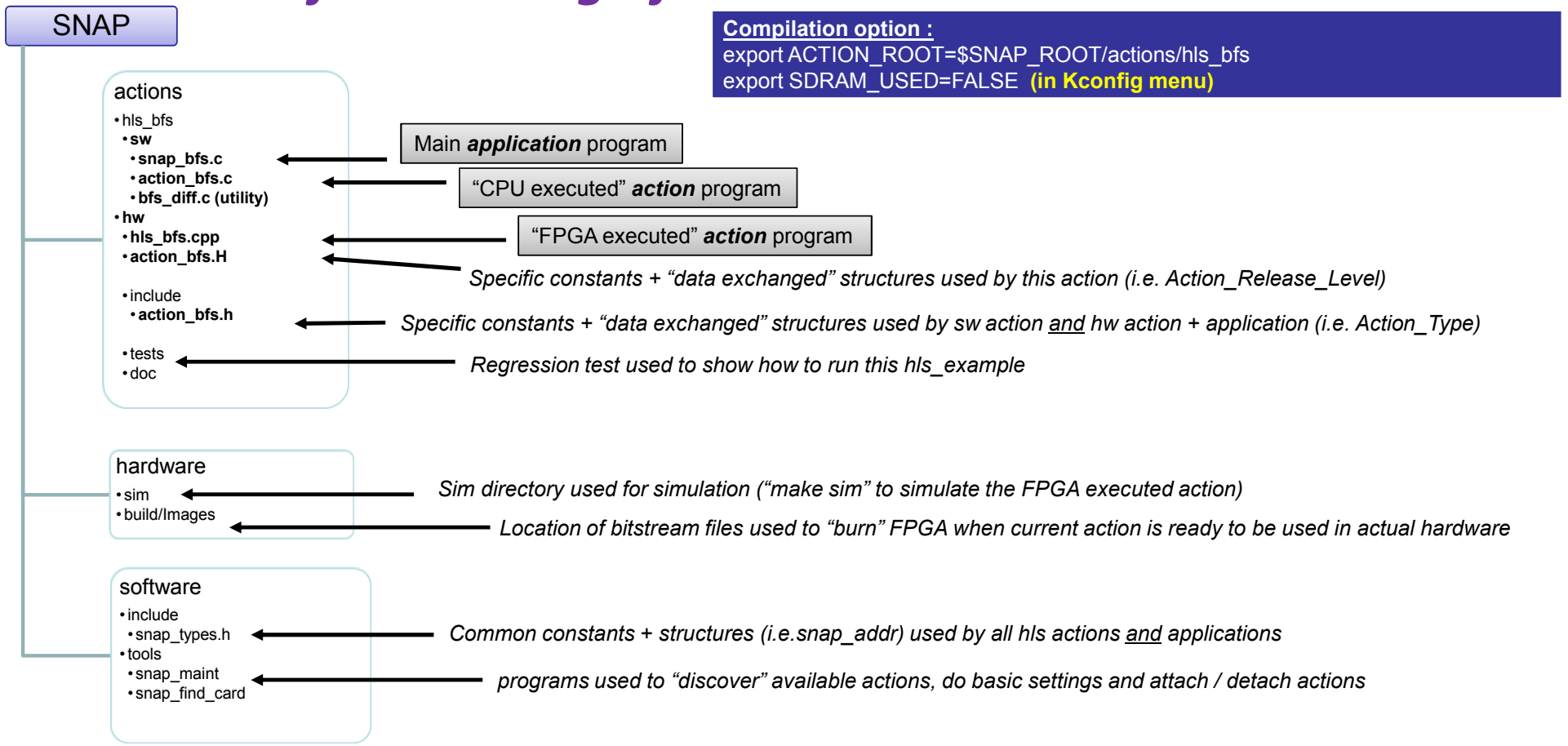
CAPI SNAP Education Series:
User Guide

CAPI SNAP Education
hls_bfs : howto?
V1.0

SNAP Framework built on Power™ CAPI technology

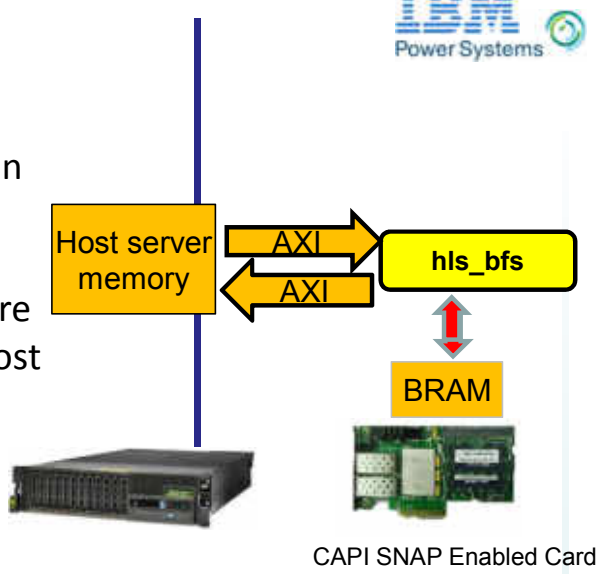
Architecture of the SNAP git files



Action overview

Purpose: BFS: breadth first search example

- Given a directed graph, BFS is a basic algorithm to traverse all of the nodes in this graph. It is one of the most fundamental operations for graph based databases.
- In traditional PCIe based FPGA acceleration, such kind of graph data structure is hard to handle. The graph nodes are stored with pointers to pointers in host memory, and CAPI unleashed an easy and nature way to access them.



When to use it:

- Understand how to access complex data structure in the host memory

Memory management:

- Application is constructing the graph structure and tells Action the head pointers.
- Action uses the head pointers to complete the node traversing in the graph.
- No local DDR used.

Known limitations:

- HLS requires transfers to be 64 byte aligned and a size of multiples of 64 bytes, and that's why the data structure definition has to consider about it.

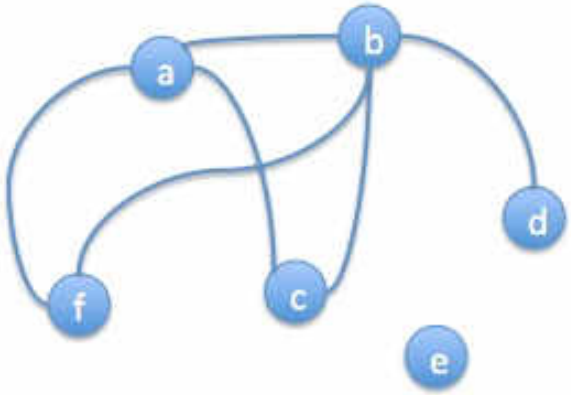
BFS Data Structure Example

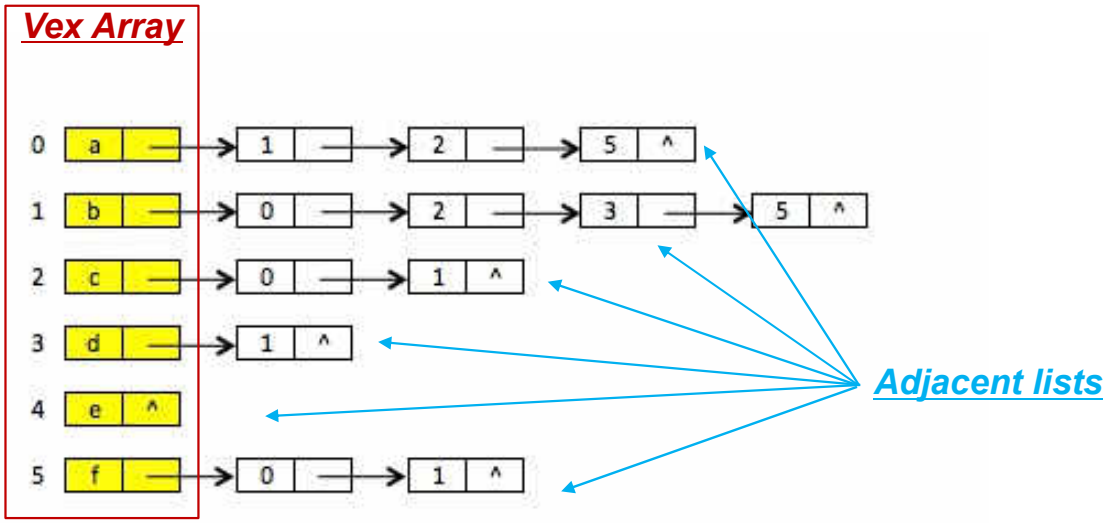
"Adjacent List"

Given V vertices, use an array to store them (Yellow ones)

And Each vertex (source vertex) has an adjacent list pointer. The adjacent list contains all of the edges connecting with the source vertex, and each edge (white ones) stores the target vertex index and a pointer to the next edge.

https://en.wikipedia.org/wiki/Breadth-first_search





Action usage

Usage:

```
./snap_bfs [-h] [-v, --verbose] [-V, --version]
        -C, --card <cardno> can be (0...3)
        -i, --input_file <graph.txt>       Input graph file. (Not Available Now!!!)
        -o, --output_file <traverse.bin>   Output traverse result file.
        -t, --timeout <seconds>       When graph is large, need to enlarge it.
        -r, --rand_nodes <N>          Generate a random graph with the number
        -s, --start_root <num>        Traverse starting node index [0...N-1], default 0
        -v, --verbose                 Show more information on screen.
                                      Automatically turned off when vex number > 20
        -V, --version                 Git version
        -I, --irq                     Enable Interrupts
```

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

SNAP_TRACE = 0x0 ➔ no debug trace
SNAP_TRACE = 0xF ➔ full debug trace

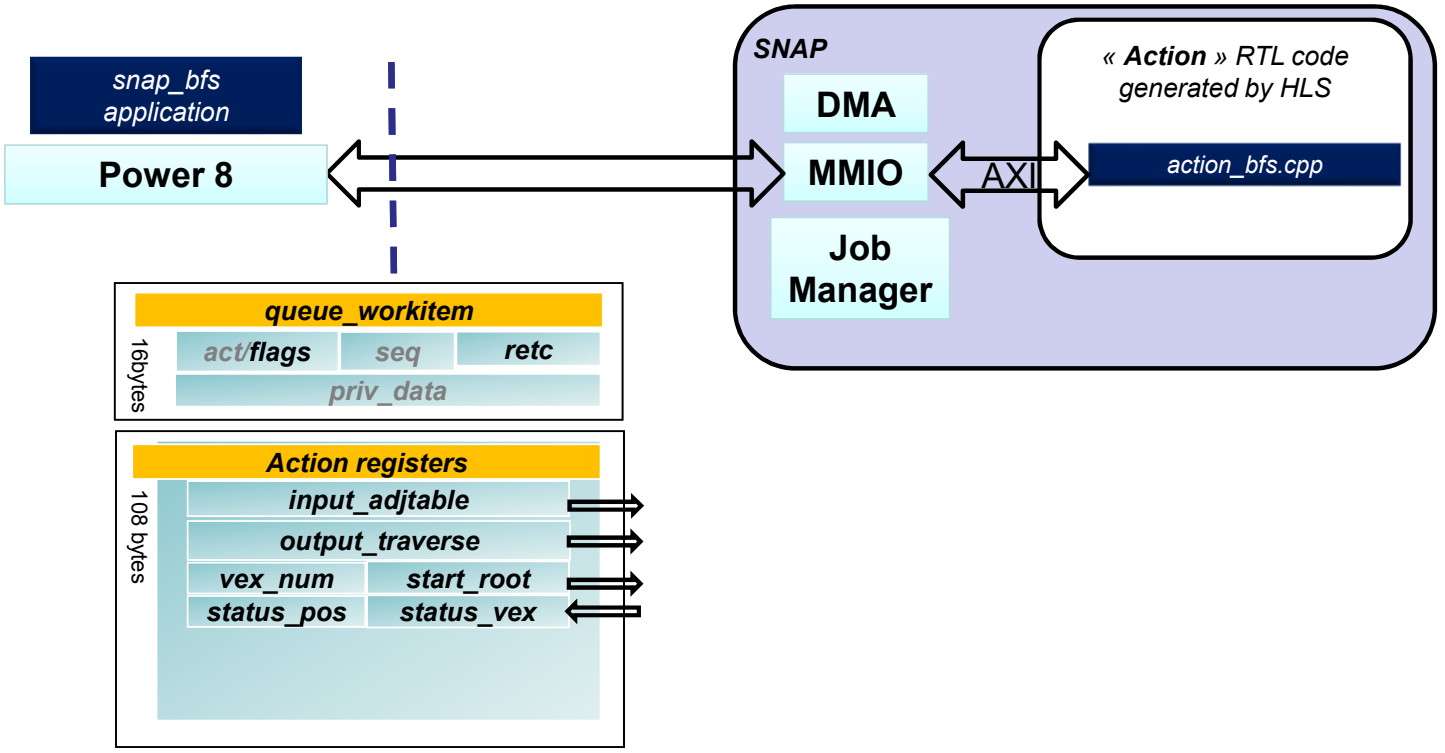
SNAP_CONFIG = FPGA ➔ hardware execution
SNAP_CONFIG = CPU ➔ software execution

Example :

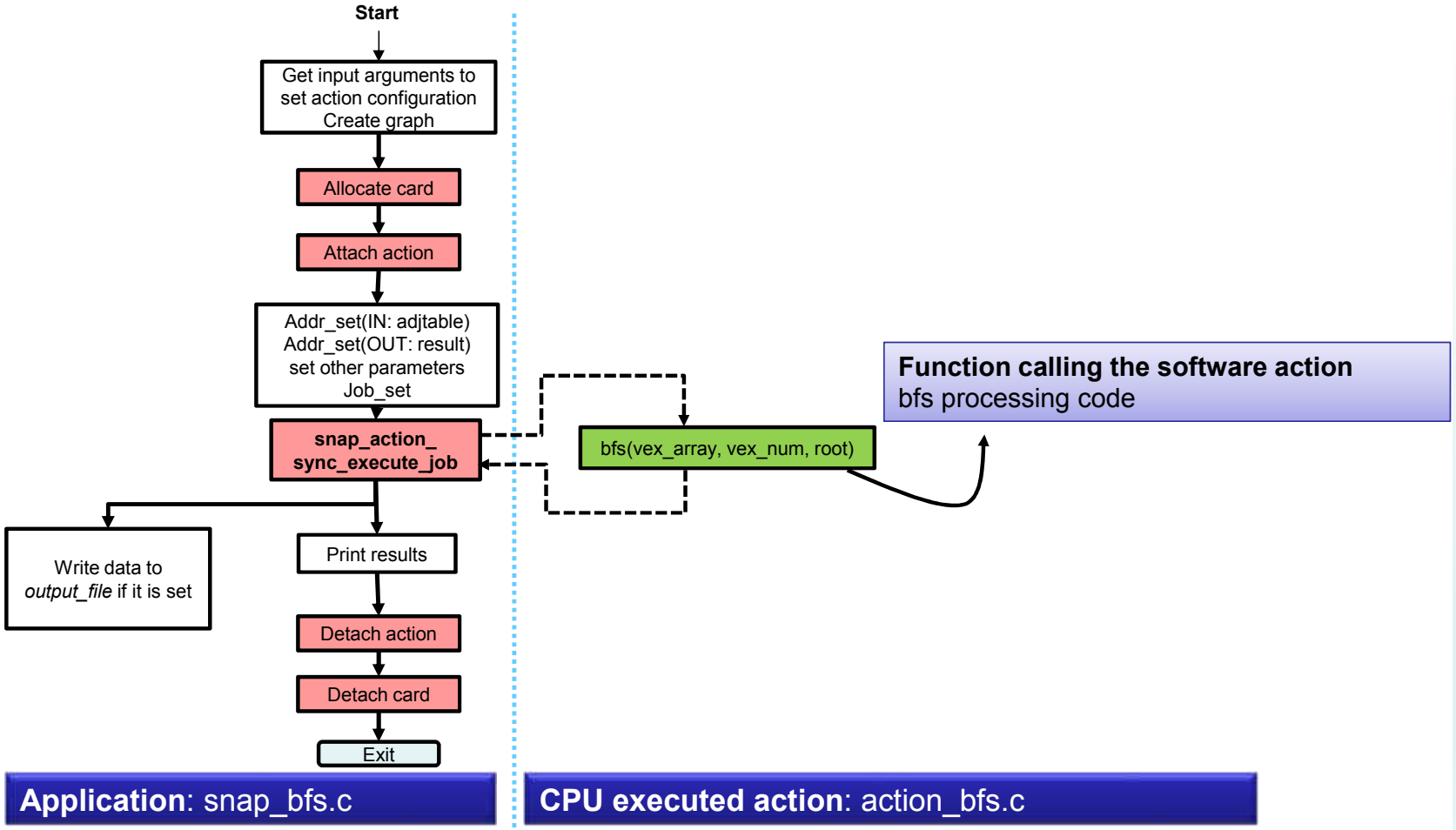
```
export SNAP_TRACE=0x0
snap_maint -vv
snap_bfs   (Traverse a small sample graph and show result on screen)

SNAP_CONFIG=FPGA snap_bfs -r 50 -s 9 -o traverse.hw.out
        (Generate a 50 nodes graph, traverse from node 9, and output to a file)
SNAP_CONFIG=CPU snap_bfs -r 50 -s 9 -o traverse.sw.out
bfs_diff   traverse.hw.out traverse.sw.out
```

hls_bfs registers



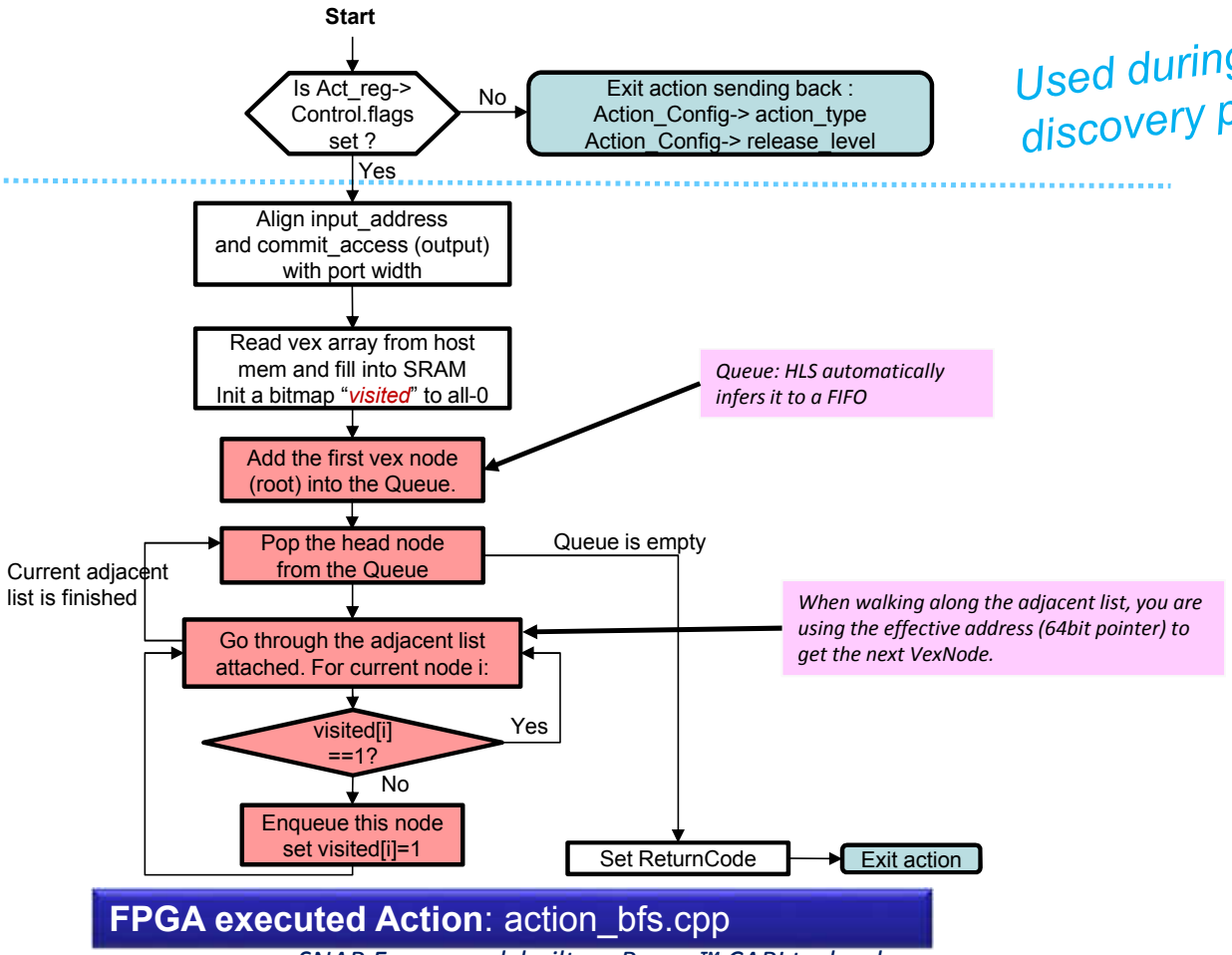
Application Code + software action code : what's in it?



Hardware action Code : what's in it?

hls_action

process_action



Used during discovery phase only

FPGA executed Action: action_bfs.cpp

Constants - Ports

Constants: ➔ $ACTION_ROOT = snap/actions/hls_bfs

Constant name	Value	Type	Definition location	Usage
BFS_ACTION_TYPE	0x10141004	Fixed	$ACTION_ROOT/include/action_bfs.h	BFS ID - list is in snap/ActionTypes.md
HW_RELEASE_LEVEL	0x00000014	Variable	$ACTION_ROOT/hw/hls_bfs.**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	No
nvme	NVMe data bus in/out Addr : 32bits - Data : 32bits	No

MMIO Registers

$ACTION_ROOT/hw/action_bfs.H

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

$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;
```

$ACTION_ROOT/include/action_bfs.h

```
typedef struct bfs_job {
    struct snap_addr input_adjtable;
    struct snap_addr output_traverse;
    uint32_t vex_num;
    uint32_t start_root;
    uint32_t status_pos;
    uint32_t status_vex;
} bfs_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;
```

User defined MMIO registers
= Parameters for the accelerated function