**HSA extension**

**AMD AQL profile library**

**API Specification**

rev 1.1.0

© Advanced Micro Devices Corp. 2017

All rights reserved.

**Revision history**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Authors** | **Description** |
| 1.0.0 | 5/17/2017 | Evgeny Shcherbakov | Initial version |
| 1.0.1 | 5/22/2017 | Evgeny Shcherbakov | Minor changes and code comments |
| 1.1.0 | 6/6/2017 | Evgeny Shcherbakov | Name change to “HAS extension AMD AQL profile” and minor fixes |

**Contents**

[**Revision history 2**](#_Toc482817158)

[**1.** **High level overview 2**](#_Toc482817159)

[**2.** **Library implementation requirements 2**](#_Toc482817160)

[**3.** **Application responsibilities and requirements 3**](#_Toc482817161)

[**4.** **Library basic usage flow 3**](#_Toc482817162)

[**5.** **Library API 4**](#_Toc482817163)

[5.1. Description 4](#_Toc482817164)

[5.2. Supported profiling features 4](#_Toc482817165)

[5.3. Profiling events 5](#_Toc482817166)

[5.4. Profiling parameters 5](#_Toc482817167)

[5.5. Profile context object 5](#_Toc482817168)

[5.6. AQL packets populating methods 6](#_Toc482817169)

[5.7. Get profile info 7](#_Toc482817170)

[**6.** **Application code examples 7**](#_Toc482817171)

[6.1. PMC profiling 7](#_Toc482817172)

[6.2. SQTT profiling 9](#_Toc482817173)

1. **High level overview**

The library provides a C based API for helper functionality to enable a profiling of GPU compute applications. The profiling includes collecting of Performance Counters (PMC) and SQ Thread Trace (SQTT) data.

An HSA application can use the library for generating HSA\_PACKET\_TYPE\_VENDOR\_SPECIFIC AQL packets which should be submitted by the application to configure, start and stop the profiling. The library is written on C and doesn’t have direct dependencies on any infrastructure and implemented according to HSA AQL, GFX PM4 packets and the GFXIP specifications.

The generated VENDOR\_SPECIFIC AQL packets carry a PM4 IB packet inside which refers to an actual PM4 commands buffer for programming GPU PMC/SQTT.

1. **Library implementation requirements**
   1. The library provides a C based API generic for PMC and SQTT profiling and supports both simultaneously.
   2. The library provides an API for querying needed buffers allocations attributes. An application performs all actual memory allocations. It is expected that the application is aware of GPU PMC/SQTT capabilities and configurations.
   3. The library is implemented according to HSA AQL, GFX PM4 packets and the GFXIP specifications. The library headers include HSA headers for using HSA status codes and the agent handle.
   4. The library is stateless. It provides the data structures and the helper methods to the application and all data provided or processed by the library is allocated in the application context.
   5. The library implementation is abstracted from the specific GFXIP. Internally there are separate instances of the implementation which are GFXIP specific and a caller is responsible to pass in the GPU handle.
   6. The block name is the same for a block instances set, for example each block instance from the TCC block set, TCC0, TCC1, …, TCCN will have the same block name HSA\_EXT\_AMD\_AQL\_PROFILE\_BLOCKS\_TCC. The full name of a block instance is formed by the block name and the block index.
2. **Application responsibilities and requirements**
3. It is expected that the application is aware of GPU PMC/SQTT capabilities and configurations.
4. The application performs all actual memory allocations, deallocations and the library provides API for querying needed allocations attributes.
5. The application is responsible for instantiating a profile object and for submitting and waiting for completion of the library generated AQL packets.
6. It is required that the command buffer was host and device accessible and executable. It is required that the output data buffer was device accessible.
7. The block name is the same for a block instances set, for example each block instance from the TCC block set, TCC0, TCC1, …, TCCN will have the same block name HSA\_EXT\_AMD\_AQL\_PROFILE\_BLOCKS\_TCC. The full name of a block instance is formed by the block name and the block index.
8. The application is responsible for copying data from the GPU memory if the profiling data was allocated in the GPU device memory.
9. The application can enable SQTT+PMC profiling for SQ by submitting PMC start packet after SQTT start packet and the same is applied for stop packets.
10. **Library basic usage flow**

GPU handle

Events list

Parameters list

profile object

profile object

* the profiling library API calls
* the application task

Legend:

STEP 5

STEP 4

STEP 3

STEP 2

STEP 1

Wait for the STOP AQL packet completion

Generate START AQL packet

Application AQL packets

Generate STOP AQL packet

Instantiate profile object

1. **Library API**

**Description**

The library provides helper methods for instantiation of the profile context object and for populating of the start and stop AQL packets. The profile object contains a profiling events list and needed for profiling buffers descriptors, a command buffer and an output data buffer. To check if there was an error the library methods return a status code. Also the library provides methods for querying required buffers attributes, to validate the event attributes and to get profiling output data.

Returned API status:

* hsa\_status\_t – HSA status codes are used from hsa.h header

Supported profiling features:

* hsa\_ext\_amd\_aql\_profile\_event\_type\_t – supported profiling events
* hsa\_ext\_amd\_aql\_profile\_block\_name\_t – supported performance counters blocks

Profiling events:

* hsa\_ext\_amd\_aql\_profile\_event\_t – PMC event object structure
* hsa\_ext\_amd\_aql\_profile\_validate\_event – method to check if event is valid for the specific GPU
* The counter ID values should be used from GFXIPs perfcounter users guides which is the counters select value, “Performance Counters Selection” chapter

Profiling parameters

* hsa\_ext\_amd\_aql\_profile\_parameter\_name\_t – supported profiling parameters
* hsa\_ext\_amd\_aql\_profile\_parameters\_t - profile parameter object

Profile context object and instantiation methods:

* hsa\_ext\_amd\_aql\_profile\_descriptor\_t – buffer descriptor
* hsa\_ext\_amd\_aql\_profile\_t – profile context object structure, contains a profiling events list and needed for profiling buffers descriptors, a command buffer and an output data buffer

AQL packets populating methods:

* hsa\_ext\_amd\_aql\_profile\_packet\_t - vendor specific profiling AQL packet
* hsa\_ext\_amd\_aql\_profile\_start – populating the START AQL packet
* hsa\_ext\_amd\_aql\_profile\_stop – populating the STOP AQL packet
* hsa\_ext\_amd\_aql\_profile\_legacy\_get\_pm4 – converting of the profiling AQL packet to PM4 packet, GFX8 support

Get profile info:

* hsa\_ext\_amd\_aql\_profile\_info\_type\_t - supported system profile info types, like the command buffer size and the profiling PMC results
* hsa\_ext\_amd\_aql\_profile\_info\_data\_t – profile generic output data
* hsa\_ext\_amd\_aql\_profile\_data\_callback\_t - definition of output data iterator callback
* hsa\_ext\_amd\_aql\_profile\_get\_info - method for getting the profile info
* hsa\_ext\_amd\_aql\_profile\_iterate\_data - method for iterating the events output data

**Supported profiling features**

Supported profiling events:

typedef enum {

HSA\_EXT\_AMD\_AQL\_PROFILE\_EVENT\_PMC,

HSA\_EXT\_AMD\_AQL\_PROFILE\_EVENT\_SQTT

} hsa\_ext\_amd\_aql\_profile\_event\_type\_t;

Supported performance counters blocks:

typedef enum {

HSA\_EXT\_AMD\_AQL\_PROFILE\_BLOCK\_SQ,

HSA\_EXT\_AMD\_AQL\_PROFILE\_BLOCKS\_TCC,

…

HSA\_EXT\_AMD\_AQL\_PROFILE\_BLOCKS\_NUMBER

} hsa\_ext\_amd\_aql\_profile\_block\_name\_t;

The block ID is the same for a block instances set. For example each block instance from the TCC block set, TCC0, TCC1, …, TCCN have the same block ID HSA\_EXT\_AMD\_AQL\_PROFILE\_BLOCKS\_TCC.

**Profiling events**

PMC event object structure:

typedef struct {

hsa\_ext\_amd\_aql\_profile\_block\_name\_t block\_name;

uint32\_t block\_index;

uint32\_t counter\_id,

} hsa\_ext\_amd\_aql\_profile\_event\_t;

‘counter\_id’ value is specified in GFXIPs perfcounter user guides which is the counters select value, “Performance Counters Selection” chapter.

Check if event is valid for the specific GPU:

hsa\_status\_t hsa\_ext\_amd\_aql\_profile\_validate\_event(

hsa\_agent\_t agent, // HSA handle for the profiling GFXIP

const hsa\_ext\_amd\_aql\_profile\_event\_t \* event, // Pointer on a validated event

bool \* result); // True if the event is valid, False otherwise

**Profiling parameters**

All parameters are generic and if not applicable for a specific profile configuration then error status will be returned.

Supported profiling parameters:

typedef enum {

// SQTT applicable parameters

HSA\_EXT\_AMD\_AQL\_PROFILE\_PARAM\_TOKEN\_MASK,

HSA\_EXT\_AMD\_AQL\_PROFILE\_PARAM\_REG\_MASK,

HSA\_EXT\_AMD\_AQL\_PROFILE\_PARAM\_VM\_ID\_MASK,

HSA\_EXT\_AMD\_AQL\_PROFILE\_PARAM\_INSTRUCTION\_MASK,

HSA\_EXT\_AMD\_AQL\_PROFILE\_PARAM\_COMPUTE\_UNIT\_TARGET,

HSA\_EXT\_AMD\_AQL\_PROFILE\_PARAM\_SHADER\_ARRAY\_TARGET,

HSA\_EXT\_AMD\_AQL\_PROFILE\_PARAM\_SIMD\_MASK,

HSA\_EXT\_AMD\_AQL\_PROFILE\_PARAM\_USER\_DATA,

HSA\_EXT\_AMD\_AQL\_PROFILE\_PARAM\_CAPTURE\_MODE,

HSA\_EXT\_AMD\_AQL\_PROFILE\_PARAM\_IS\_WRAPPED,

HSA\_EXT\_AMD\_AQL\_PROFILE\_PARAM\_RANDOM\_SEED // Seems to be obsolete and a candidate for removal

} hsa\_ext\_amd\_aql\_profile\_parameter\_name\_t;

Profile parameter object:

typedef struct {

hsa\_ext\_amd\_aql\_profile\_parameter\_name\_t parameter\_name;

uint32\_t value;

} hsa\_ext\_amd\_aql\_profile\_parameters\_t;

**Profile context object**

The library provides a profile object structure which contains the events array, a buffer for the profiling start/stop commands and a buffer for the output data. The buffers are specified by the buffer descriptors and allocated by the application.

The buffers allocation attributes, the command buffer size, the PMC output buffer size as well as profiling output data can be get using the generic get profile info helper method hsa\_ext\_amd\_aql\_profile\_get\_info.

Buffer descriptor:

typedef struct {

    void \* ptr;

    uint32\_t size;

} hsa\_ext\_amd\_aql\_profile\_descriptor\_t;

Profile context object structure, contains profiling events list and needed for profiling buffers descriptors, a command buffer and an output data buffer:

typedef struct {

hsa\_agent\_t agent; // GFXIP handle

hsa\_ext\_amd\_aql\_profile\_event\_type\_t type; // Events type

const hsa\_ext\_amd\_aql\_profile\_event\_t \* events; // Events array

uint32\_t event\_count; // Events count

const hsa\_ext\_amd\_aql\_profile\_parameters\_t \* parameters; // Parameters array

uint32\_t parameter\_count; // Parameters count

hsa\_ext\_amd\_aql\_profile\_descriptor\_t output\_buffer; // Output data buffer

hsa\_ext\_amd\_aql\_profile\_descriptor\_t command\_buffer; // PM4 commands buffer

} hsa\_ext\_amd\_aql\_profile\_t;

**AQL packets populating methods**

The helper methods to populate provided by the application START and STOP AQL packets which the application is required to submit before and after profiled GPU task packets respectively.

Vendor specific profiling AQL packet:

typedef struct {

uint16\_t header;

…

hsa\_signal\_t completion\_signal;

} hsa\_ext\_amd\_aql\_profile\_packet\_t;

The format field in the profile AQL packets header set to VENDOR\_SPECIFIC, which is zero.

Method to populate the provided AQL packet with profiling start commands:

hsa\_status\_t hsa\_ext\_amd\_aql\_profile\_start(

const hsa\_ext\_amd\_aql\_profile\_t \* profile, // [in] profile context object

hsa\_ext\_amd\_aql\_profile\_packet\_t \* aql\_start\_packet); // [out] profile start AQL packet

Method to populate the provided AQL packet with profiling stop commands:

hsa\_status\_t hsa\_ext\_amd\_aql\_profile\_stop(

const hsa\_ext\_amd\_aql\_profile\_t \* profile, // [in] profile context object

hsa\_ext\_amd\_aql\_profile\_packet\_t \* aql\_stop\_packet); // [out] profile stop AQL packet

Converting of the profiling AQL packet to PM4 packet, GFX8 support:

#define HSA\_EXT\_AMD\_AQL\_PROFILE\_LEGACY\_PM4\_PACKET\_SIZE 64

hsa\_status\_t hsa\_ext\_amd\_aql\_profile\_legacy\_get\_pm4(

const hsa\_ext\_amd\_aql\_profile\_packet\_t \* aql\_packet, // [in] AQL profile start/stop packet

void \* pm4); // [out] profile legacy PM4 packet blob

**Get profile info**

Generic method for getting various profile info including profile buffers attributes like the command buffer size and the profiling results.

It’s implied that all counters are 64bit values.

Profile attributes:

typedef union {

HSA\_EXT\_AMD\_AQL\_PROFILE\_INFO\_COMMAND\_BUFFER\_SIZE, // get\_info returns uint32\_t value

HSA\_EXT\_AMD\_AQL\_PROFILE\_INFO\_PMC\_BUFFER\_SIZE, // get\_info returns uint32\_t value

HSA\_EXT\_AMD\_AQL\_PROFILE\_INFO\_PMC\_DATA, // get\_info returns PMC uint64\_t value

// in info\_data\_t object

HSA\_EXT\_AMD\_AQL\_PROFILE\_INFO\_SQTT\_DATA // get\_info returns SQTT buffer pointer/size

// in info\_data\_t object

} hsa\_ext\_amd\_aql\_profile\_info\_type\_t;

Profile generic output data:

typedef struct {

uint32\_t sample\_id; // PMC sample or SQTT buffer index

union {

struct {

hsa\_ext\_amd\_aql\_profile\_event\_t event; // PMC event

uint64\_t result; // PMC result

} pmc\_data;

hsa\_ext\_amd\_aql\_profile\_descriptor\_t sqtt\_data; // SQTT output data descriptor

};

} hsa\_ext\_amd\_aql\_profile\_info\_data\_t;

Definition of output data iterator callback:

hsa\_status\_t (\*hsa\_ext\_amd\_aql\_profile\_data\_callback\_t)(

hsa\_ext\_amd\_aql\_profile\_info\_type\_t info\_type, // [in] data type. PMC or SQTT data

hsa\_ext\_amd\_aql\_profile\_info\_data\_t \* info\_data, // [in] info data object

void \* callback\_data); // [in/out] data pointer passed to the callback

Method for getting the profile info:

hsa\_status\_t hsa\_ext\_amd\_aql\_profile\_get\_info(

const hsa\_ext\_amd\_aql\_profile\_t \* profile, // [in] profile context object

hsa\_ext\_amd\_aql\_profile\_info\_type\_t attribute, // [in] requested profile attribute

void \* value); // [in/out] returned value

Method for iterating the events output data:

hsa\_status\_t hsa\_ext\_amd\_aql\_profile\_iterate\_data(

const hsa\_ext\_amd\_aql\_profile\_t \* profile, // [in] profile context object

hsa\_ext\_amd\_aql\_profile\_data\_callback\_t callback; // [in] callback to iterate the output data

void \* callback\_data); // [in/out] data passed to the callback

1. **Application code examples**

**PMC profiling**

hsa\_status\_t status;

hsa\_agent\_t agent;

hsa\_ext\_amd\_aql\_profile\_t profile;

hsa\_ext\_amd\_aql\_profile\_event\_t event;

hsa\_ext\_amd\_aql\_profile\_packet\_t \* start\_packet;

hsa\_ext\_amd\_aql\_profile\_packet\_t \* stop\_packet;

hsa\_ext\_amd\_aql\_profile\_info\_data\_t sample;

uint32\_t command\_buffer\_alignment;

uint32\_t command\_buffer\_size;

uint32\_t output\_buffer\_alignment;

uint32\_t output\_buffer\_size;

// Resulted PMC sample value

uint64\_t result;

// GPU identificator

hsa\_iterate\_agents(<get agent callback>, &agent);

// Instantiation of the profile object //////////////////////////////////////////////////////////////

// Set the event fields

event.block\_name = HSA\_EXT\_AMD\_AQL\_PROFILE\_BLOCK\_SQ;

event.block\_index = 0;

event.counter\_id = <some counter ID>;

// Initialization the profile

memset(&profile, 0, sizeof(profile));

profile.agent = agent;

profile.type = HSA\_EXT\_AMD\_AQL\_PROFILE\_EVENT\_PMC;

// set enabled events list, one event in this example

profile.events = &event;

profile.event\_count = 1;

// Profile buffers attributes

command\_buffer\_alignment = <command buffer alignment>;

status = hsa\_ext\_amd\_aql\_profile\_get\_info(

&profile,

HSA\_EXT\_AMD\_AQL\_PROFILE\_INFO\_COMMAND\_BUFFER\_SIZE,

&command\_buffer\_size);

<check the status>

output\_buffer\_alignment = <PMC output data alignment>;

status = hsa\_ext\_amd\_aql\_profile\_get\_info(

&profile,

HSA\_EXT\_AMD\_AQL\_PROFILE\_INFO\_PMC\_DATA\_SIZE,

&output\_buffer\_size);

<check the status>

// Application is allocating the command buffer and output buffer

profile.command\_buffer.ptr = allocate(

command\_buffer\_alignment,

command\_buffer\_size,

MODE\_HOST\_ACC|MODE\_DEV\_ACC|MODE\_EXEC\_DATA);

profile.command\_buffer.size = command\_buffer\_size;

profile.output\_buffer.ptr = allocate(

output\_buffer\_alignment,

output\_buffer\_size,

MODE\_HOST\_ACC|MODE\_DEV\_ACC);

profile.output\_buffer.size = output\_buffer­\_size;

// Generation of the START AQL packet //////////////////////////////////////////////////////////////

<enqueuing AQL start\_paket>

// Populating the AQL start packet

status = hsa\_ext\_amd\_aql\_profile\_start(&profile, start\_packet);

<check the status>

<enqueuing AQL packets of the application task>

// Generation of the STOP AQL packet //////////////////////////////////////////////////////////////

<enqueuing AQL stop\_paket>

// Populating the AQL stop packet

status = hsa\_ext\_amd\_aql\_profile\_stop(&profile, stop\_packet);

<check the status>

<wait for the AQL stop\_paket completion>

// Query PMC sample for shader 0 as sample.pmc\_data.result

sample.sample\_id = 0;

sample.pmc\_data.event = event;

hsa\_ext\_amd\_aql\_profile\_get\_info(

&profile,

HSA\_EXT\_AMD\_AQL\_PROFILE\_INFO\_PMC\_DATA,

&sample);

// Final PMC sample result

result = sample.pmc\_data.result;

**SQTT profiling**

hsa\_status\_t status;

hsa\_agent\_t agent;

hsa\_ext\_amd\_aql\_profile\_t profile;

hsa\_ext\_amd\_aql\_profile\_parameter\_t parameter;

hsa\_ext\_amd\_aql\_profile\_packet\_t \* start\_packet;

hsa\_ext\_amd\_aql\_profile\_packet\_t \* stop\_packet;

hsa\_ext\_amd\_aql\_profile\_info\_data\_t sample;

uint32\_t command\_buffer\_alignment;

uint32\_t command\_buffer\_size;

uint32\_t output\_buffer\_alignment;

uint32\_t output\_buffer\_size;

// Resulted SQTT trace

hsa\_ext\_amd\_aql\_profile\_descriptor\_t trace;

// GPU identificator

hsa\_iterate\_agents(<get agent callback>, &agent);

// Instantiation of the profile object //////////////////////////////////////////////////////////////

// Set the parameter fields

parameter.name = HSA\_EXT\_AMD\_AQL\_PROFILE\_PARAM\_INSTRUCTION\_MASK;

parameter.value = <some trace instructions mask>;

// Initialization the profile

memset(&profile, 0, sizeof(profile));

profile.agent = agent;

profile.type = HSA\_EXT\_AMD\_AQL\_PROFILE\_EVENT\_SQTT;

// set parameters list, one parameter in this example

profile.parameters = &parameter;

profile.parameter\_count = 1;

// Profile buffers attributes

command\_buffer\_alignment = <command buffer alignment>;

status = hsa\_ext\_amd\_aql\_profile\_get\_info(

&profile,

HSA\_EXT\_AMD\_AQL\_PROFILE\_INFO\_COMMAND\_BUFFER\_SIZE,

&command\_buffer\_size);

<check the status>

output\_buffer\_alignment = <SQTT output data alignment>;

output\_buffer\_size = <SQTT data application estimated size>;

// Application is allocating the command buffer and output buffer

profile.command\_buffer.ptr = allocate(

command\_buffer\_alignment,

command\_buffer\_size,

MODE\_EXEC\_DATA|MODE\_HOST\_ACC|MODE\_DEV\_ACC);

profile.command\_buffer.size = command\_buffer\_size;

profile.output\_buffer.ptr = allocate(

output\_buffer\_alignment,

output\_buffer\_size,

MODE\_DEV\_MEM);

profile.output\_buffer.size = output\_buffer\_size;

// Generation of the START AQL packet //////////////////////////////////////////////////////////////

<enqueuing AQL start\_packet>

// Populating the AQL start packet

status = hsa\_ext\_amd\_aql\_profile\_start(&profile, start\_packet);

<check the status>

<enqueuing AQL packets of the application task>

// Generation of the STOP AQL packet //////////////////////////////////////////////////////////////

<enqueuing AQL stop\_paket>

// Populating the AQL stop packet

status = hsa\_ext\_amd\_aql\_profile\_stop(&profile, stop\_packet);

<check the status>

<wait for the AQL stop\_packet completion>

// Query SQTT trace for shader engine 0 as sample.sqtt\_data

sample.sample\_id = 0;

hsa\_ext\_amd\_aql\_profile\_get\_info(

&profile,

HSA\_EXT\_AMD\_AQL\_PROFILE\_INFO\_PMC\_DATA,

&sample);

// Final SQTT trace descriptor

trace = sample.sqtt\_data;