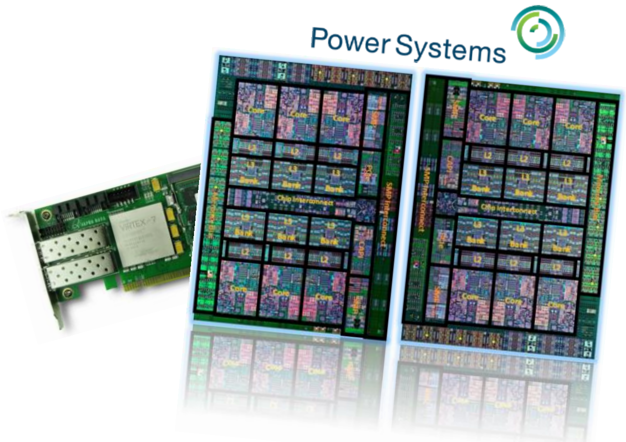
CAPI SNAP Education Series:
User Guide

!! Sometimes building image fails (timing issue)
=> Need code change to prevent this !!

CAPI SNAP Education
hls_hashjoin : howto?
V2.1

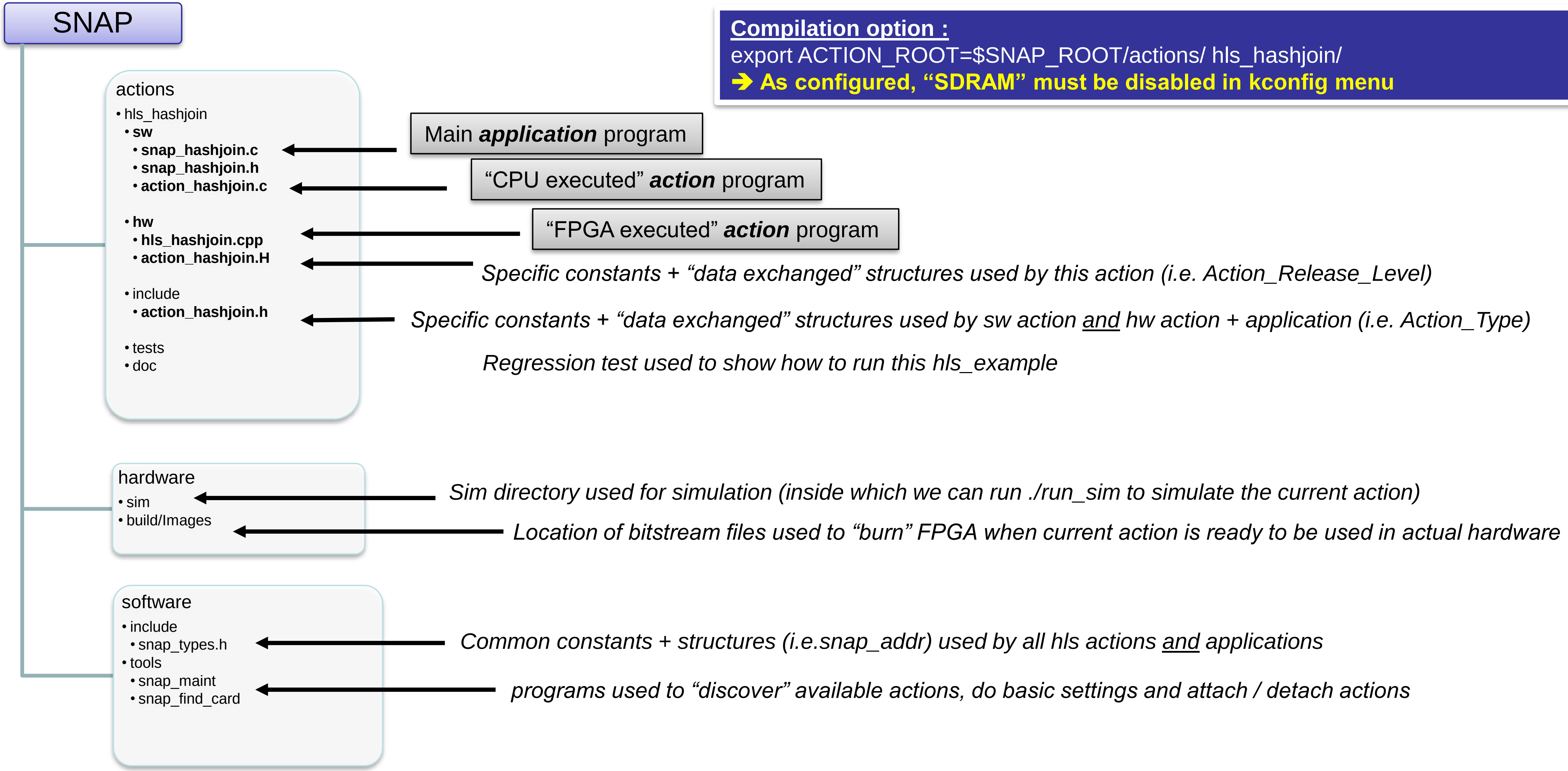


October 24th, 2017

SNAP Framework built on Power™ CAPI technology

Architecture of the SNAP git files

Compilation option :
export ACTION_ROOT=$SNAP_ROOT/actions/ hls_hashjoin/
➔ **As configured, “SDRAM” must be disabled in kconfig menu**



SNAP

actions
- hls_hashjoin
  - **sw**
    - **snap_hashjoin.c** ← Main ***application*** program
    - **snap_hashjoin.h**
    - **action_hashjoin.c** ← “CPU executed” ***action*** program
  - **hw**
    - **hls_hashjoin.cpp** ← “FPGA executed” ***action*** program
    - **action_hashjoin.H** ← *Specific constants + “data exchanged” structures used by this action (i.e. Action_Release_Level)*
  - include
    - **action_hashjoin.h** ← *Specific constants + “data exchanged” structures used by sw action <u>and</u> hw action + application (i.e. Action_Type)*
  - tests — *Regression test used to show how to run this hls_example*
  - doc

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 <u>and</u> applications*
- tools
  - snap_maint
  - snap_find_card ← *programs used to “discover” available actions, do basic settings and attach / detach actions*

Action overview

Purpose: Port a hashjoin function

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

When to use it:

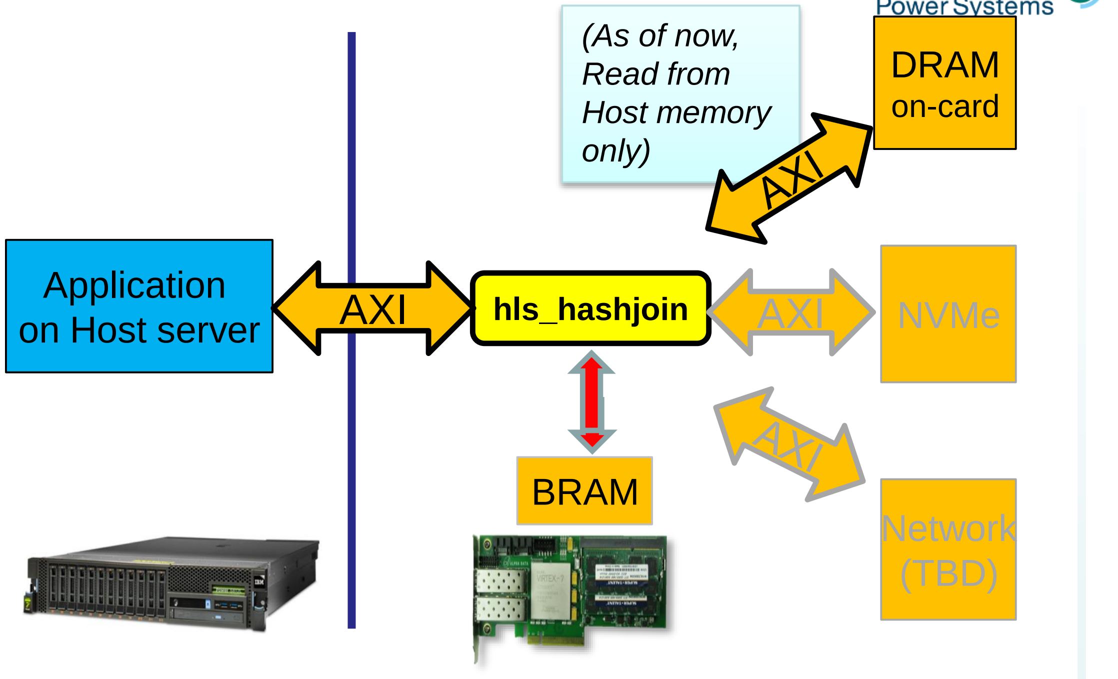
- Understand HLS constraints when working with large database

Memory management:

- All memory allocation is managed by the application

Known limitations:

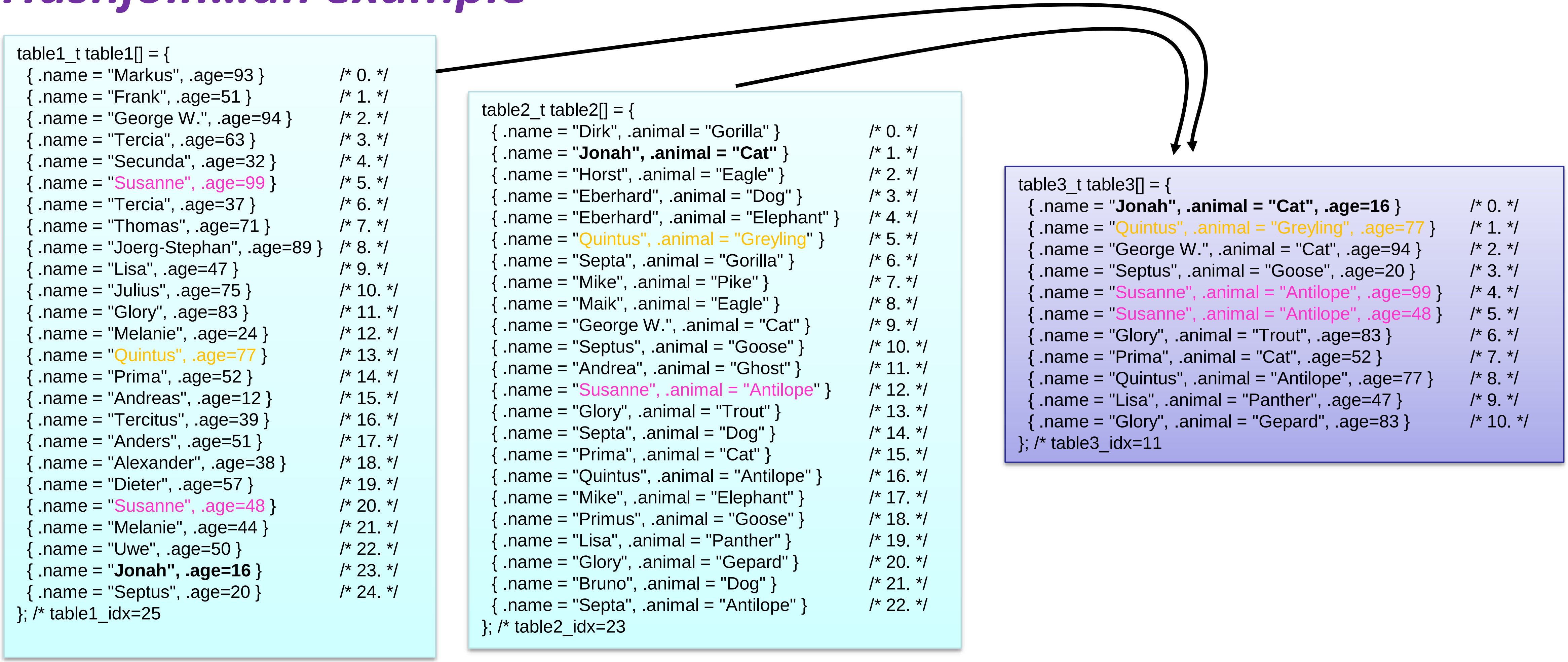
- All data are 64 bytes aligned to ease access
- Data taken from Host memory instead of DDR


(As of now, Read from Host memory only)
DRAM on-card
AXI
Application on Host server
AXI
hls_hashjoin
AXI
NVMe
BRAM
AXI
Network (TBD)

CAPI SNAP Enabled Card

```
typedef struct table1_s {
        hashkey_t name;         /* 64 bytes */
        uint32_t age;          /*  4 bytes */
        uint8_t reserved[60];   /* 60 bytes */
} table1_t;
```

Hashjoin…an example

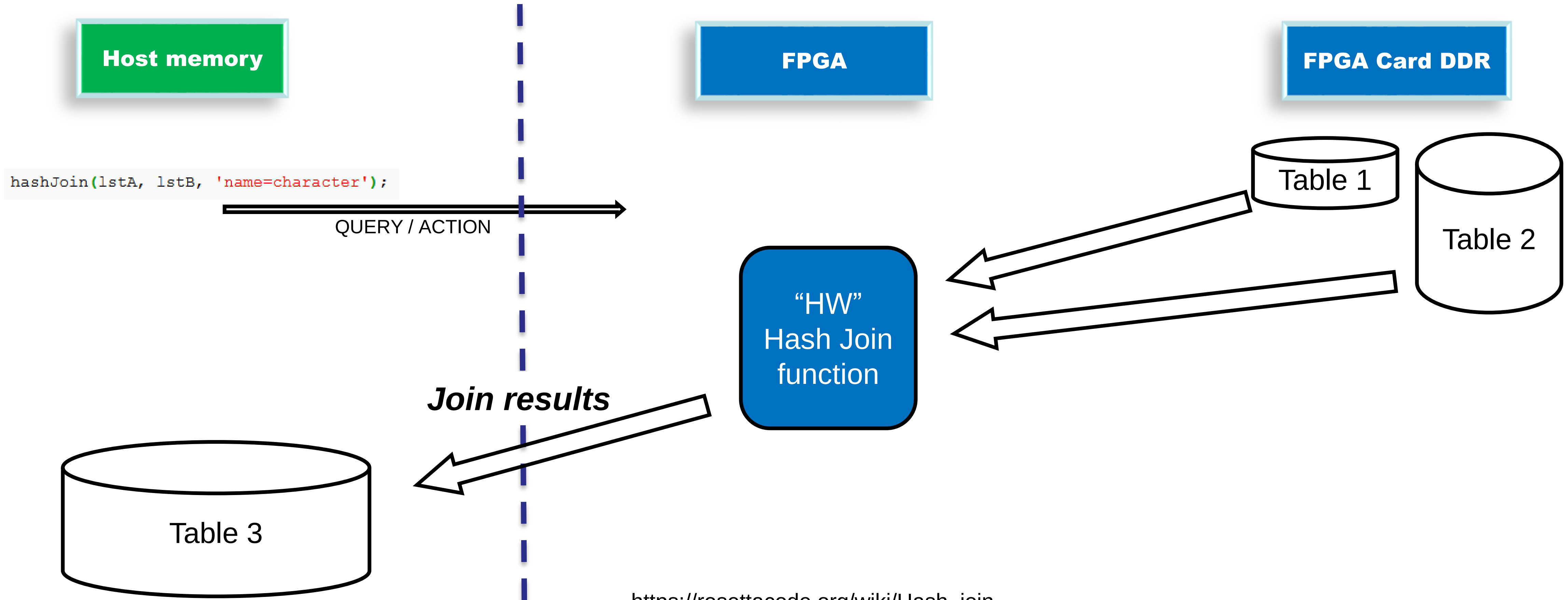


```
table1_t table1[] = {
  { .name = "Markus", .age=93 }        /* 0. */
  { .name = "Frank", .age=51 }         /* 1. */
  { .name = "George W.", .age=94 }     /* 2. */
  { .name = "Tercia", .age=63 }        /* 3. */
  { .name = "Secunda", .age=32 }       /* 4. */
  { .name = "Susanne", .age=99 }       /* 5. */
  { .name = "Tercia", .age=37 }        /* 6. */
  { .name = "Thomas", .age=71 }        /* 7. */
  { .name = "Joerg-Stephan", .age=89 } /* 8. */
  { .name = "Lisa", .age=47 }          /* 9. */
  { .name = "Julius", .age=75 }        /* 10. */
  { .name = "Glory", .age=83 }         /* 11. */
  { .name = "Melanie", .age=24 }       /* 12. */
  { .name = "Quintus", .age=77 }       /* 13. */
  { .name = "Prima", .age=52 }         /* 14. */
  { .name = "Andreas", .age=12 }       /* 15. */
  { .name = "Tercitus", .age=39 }      /* 16. */
  { .name = "Anders", .age=51 }        /* 17. */
  { .name = "Alexander", .age=38 }     /* 18. */
  { .name = "Dieter", .age=57 }        /* 19. */
  { .name = "Susanne", .age=48 }       /* 20. */
  { .name = "Melanie", .age=44 }       /* 21. */
  { .name = "Uwe", .age=50 }           /* 22. */
  { .name = "Jonah", .age=16 }         /* 23. */
  { .name = "Septus", .age=20 }        /* 24. */
}; /* table1_idx=25
```

```
table2_t table2[] = {
  { .name = "Dirk", .animal = "Gorilla" }        /* 0. */
  { .name = "Jonah", .animal = "Cat" }           /* 1. */
  { .name = "Horst", .animal = "Eagle" }         /* 2. */
  { .name = "Eberhard", .animal = "Dog" }        /* 3. */
  { .name = "Eberhard", .animal = "Elephant" }   /* 4. */
  { .name = "Quintus", .animal = "Greyling" }    /* 5. */
  { .name = "Septa", .animal = "Gorilla" }       /* 6. */
  { .name = "Mike", .animal = "Pike" }           /* 7. */
  { .name = "Maik", .animal = "Eagle" }          /* 8. */
  { .name = "George W.", .animal = "Cat" }       /* 9. */
  { .name = "Septus", .animal = "Goose" }        /* 10. */
  { .name = "Andrea", .animal = "Ghost" }        /* 11. */
  { .name = "Susanne", .animal = "Antilope" }    /* 12. */
  { .name = "Glory", .animal = "Trout" }         /* 13. */
  { .name = "Septa", .animal = "Dog" }           /* 14. */
  { .name = "Prima", .animal = "Cat" }           /* 15. */
  { .name = "Quintus", .animal = "Antilope" }    /* 16. */
  { .name = "Mike", .animal = "Elephant" }       /* 17. */
  { .name = "Primus", .animal = "Goose" }        /* 18. */
  { .name = "Lisa", .animal = "Panther" }        /* 19. */
  { .name = "Glory", .animal = "Gepard" }        /* 20. */
  { .name = "Bruno", .animal = "Dog" }           /* 21. */
  { .name = "Septa", .animal = "Antilope" }      /* 22. */
}; /* table2_idx=23
```

```
table3_t table3[] = {
  { .name = "Jonah", .animal = "Cat", .age=16 }            /* 0. */
  { .name = "Quintus", .animal = "Greyling", .age=77 }     /* 1. */
  { .name = "George W.", .animal = "Cat", .age=94 }        /* 2. */
  { .name = "Septus", .animal = "Goose", .age=20 }         /* 3. */
  { .name = "Susanne", .animal = "Antilope", .age=99 }     /* 4. */
  { .name = "Susanne", .animal = "Antilope", .age=48 }     /* 5. */
  { .name = "Glory", .animal = "Trout", .age=83 }          /* 6. */
  { .name = "Prima", .animal = "Cat", .age=52 }            /* 7. */
  { .name = "Quintus", .animal = "Antilope", .age=77 }     /* 8. */
  { .name = "Lisa", .animal = "Panther", .age=47 }         /* 9. */
  { .name = "Glory", .animal = "Gepard", .age=83 }         /* 10. */
}; /* table3_idx=11
```

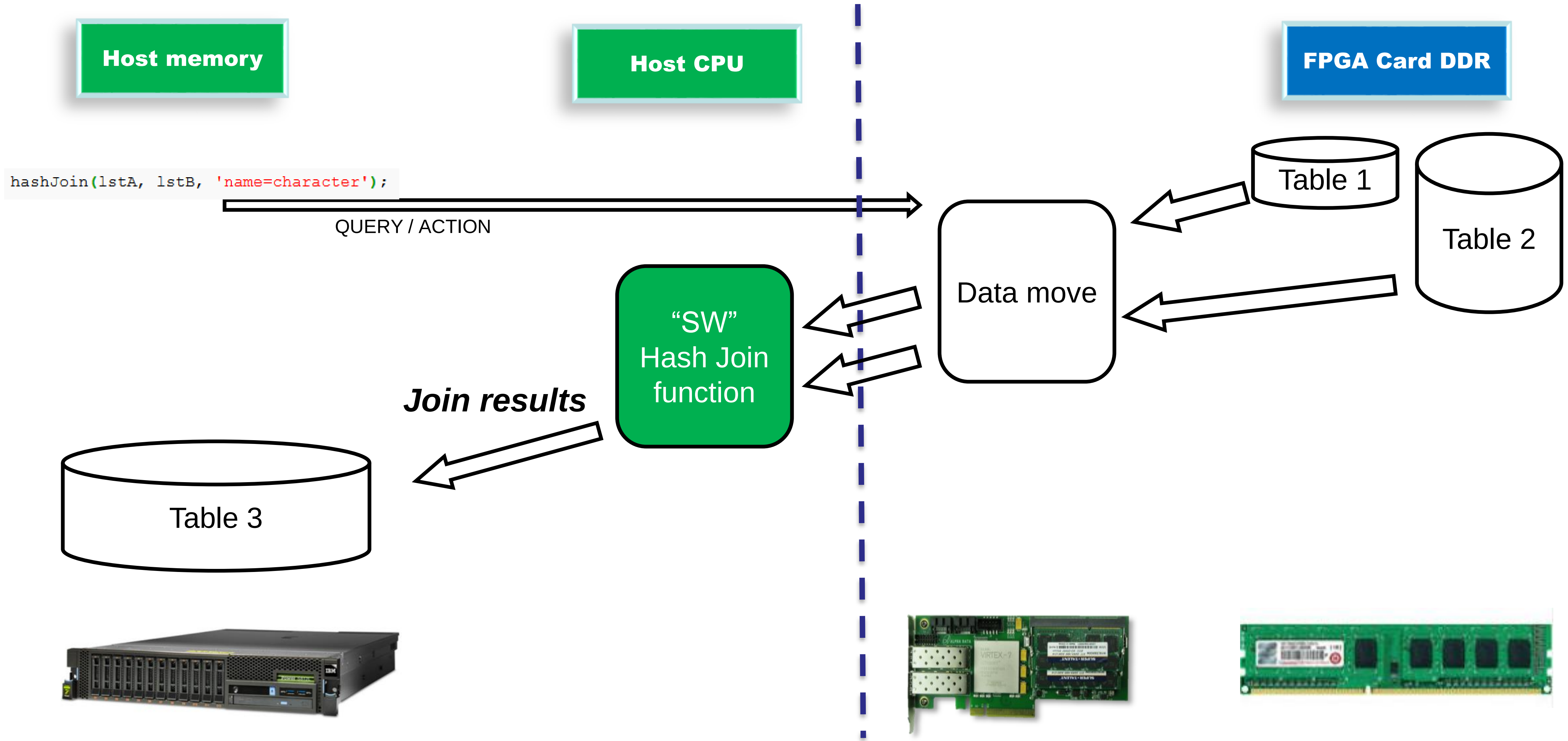
Hash Join : data are in card DDR - processing done in FPGA



```
hashJoin(lstA, lstB, 'name=character');
```

https://rosettacode.org/wiki/Hash_join

Hash Join : data are in card DDR - processing done in CPU



```
hashJoin(lstA, lstB, 'name=character');
```

Action usage

Usage: `./snap_hashjoin -usage`

```
Usage: ./snap_hashjoin [-h] [-v, --verbose] [-V, --version]
  -C, --card <cardno> can be (0...3)
  -t, --timeout <timeout>  Timefor for job completion. (default 10 sec)
  -Q, --t1-entries <items> Entries in table1. (maximum TABLE1_SIZE defined in snap/software/examples/action_hashjoin.h)
  -T, --t2-entries <items> Entries in table2.
  -s, --seed <seed>        Random seed to enable recreation.
  -N, --no irq             Disable IRQs
```

Options:

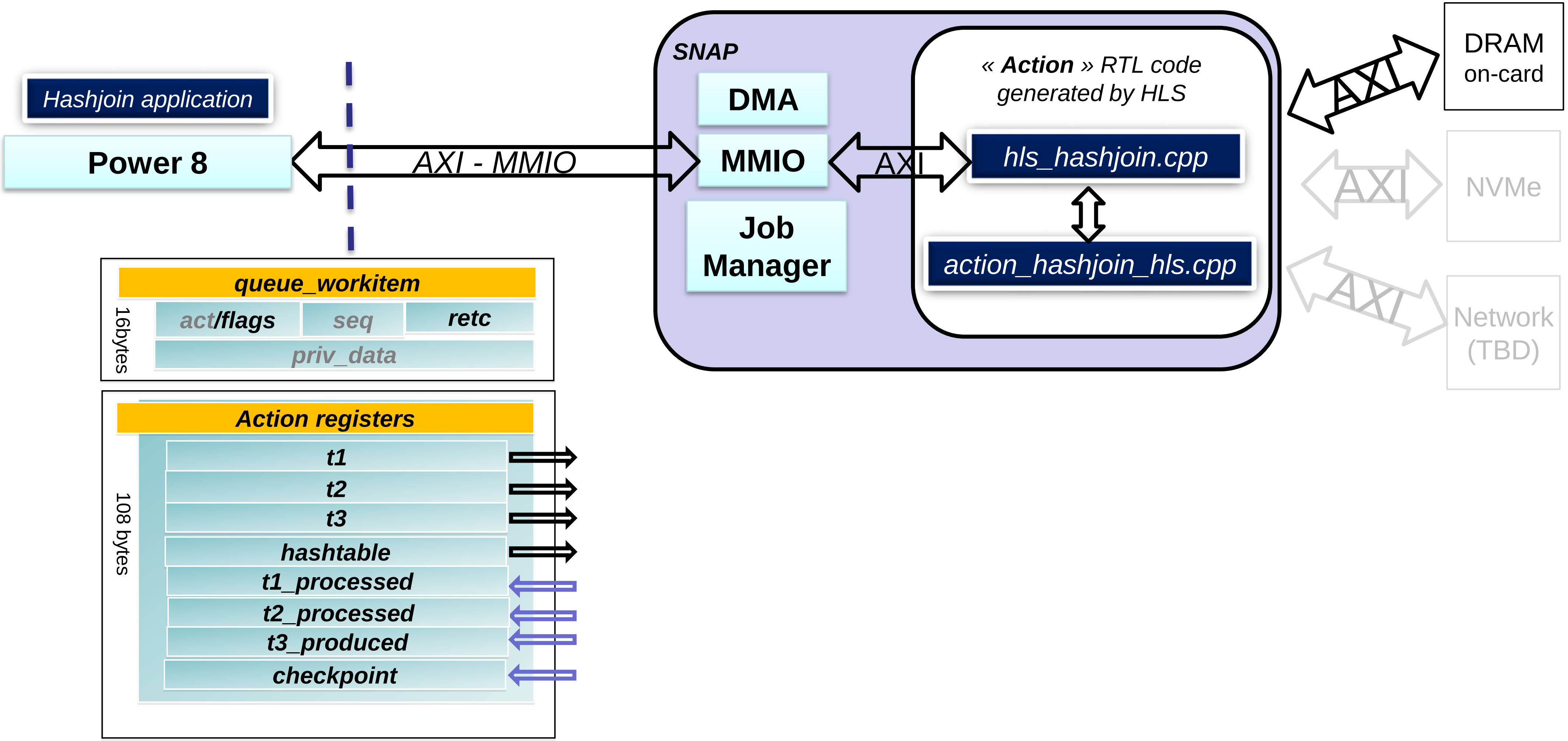
```
 SNAP_TRACE = 0x0 → no debug trace
 SNAP_TRACE = 0xF → full debug trace
SNAP_CONFIG = FPGA→ hardware execution
SNAP_CONFIG = CPU → software execution
```

Example :

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

#echo Random generation of 2 tables with default table size (T1 = 25 entries / T2 = 23 entries)
SNAP_CONFIG=CPU ./snap_hashjoin -C1 -vv -t2500
#echo Random generation of 2 tables with 30 entries for T1 and 60 for T2 => action will call 2 times the action
SNAP_CONFIG=CPU ./snap_hashjoin -C1 -vv -t2500 -Q 30 -T 60
```

Hashjoin registers



Hashjoin application

Power 8

AXI - MMIO

SNAP

DMA

MMIO

Job Manager

AXI

« **Action** » RTL code generated by HLS

hls_hashjoin.cpp

action_hashjoin_hls.cpp

AXI

DRAM on-card

AXI

NVMe

AXI

Network (TBD)

queue_workitem

16bytes

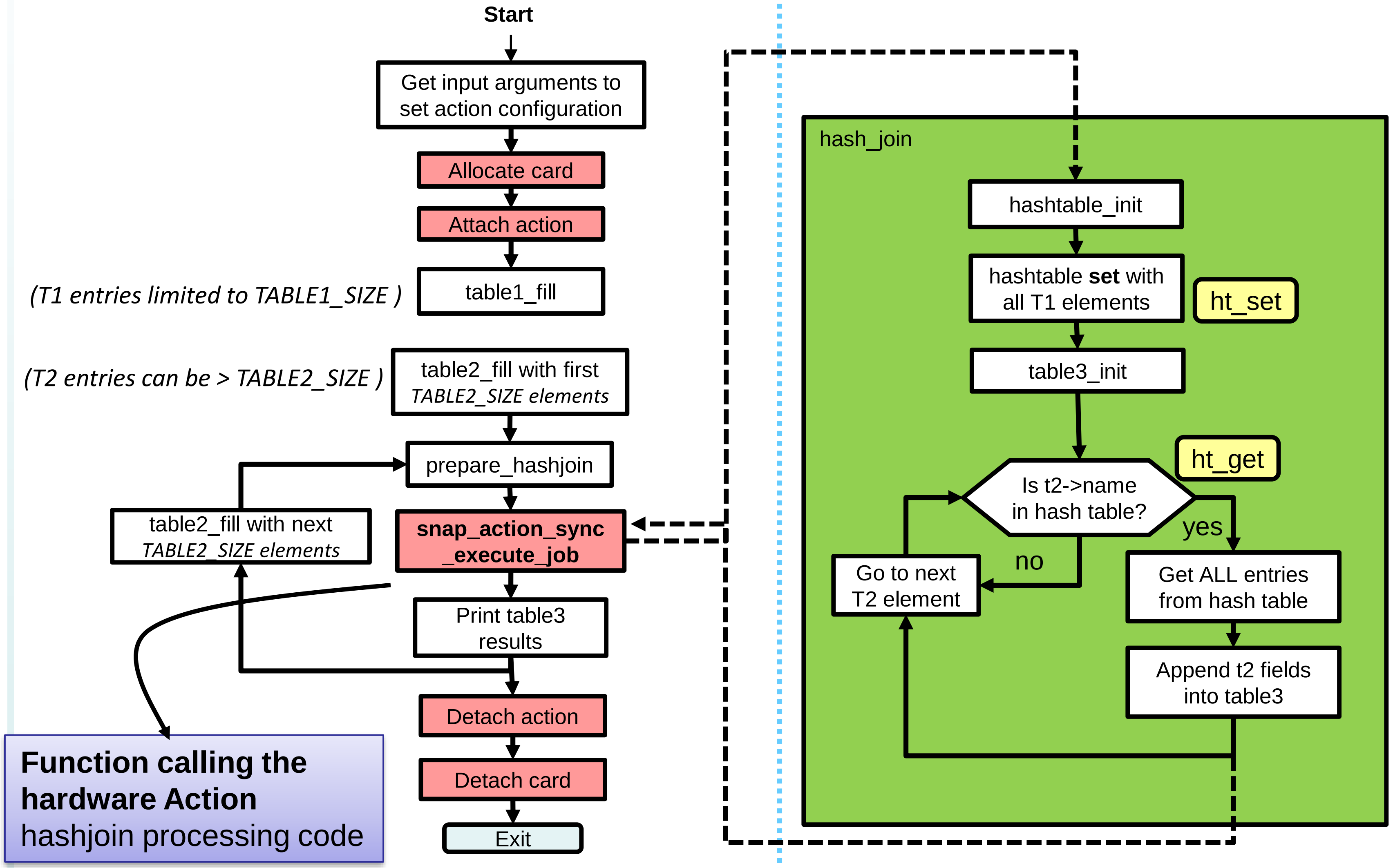
act/flags	seq	retc
priv_data		

Action registers

108 bytes

- t1
- t2
- t3
- hashtable
- t1_processed
- t2_processed
- t3_produced
- checkpoint

Application Code : what’s in it?


Start
Get input arguments to set action configuration
Allocate card
Attach action
(T1 entries limited to TABLE1_SIZE)
table1_fill
(T2 entries can be > TABLE2_SIZE)
table2_fill with first *TABLE2_SIZE elements*
prepare_hashjoin
table2_fill with next *TABLE2_SIZE elements*
snap_action_sync _execute_job
Print table3 results
Detach action
Detach card
Exit
hash_join
hashtable_init
hashtable **set** with all T1 elements
ht_set
table3_init
ht_get
Is t2->name in hash table?
yes
no
Go to next T2 element
Get ALL entries from hash table
Append t2 fields into table3

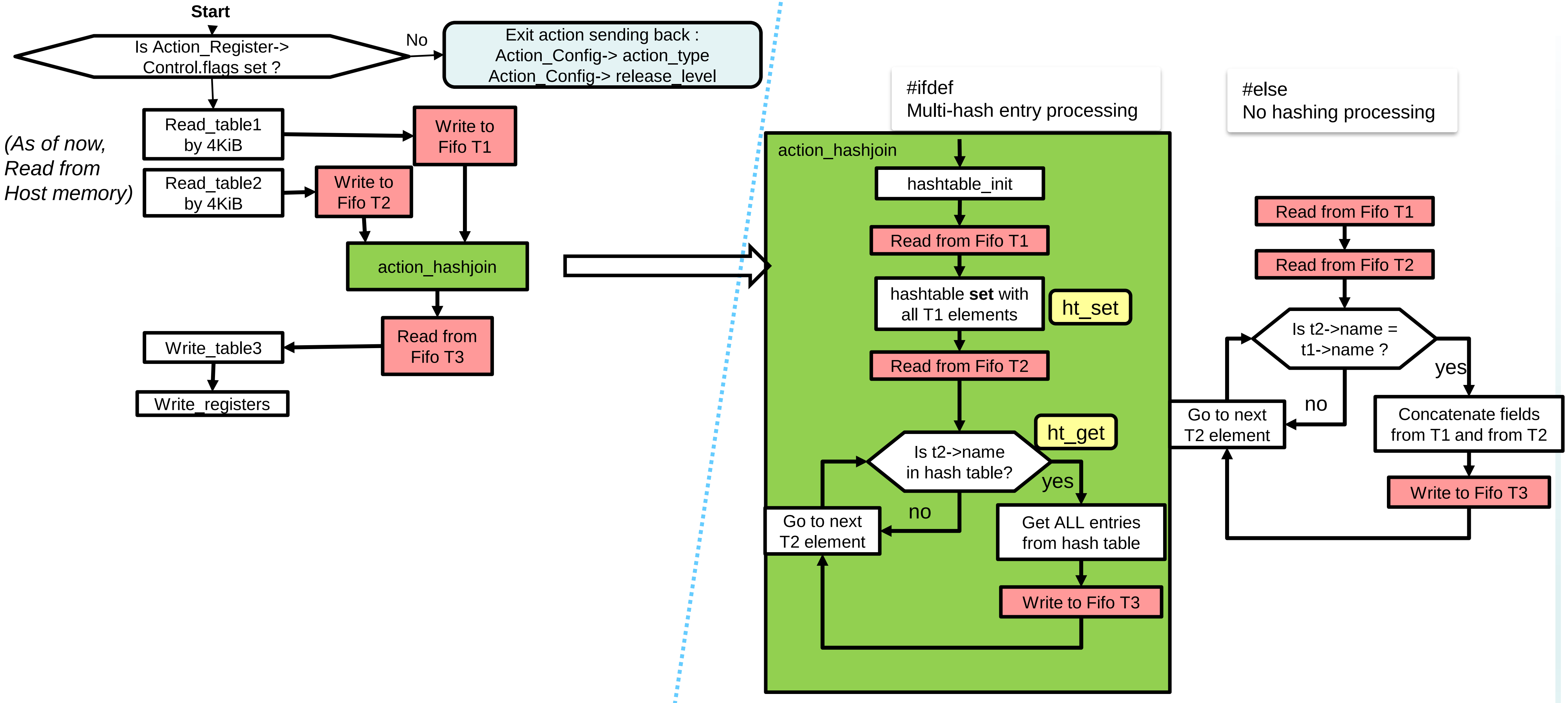
Function calling the hardware Action hashjoin processing code

Function calling the software hashjoin processing code (application sw code # action code)

Application: snap_hashjoin.c

Simulation: action_hashjoin.c

Action hashjoin Code : what’s in it?



Start

Is Action_Register-> Control.flags set ?

No

Exit action sending back :
Action_Config-> action_type
Action_Config-> release_level

(As of now, Read from Host memory)

Read_table1 by 4KiB

Write to Fifo T1

Read_table2 by 4KiB

Write to Fifo T2

action_hashjoin

Read from Fifo T3

Write_table3

Write_registers

#ifdef
Multi-hash entry processing

action_hashjoin

hashtable_init

Read from Fifo T1

hashtable **set** with all T1 elements

ht_set

Read from Fifo T2

ht_get

Is t2->name in hash table?

yes

no

Go to next T2 element

Get ALL entries from hash table

Write to Fifo T3

#else
No hashing processing

Read from Fifo T1

Read from Fifo T2

Is t2->name = t1->name ?

yes

no

Go to next T2 element

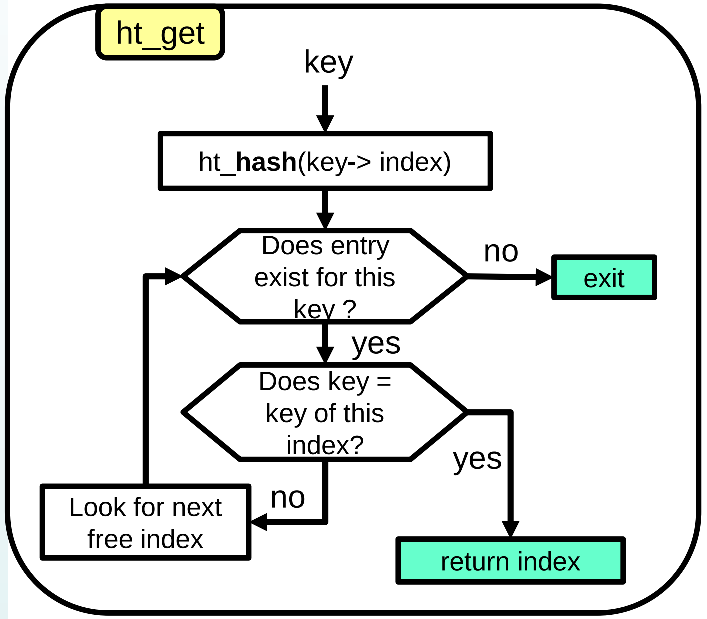
Concatenate fields from T1 and from T2

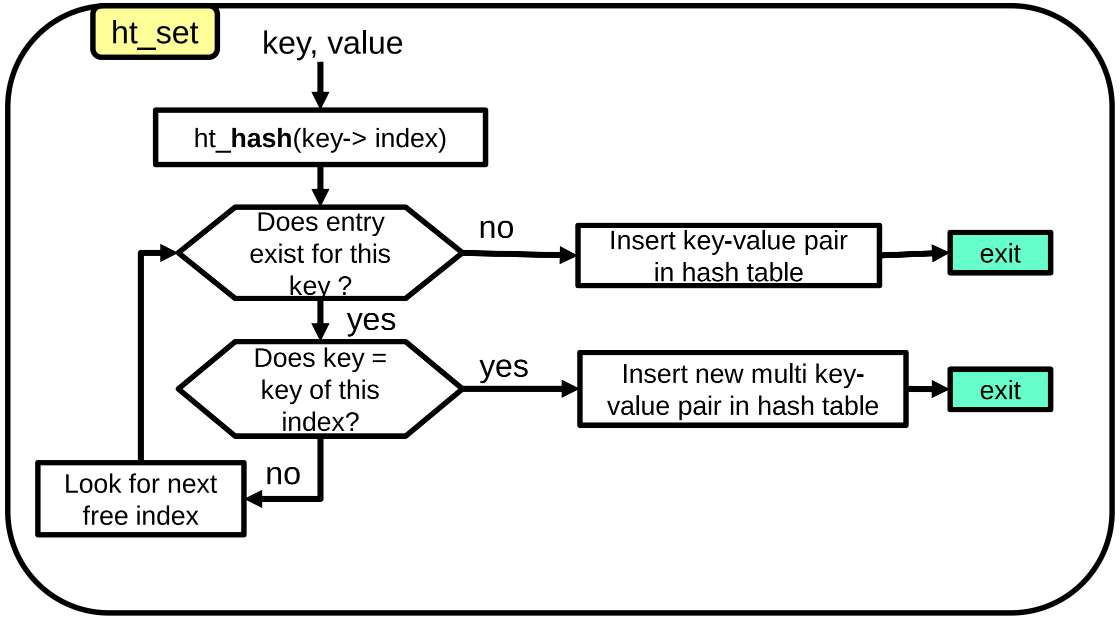
Write to Fifo T3

Application: snap_hashjoin.c

Action: action_hashjoin.cpp

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




ht_set
key, value
ht_**hash**(key-> index)
Does entry exist for this key ?
no
Insert key-value pair in hash table
exit
yes
Does key = key of this index?
yes
Insert new multi key-value pair in hash table
exit
no
Look for next free index

Application : snap_hashjoin.c / **Action**: action_hashjoin.cpp

Constants - Ports

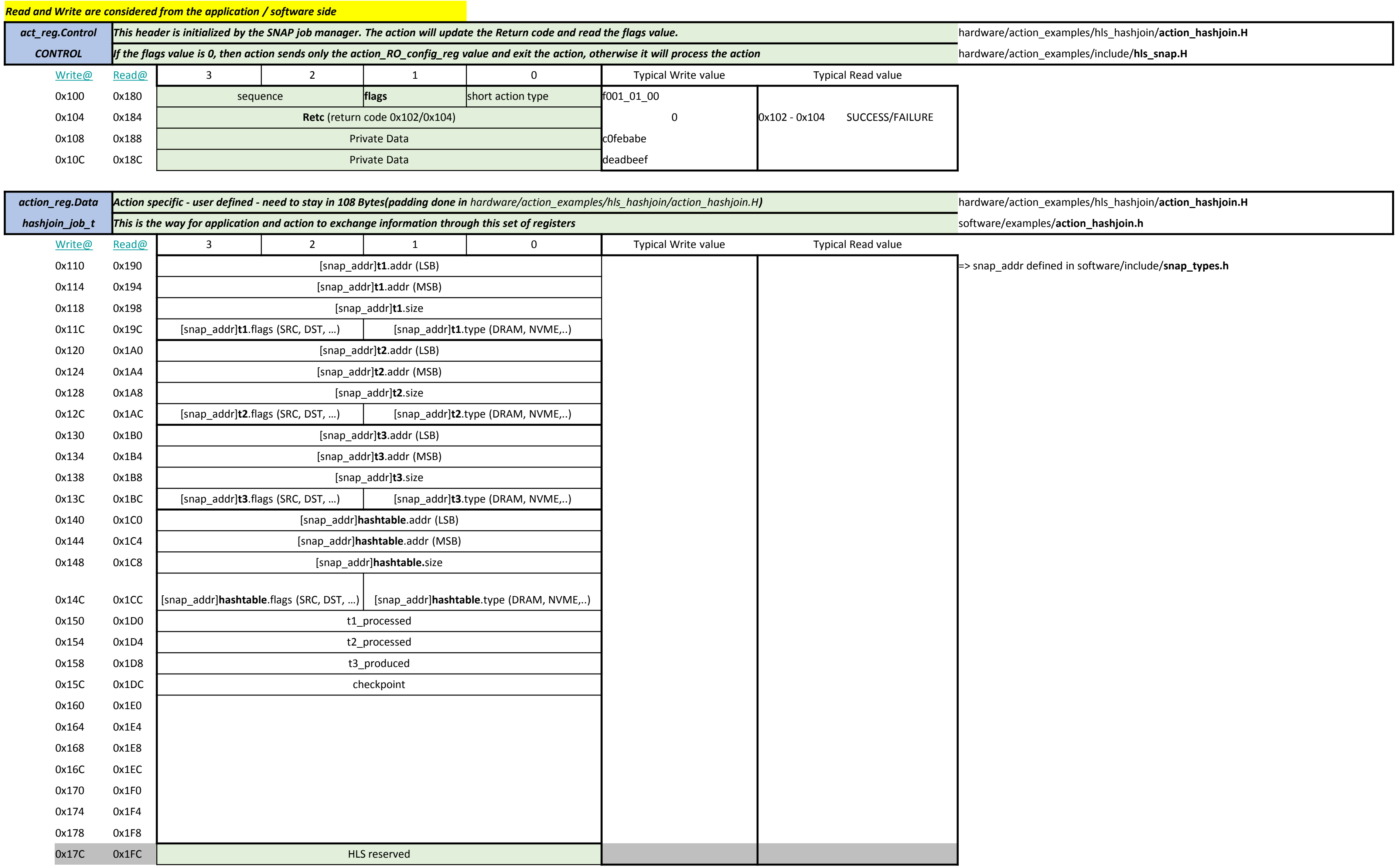
Constants: ➔ $ACTION_ROOT = snap/actions/hls_hashjoin

Constant name	Value	Type	Definition location	Usage
HASHJOIN_ACTION_TYPE	0x10141002	Fixed	$ACTION_ROOT/include/action_hashjoin.h	Checksum ID - list is in snap/ActionTypes.md
RELEASE_LEVEL	0x00000021	Variable	$ACTION_ROOT/hw/action_hashjoin.**H**	release level – user defined
TABLE1_SIZE	32	Variable	$ACTION_ROOT/hw/action_hashjoin.**H**	Maximum number of entries for Table1
TABLE2_SIZE	32	Variable	$ACTION_ROOT/hw/action_hashjoin.**H**	Maximum size of the Table 2 for the hardware action, but entries can be from any size. Multiple calls to Action will return table3 results
TABLE3_SIZE	(TABLE1_SIZE * TABLE2_SIZE)	*Operation*	$ACTION_ROOT/hw/action_hashjoin.**H**	Table 3 will be from any size
HT_SIZE	(TABLE1_SIZE * 16)	*Operation*	$ACTION_ROOT/hw/action_hashjoin.**H**	Definition of the size of hashtable
HT_MULTI	(TABLE1_SIZE)	*Operation*	$ACTION_ROOT/hw/action_hashjoin.**H**	multihash entries depends on table1

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)*

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_hashjoin.H**
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*						hardware/action_examples/include/**hls_snap.H**
Write@	Read@	3	2	1	0	Typical Write value	Typical Read value	
0x100	0x180	sequence		**flags**	short action type	f001_01_00		
0x104	0x184	**Retc** (return code 0x102/0x104)				0	0x102 - 0x104 SUCCESS/FAILURE	
0x108	0x188	Private Data				c0febabe		
0x10C	0x18C	Private Data				deadbeef		

action_reg.Data		*Action specific - user defined - need to stay in 108 Bytes(padding done in* hardware/action_examples/hls_hashjoin/action_hashjoin.H*)*						hardware/action_examples/hls_hashjoin/**action_hashjoin.H**
hashjoin_job_t		*This is the way for application and action to exchange information through this set of registers*						software/examples/**action_hashjoin.h**
Write@	Read@	3	2	1	0	Typical Write value	Typical Read value	
0x110	0x190	[snap_addr]**t1**.addr (LSB)						=> snap_addr defined in software/include/**snap_types.h**
0x114	0x194	[snap_addr]**t1**.addr (MSB)						
0x118	0x198	[snap_addr]**t1**.size						
0x11C	0x19C	[snap_addr]**t1**.flags (SRC, DST, …)		[snap_addr]**t1**.type (DRAM, NVME,..)				
0x120	0x1A0	[snap_addr]**t2**.addr (LSB)						
0x124	0x1A4	[snap_addr]**t2**.addr (MSB)						
0x128	0x1A8	[snap_addr]**t2**.size						
0x12C	0x1AC	[snap_addr]**t2**.flags (SRC, DST, …)		[snap_addr]**t2**.type (DRAM, NVME,..)				
0x130	0x1B0	[snap_addr]**t3**.addr (LSB)						
0x134	0x1B4	[snap_addr]**t3**.addr (MSB)						
0x138	0x1B8	[snap_addr]**t3**.size						
0x13C	0x1BC	[snap_addr]**t3**.flags (SRC, DST, …)		[snap_addr]**t3**.type (DRAM, NVME,..)				
0x140	0x1C0	[snap_addr]**hashtable**.addr (LSB)						
0x144	0x1C4	[snap_addr]**hashtable**.addr (MSB)						
0x148	0x1C8	[snap_addr]**hashtable**.size						
0x14C	0x1CC	[snap_addr]**hashtable**.flags (SRC, DST, …)		[snap_addr]**hashtable**.type (DRAM, NVME,..)				
0x150	0x1D0	t1_processed						
0x154	0x1D4	t2_processed						
0x158	0x1D8	t3_produced						
0x15C	0x1DC	checkpoint						
0x160	0x1E0							
0x164	0x1E4							
0x168	0x1E8							
0x16C	0x1EC							
0x170	0x1F0							
0x174	0x1F4							
0x178	0x1F8							
0x17C	0x1FC	HLS reserved						

Measured performance

Times are in ***µs***			**"SW" hashjoin process (on CPU)**			**"HW" hashjoin process (on FPGA)**						
2x64B	2x64B						***1/2 of T2***	3x64B			***1/2 of T2***	***1/4 of T2***
T1 (entries)	T2 (entries)	T1+T2 Size(Bytes)	**DDR to Host**	**SW**	**Total**	**HW**	**T3 (entries)**	**Size in Bytes**	**DDR to Host**	**Total**	Avg Speed up	Avg Speed up
30	50	10,240	7,314	511	7,825	241	25	4,800	3,429	3,670	2.1	4.1
30	500	67,840	48,457	1,014	49,471	1,318	250	48,000	34,286	35,604	1.4	2.7
30	5,000	643,840	459,886	5,132	465,018	12,222	2,500	480,000	342,857	355,079	1.3	2.5
30	50,000	6,403,840	4,574,171	34,493	4,608,664	114,813	25,000	4,800,000	3,428,571	3,543,384	1.3	2.5

For this performance measurement, we considered :

- 2 tables in entries containing each 2 x 64Bytes fields
- 1 table for results containing 3 x 64Bytes fields
- ***We considered that results are ½ of the number of the entries of the largest table (realistic ?)***

Comment : No real optimization work was done on this example since hash logic may need to be modified to build the image an easier way

What else ?

Path of improvement ?

1. Find how write the algorithm so that HLS can parallelize :
   - Building hashtable with new elements from table 1 (small table)
   - Checking / adding new elements from table 2 (large table)
   - Filling result table 3

   ➔ This mean handling collision
2. Enable conversion/type-casting of flat memory to structures
3. Support unaligned data access e.g. special FIFOs
4. Pointers, dynamic memory allocation, ...

Backup slides

Experiences during implementation

CAPI should allow direct access to host-memory. But ...

1. Simple conversion (e.g. casting) from void */uint8_t * to struct table_t *data do not work
   ***Circumvention**: manual data conversion*

```
static void read_table1(snap_membus_t *mem, unsigned int max_lines,
            t1_fifo_t *fifo1, uint32_t t1_used)
{
      unsigned int i;
      snap_4KiB_t buf;

      snap_4KiB_rinit(&buf, mem, max_lines);
read_table1_loop:
      for (i = 0; i < t1_used; i++) {
#pragma HLS PIPELINE
            snap_membus_t b[2];
            table1_t t1;

            snap_4KiB_get(&buf, &b[0]);
            copy_hashkey(b[0], t1.name);
            snap_4KiB_get(&buf, &b[1]);
            t1.age = b[1](31, 0);
            fifo1->write(t1)
      }
}
```

Would like to have something alike:

```
      table1_t t1[16];
      memcopy(&t1, (hmem_axi_bus + t1_offs), sizeof(t1));

Or even better:

      table1_t *t1 = (table1_t *)(hmem_axi_bus + t1_addr);
```

... more

2. SNAP 512 bit fixed width bus causes code complexity when handling unaligned data – currently no helper support
   ***Circumvention**: align and pad data to 512 bit boundaries to keep code simple, transfer more data than needed if code complexity should be low*
   ***Future**: Maybe the AXI_to_table FIFO?*
   ***Impact :** More data-movement complexity to avoid transferring additional data, no possibility in HLS to write preceding bytes (we do not own those)*

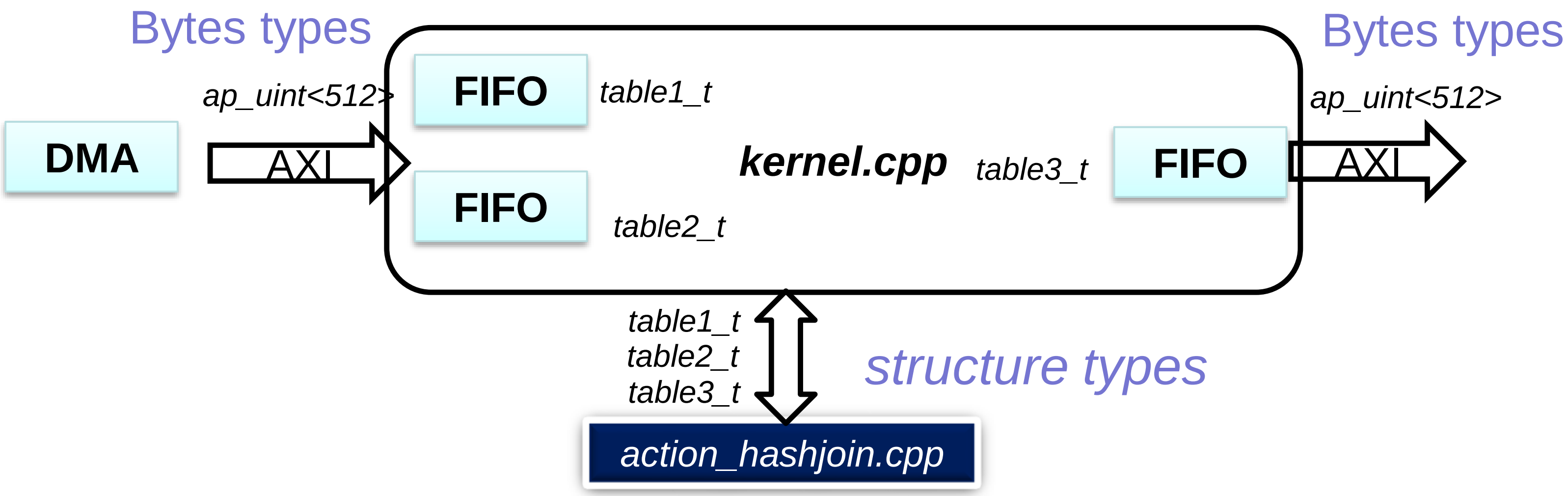
```
typedef struct table1_s {
        hashkey_t name;             /* 64 bytes */
        unsigned int age;           /*  4 bytes */
        unsigned char reserved[60]; /* 60 bytes */
} table1_t;

typedef struct table2_s {
        hashkey_t name;             /* 64 bytes */
        hashkey_t animal;           /* 64 bytes */
} table2_t;

typedef struct table3_s {
        hashkey_t animal;           /* 64 bytes */
        hashkey_t name;             /* 64 bytes */
        unsigned int age;           /*  4 bytes */
        unsigned char reserved[60]; /* 60 bytes */
} table3_t;
```

Suggestions to solve items 1 & 2 : use FIFO to cast types

- Connect a FIFO between the AXI bus and the action_hashjoin function ➔ DONE
- Build with MIG / Memory Interface Generator a FIFO which has following characteristics: ➔ EXIST
  - Input data bus width is fixed to 512 bits/64B
  - Output data bus width will be adapted to the action (e.g. 32B/64B/128Bytes)
  - Depth of FIFO can absorb the 4KiB required by PSL access
- Call this FIFO in C program such that it can be integrated by HLS ➔ **MISSING**
- Use FIFO to do type casting : ap_uint in input and structure in output ➔ **MISSING**



… more

3. Complex pointer operations are not possible, e.g. dynamic memory management, pointer lists, etc.
   Circumvention: Rewrite the code

   Maybe: Special allocators for fixed size objects? E.g. in external onboard DDR memory. Simplistic cache?

History of this document and of the action release level

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