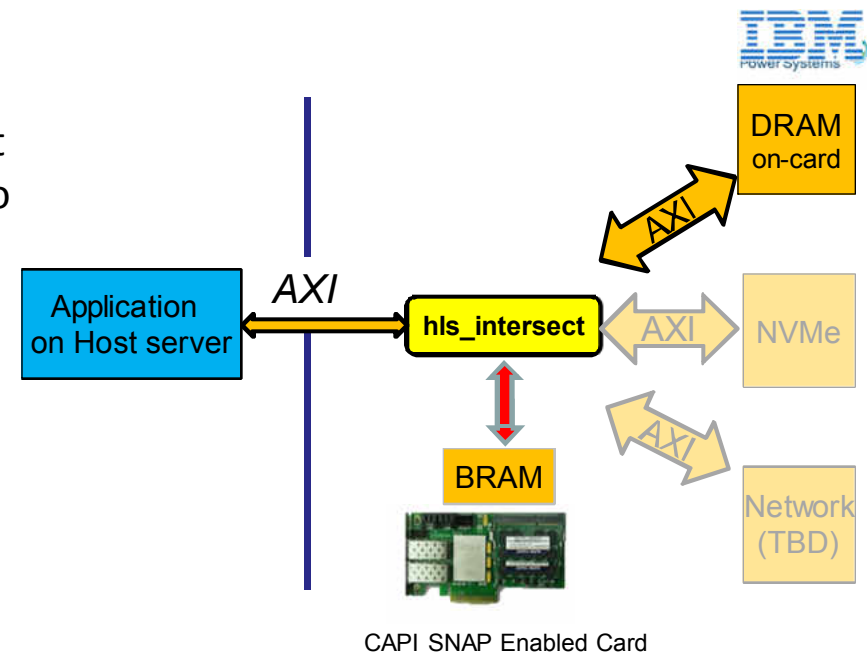
- Understand HLS constraints when working with large database

### Memory management:

- All memory allocation is managed by the application

### Known limitations:

- All data are 64 bytes aligned to ease access



## Two methods in implementation


1. Hash
2. Sort-merge

## Hash Table Method Details in HLS Action

- Element size = 64bytes (63 valid characters + '\0')
- Maximum input array size is set to 1GB (16 Million elements)
- Hash Method:
  - A hash function to calculate the hash key for each element.
  - Hash Table Entries =  $2^{22}$  (4 Million). Each entry can hold addresses of 15 elements which share the same hash key, and an additional *count* indicating how many elements are in this entry.
  - Hash Entry stores the DDR address (4bytes) of source Arrays instead of data content (64bytes).
  - If more than 15 elements are having the same hash key, overflow happens. --Not solved now. Return FAILURE.
  - Use FPGA DDR to store Hash Table ( 4Million \* (15+1) \* 4bytes = 256Mbytes)
  - Use Block RAM to hold the status of Hash Table (4Mbits, which takes about 10% of total 38Mbits BRAM in KU060)
    - Initialize to zero at the beginning
    - 1 bit for each Hash Entry: 0=not used, 1=used
    - very fast to know if one hash entry has been used or not (in 1 cycle)

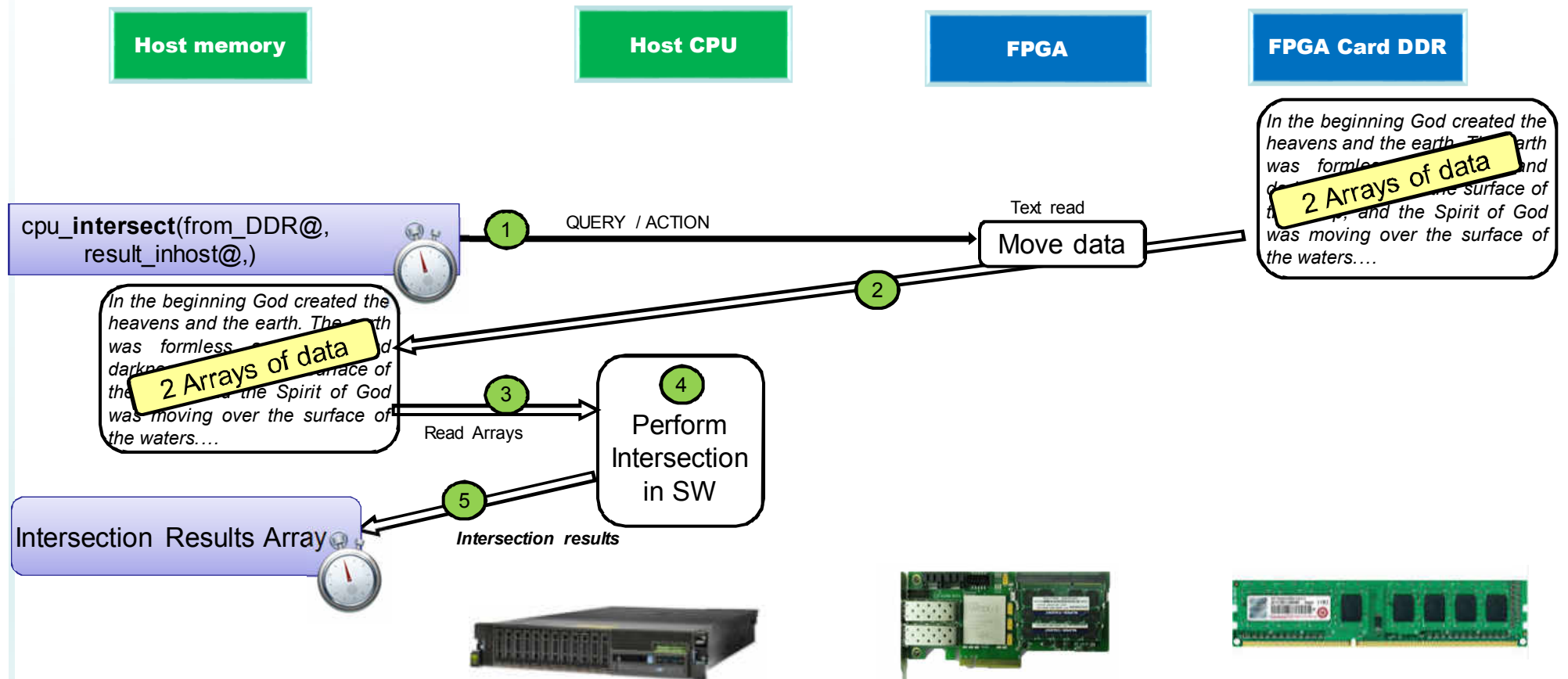
## Sort-Merge Method Details in HLS Action

- Element size = 64bytes (63 valid characters + '\0')
- Maximum input array size is set to 1GB (16 Million elements)
- 1<sup>st</sup> round Bubble-sort in local BRAM + 2<sup>nd</sup> round Merge-sort on DDR
- Multiple bubble sort engines running in parallel
  - Each engine deals with 32 elements
  - 16 engines
- As a common problem of sorting, above parameters and even the full structure still in adjusting
- After two tables are sorted, do a final intersection operation.
- **NOTE: two methods are not implemented simultaneously in hardware. Read "hls\_intersect/README.md for more information"**

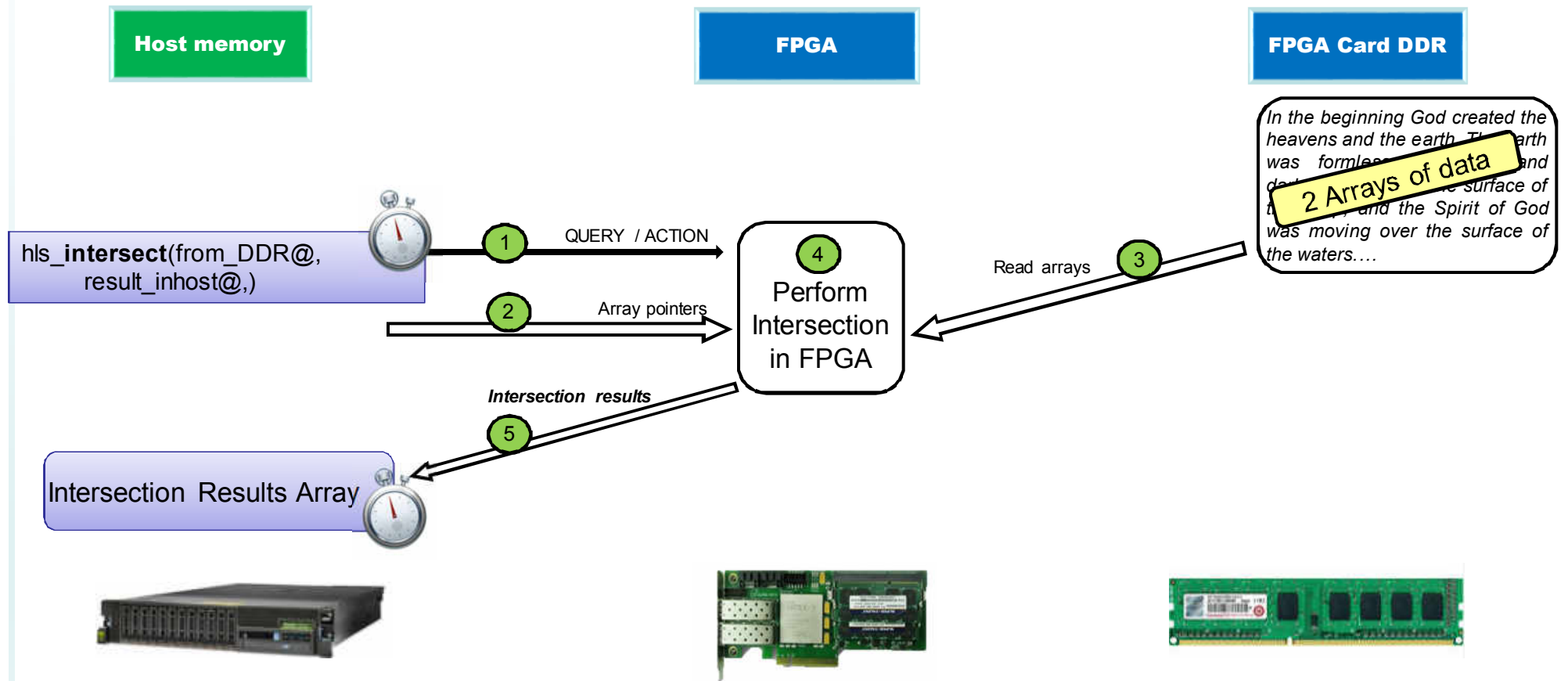


```
merge_intersection():
while (i < N1 && j < N2) {
    if (table1[i] == table2[j]) {
        output to res_table;
        i++;
        j++;
    } else if (table1[i] > table2[j])
        i++;
    else
        j++;
}
```

# SW Intersection: processing data from DDR in CPU



# FPGA Intersection: processing data from DDR in **FPGA**



## Action usage

**Usage:** `./snap_intersect [-h] [-v, --verbose] [-V, --version]`

`-C, --card <cardno>` can be (0...3)  
`-t, --timeout <seconds>` timeout seconds.

-----  
`-i, --input1 <file1.txt>` input file 1.  
`-j, --input2 <file2.txt>` input file 2.  
`-o, --output <result.txt>` output file.

-----  
`-n, --num <int>` How many elements in the table for random generated array. - default is 20  
`-l, --len <int>` length of the random string. - default is 1  
`-s, --software` Use software approach.  
`-m, --method <0/1/2>` 0: compare one by one (Slow and only in SW).  
1: Use hash table  
2: Use sort and merge  
`-I, --irq` Enable Interrupts

### Example :

```
export SNAP_TRACE=0x0
$SNAP_ROOT/software/tools/snap_maint
```

```
#echo Random generation of 2 tables with default table size (20 entries)
```

```
SNAP_CONFIG=0x0 ./snap_intersect -C1 -vv
```

```
#echo Random generation of 2 tables with table size = 40 entries => intersection in CPU
```

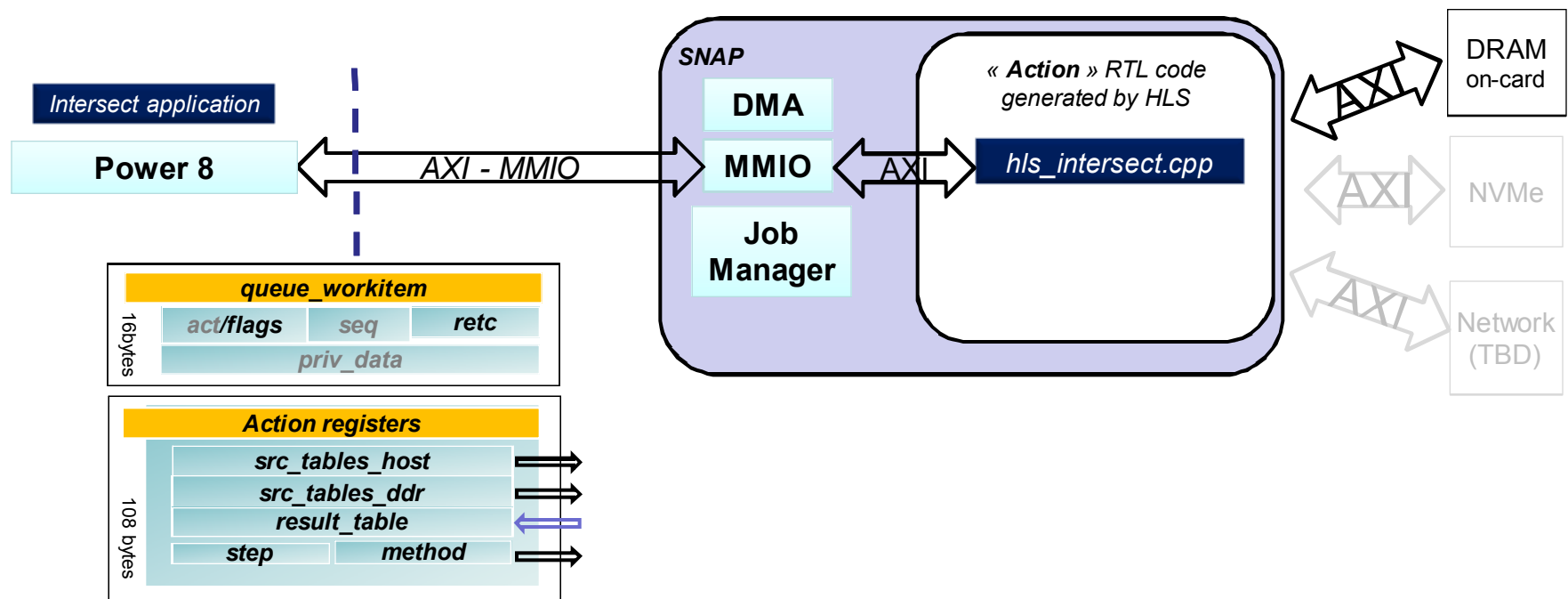
```
SNAP_CONFIG=0x0 ./snap_intersect -C1 -vv -s -n 40
```

#### Options:

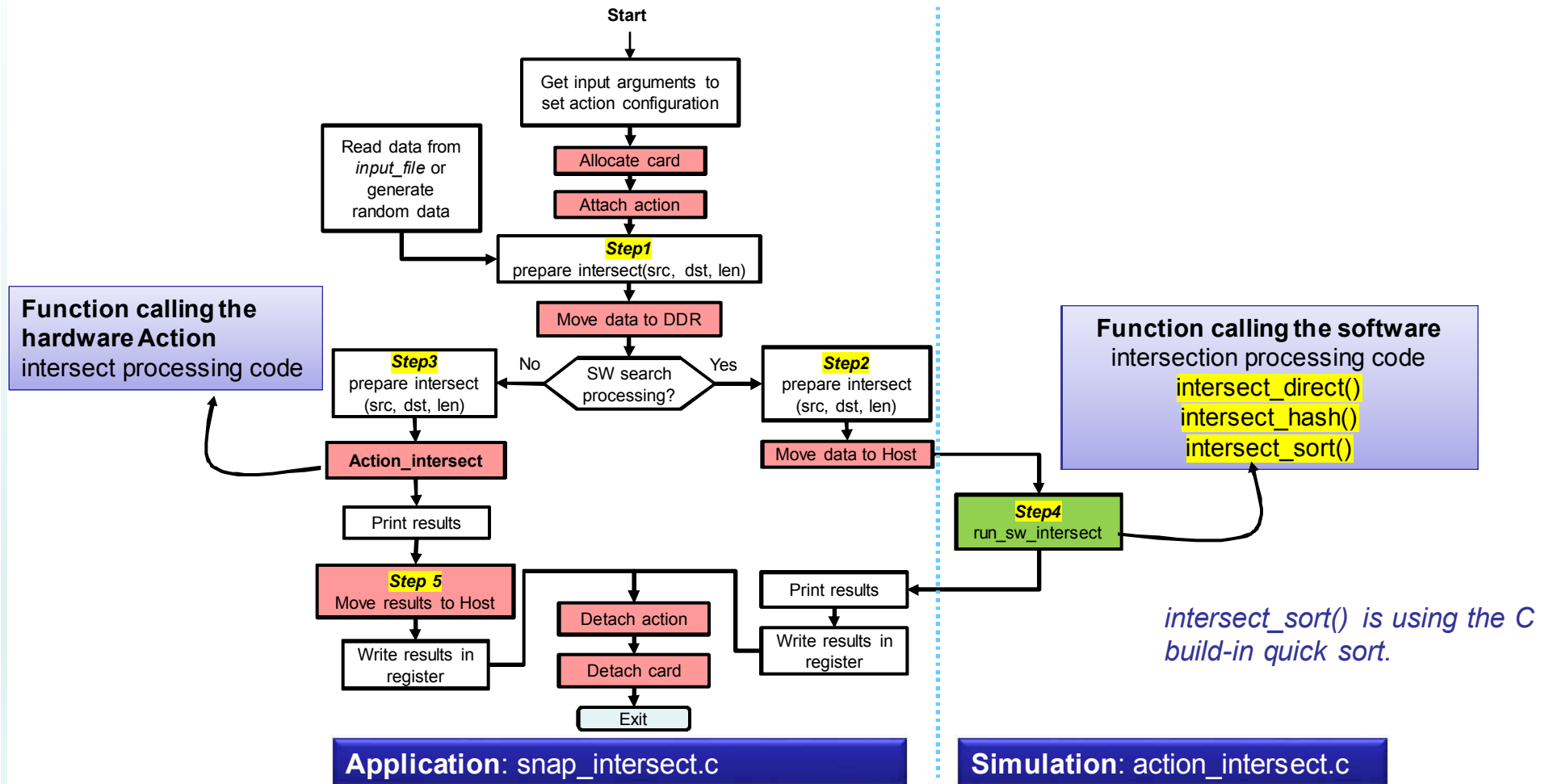
`SNAP_TRACE = 0x0` → no debug trace  
`SNAP_TRACE = 0xF` → full debug trace  
`SNAP_CONFIG = 0x0` → FPGA execution  
`SNAP_CONFIG = 0x1` → CPU execution



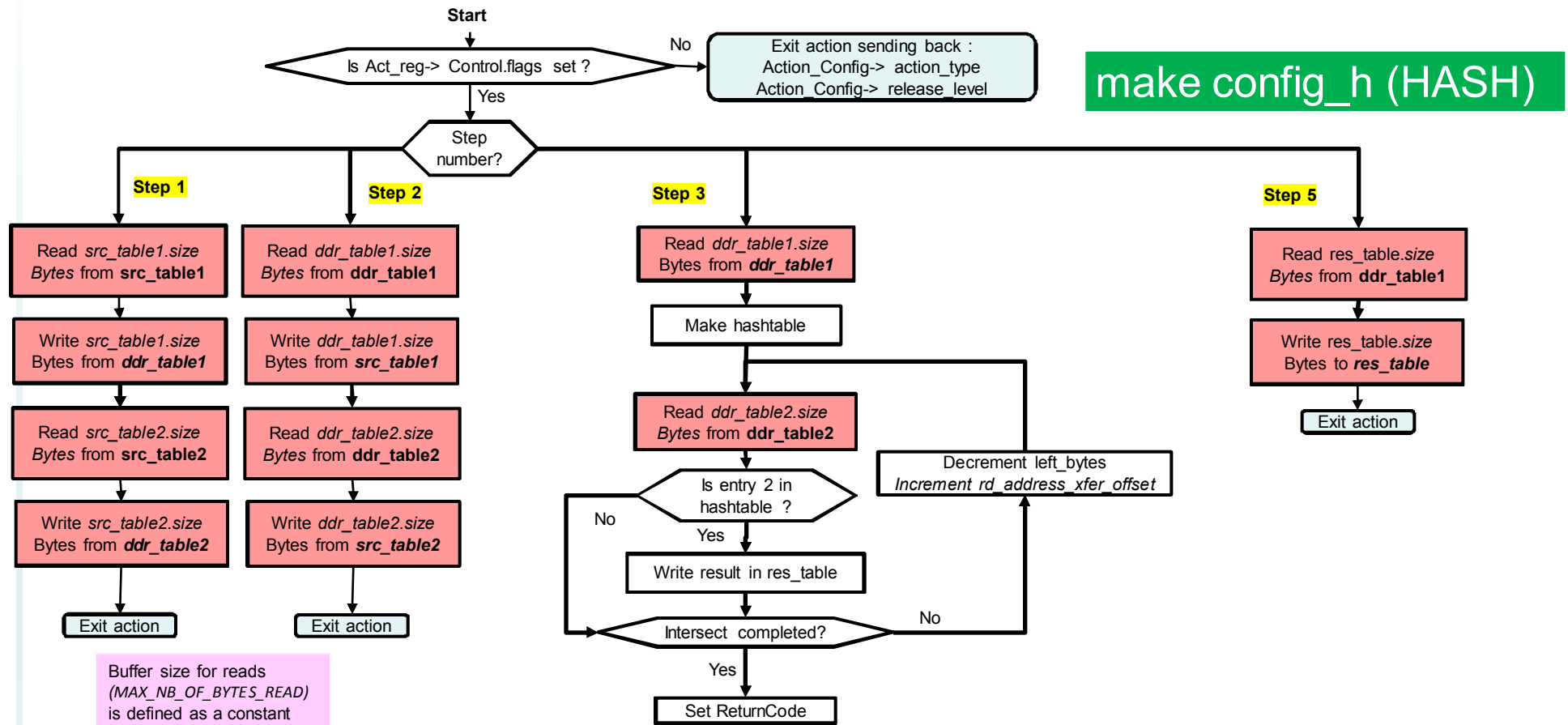
# Intersection registers



# Application Code : what's in it?

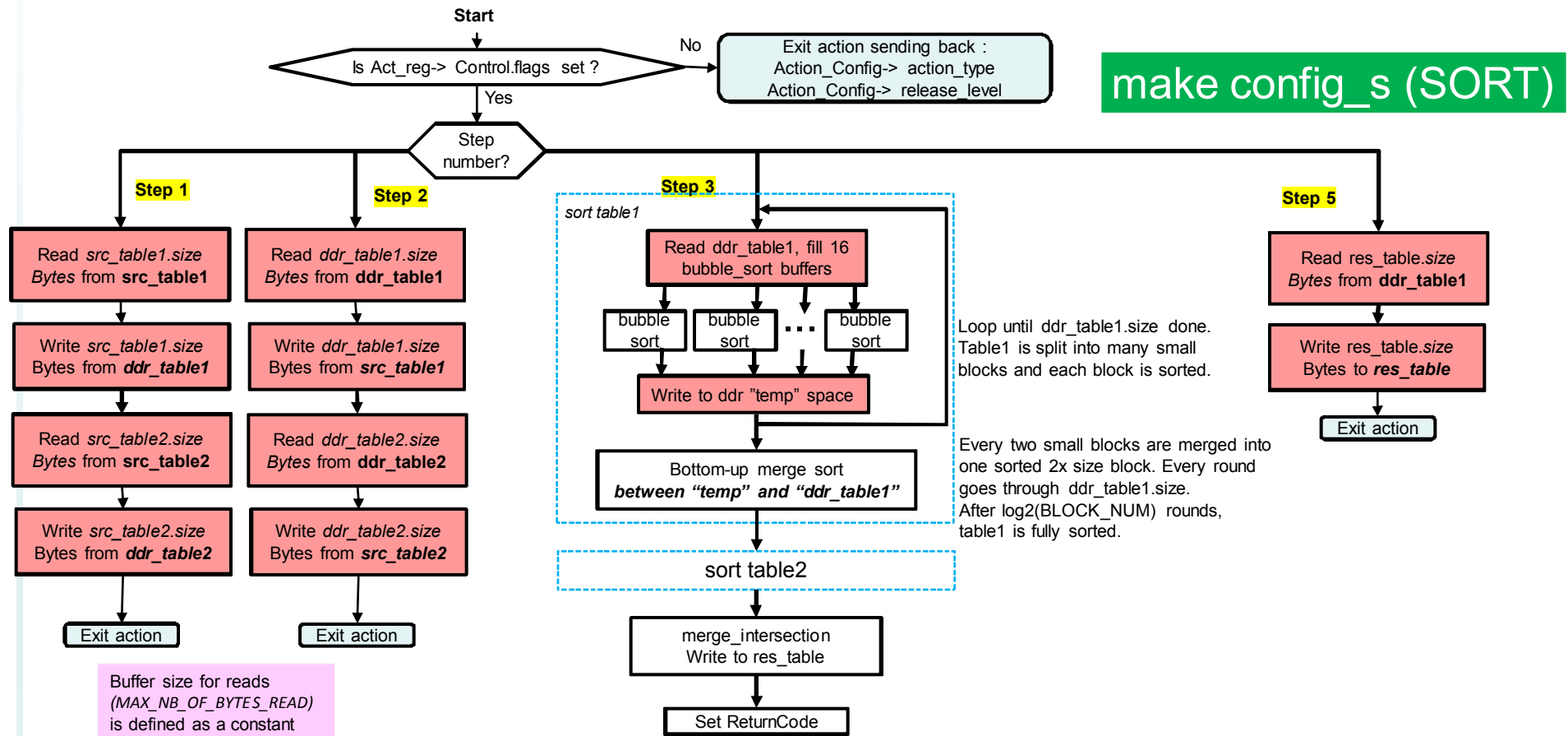


# Action intersect Code : what's in it?



Action: hls\_intersect.cpp

# Action intersect Code : what's in it?



Action: hls\_intersect.cpp

## Constants - Ports

### Constants:

Constant name	Value	Type	Definition location	Usage
INTERSECT_H_ACTION_TYPE	0x10141005	Fixed	include/action_intersect_common.h	Intersect ID - list is in snap/ActionTypes.md
INTERSECT_S_ACTION_TYPE	0x10141006	Fixed		
HW_RELEASE_LEVEL	0x00000017	Variable	hw_h/hls_intersect_h.cpp hw_s/hld_intersect_s.cpp	release level – user defined
NUM_TABLES	2	Variable	include/action_intersect_common.h	Number of tables
MAX_TABLE_SIZE	(uint64_t) (1<<30) = 1GBytes	Variable	include/action_intersect_common.h	Max size (in bytes) for each table
HW_HT_ENTRY_NUM_EXP	22	Operation specific	hw_h/hls_intersect_h.H	Hardware Hash table implementation is different compared to software due to the limit of RAM resource and Timing.

### 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	Not used

# 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.		hardware/action_examples/hls_hashjoin/action_intersect.H
CONTROL	If the flags value is 0, then action sends only the action_RQ_config_reg value and exit the action, otherwise it will process the action		hardware/action_examples/include/hls_snap.H

Simu - WR	Write@	Read@	3	2	1	0	Typical Writevalue	Typical Read value
0x3C40	0x100	0x180	sequence		flags	shortaction type	f001_01_00	
0x3C41	0x104	0x184	Retc (return code 0x102/0x104)				0	0x102 - 0x104 SUCCESS/FAILURE
0x3C42	0x108	0x188	PrivateData				c0febabe	
0x3C43	0x10C	0x18C	PrivateData				deadbeef	

action_reg.Data	Action specific - user defined - need to stay in 108 Bytes(padding done in hardware/action_examples/hls_hashjoin/action_intersect.H		hardware/action_examples/hls_hashjoin/action_intersect.H
intersect job t	This is the way for application and action to exchange information through this set of registers		software/examples/action_hashjoin.h

Simu - WR	Write@	Read@	3	2	1	0	Typical Writevalue	Typical Read value
0x3C44	0x110	0x190	[snap_addr]src_tables_host[0].addr (LSB)					=> snap_addr defined in software/include/snap_types.h
0x3C45	0x114	0x194	[snap_addr]src_tables_host[0].addr (MSB)					
0x3C46	0x118	0x198	[snap_addr]src_tables_host[0].size					
0x3C47	0x11C	0x19C	[snap_addr]src_tables_host[0].flags (SRC, DST, ...)		[snap_addr]src_tables_host[0].type (DRAM, NVME,...)			
0x3C48	0x120	0x1A0	[snap_addr]src_tables_host[1].addr (LSB)					
0x3C49	0x124	0x1A4	[snap_addr]src_tables_host[1].addr (MSB)					
0x3C4A	0x128	0x1A8	[snap_addr]src_tables_host[1].size					
0x3C4B	0x12C	0x1AC	[snap_addr]src_tables_host[1].flags (SRC, DST, ...)		[snap_addr]src_tables_host[1].type (DRAM, NVME,...)			
0x3C4C	0x130	0x1B0	[snap_addr]src_tables_ddr[0].addr (LSB)					
0x3C4D	0x134	0x1B4	[snap_addr]src_tables_ddr[0].addr (MSB)					
0x3C4E	0x138	0x1B8	[snap_addr]src_tables_ddr[0].size					
0x3C4F	0x13C	0x1BC	[snap_addr]src_tables_ddr[0].flags (SRC, DST, ...)		[snap_addr]src_tables_ddr[0].type (DRAM, NVME,...)			
0x3C50	0x140	0x1C0	[snap_addr]src_tables_ddr[1].addr (LSB)					
0x3C51	0x144	0x1C4	[snap_addr]src_tables_ddr[1].addr (MSB)					
0x3C52	0x148	0x1C8	[snap_addr]src_tables_ddr[1].size					
0x3C53	0x14C	0x1CC	[snap_addr]src_tables_ddr[1].flags (SRC, DST, ...)		[snap_addr]src_tables_ddr[1].type (DRAM, NVME,...)			
0x3C54	0x150	0x1D0	[snap_addr]result_table.addr (LSB)					
0x3C55	0x154	0x1D4	[snap_addr]result_table.addr (MSB)					
0x3C56	0x158	0x1D8	[snap_addr]result_table.size					
0x3C57	0x15C	0x1DC	[snap_addr]result_table.flags (SRC, DST, ...)		[snap_addr]result_table.type (DRAM, NVME,...)			
0x3C58	0x160	0x1E0	step					
0x3C59	0x164	0x1E4	method					
0x3C5A	0x168	0x1E8						
0x3C5B	0x16C	0x1EC						
	0x170	0x1F0						
	0x174	0x1F4						
	0x178	0x1F8						
0x17C	0x1FC	0x1FC	HLS reserved					

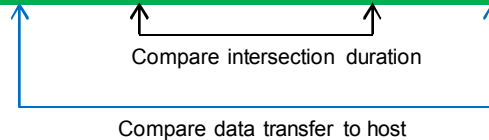
# Performance results (USE\_HASH)

SW CPU procedure:  
 (Step2) Copy Source  
 arrays from DDR to Host  
 (Step4) SW intersection

FPGA procedure:  
 (Step3) HW intersection  
 (Step5) Copy Result array  
 from DDR to Host

1. Data transfer steps (step2 for CPU and step5 for FPGA) only takes a small portion compared to the intersection time.
2. HW intersection gets overflow when table size is bigger for this dataset.

			Memcopy	Memcopy	SW	Memcopy	HW	Memcopy		Acceleration	Acceleration	Overall		
			Host->DDR	DDR->Host (copy src)		Host->DDR		DDR->Host (copy results)		Step3.vs.Step4	Step5.vs.Step2	(3+5).vs.(2+4)	DDR->Host bandwidth	Host->DDR bandwidth
#	Element number	Two tables size	Step1	Step2	Step4	Step1	Step3	Step5	Result number	Calculation faster	Copy less data		(32burst) MB/s	(32burst) MB/s
1	256	32K	165	46	31087	144	183	30	131	169.87	1.53	146.16	712.35	227.56
2	512	64K	169	69	26764	171	353	39	267	75.82	1.77	68.45	949.80	383.25
3	1024	128K	224	117	25151	228	646	79	524	38.93	1.48	34.85	1120.27	574.88
4	2048	256K	342	217	31046	338	1320	126	1012	23.52	1.72	21.62	1208.04	775.57
5	4096	512K	566	408	29830	576	2608	215	2094	11.44	1.90	10.71	1285.02	910.22
6	8192	1M	1022	794	31344	1022	5171	349	4094	6.06	2.28	5.82	1320.62	1026.00
7	16384	2M	1931	1577	37608	1929	10318	637	8176	3.64	2.48	3.58	1329.84	1087.17
8	32768	4M	3774	3149	67848	3774	20672	1253	16364	3.28	2.51	3.24	1331.95	1111.37
9	65536	8M	7472	6271	130202	7474	49989	2512	32883	2.60	2.50	2.60	1337.68	1122.37
10	131072	16M	14882	12477	370224	14871	117509	4901	65611	3.15	2.55	3.13	1344.65	1128.18
11	262144	32M	29909	24751	1669516	30459	269683	9557	130748	6.19	2.59	6.07	1355.68	1101.63
12	524288	64M	60259	49744	6606585	60437	669981	17050	261822	9.86	2.92	9.69	1349.08	1110.39



# Performance results (USE\_SORT)

1. HW sort is slower than software's quicksort due to current poor single read performance to DDR AXI interface.

			Memcopy	Memcopy	SW	Memcopy	HW	Memcopy			Acceleration	Overall		
			Host->DDR	DDR->Host		Host->DDR		DDR->Host		Step3.vs.Step4	Step5.vs.Step2	(3+5).vs.(2+4)	DDR->Host bandwidth	Host->DDR bandwidth
#	Element number	Two tables size	Step1	Step2	Step4	Step1	Step3	Step5	Result number	Calculation faster	Copy less data		(32burst) MB/s	(32burst) MB/s
1	256	32K	82	46	900	79	887	32	123	1.01	1.44	1.03	712.35	414.78
2	512	64K	103	70	1595	105	2141	38	260	0.74	1.84	0.76	936.23	624.15
3	1024	128K	162	119	4167	162	5107	81	533	0.82	1.47	0.83	1101.45	809.09
4	2048	256K	279	216	9533	273	12135	125	1027	0.79	1.73	0.80	1213.63	960.23
5	4096	512K	501	412	15290	511	27842	176	2025	0.55	2.34	0.56	1272.54	1026.00
6	8192	1M	969	794	29395	967	63446	323	4025	0.46	2.46	0.47	1320.62	1084.36
7	16384	2M	1888	1577	58081	1867	141110	654	8195	0.41	2.41	0.42	1329.84	1123.27
8	32768	4M	3691	3131	120745	3711	313279	1257	16459	0.39	2.49	0.39	1339.61	1130.24
9	65536	8M	7356	6241	257185	7357	683646	2428	32657	0.38	2.57	0.38	1344.11	1140.22
10	131072	16M	14796	12465	551175	14767	1491830	4886	65694	0.37	2.55	0.38	1345.95	1136.13
11	262144	32M	29840	24713	1194737	29859	3211901	9492	131065	0.37	2.60	0.38	1357.76	1123.76
12	524288	64M	60294	50429	2624655	60159	6922383	17900	262486	0.38	2.82	0.39	1330.76	1115.52
13	1048576	128M	121595	99480	5697982	122221	14760017	33536	523989	0.39	2.97	0.39	1349.19	1098.16
14	2097152	256M	244569	199358	12235627	246266	31504604	59799	1048435	0.39	3.33	0.39	1346.50	1090.02
15	4194304	512M	524094	448991	27493510	530435	66676492	118513	2096618	0.41	3.79	0.42	1195.73	1012.13
16	8388608	1G	1157677	861155	56912374	1157870	141314608	229377	4195111	0.40	3.75	0.41	1246.86	927.34



## What else ?

1. FPGA is powerful for such functions:
  - Calculate the hash function to generate a key in 1 or 2 cycles (considering the combination logic timing)
  - Compare the two 64byte elements in 2 cycles
  - Parallel bubble-sort: Take 16 engines and 32 elements for example, can complete sorting 16 small blocks in 2048cycles ( $32 \times 32 / 2$  iterations \* 4cycles per iteration)
2. Random access to DDR interface has many cycles' overhead, which restricts the overall performance.
  - For USE\_SORT: Bottom-up merge, and merge\_intersection both operate on single read from DDR.
  - For USE\_HASH: When hash\_used[i] = 1, still need to fetch the element from DDR, and this is also single read.
  - Add a cache?