AMD GPU Debug API Specification

Generated by Doxygen 1.8.11

Contents

1 Introduction **1**

1.1 Overview . . . 1

1.2 Assumptions . . . 1

1.3 Requirements . . . 1

2 Data Structure Index **3**

2.1 Data Structures . . . 3

3 File Index **5**

3.1 File List . . . 5

4 Data Structure Documentation **7**

4.1 HwDbgDataBreakpointInfo Struct Reference . . . 7

4.1.1 Detailed Description . . . 7

4.1.2 Field Documentation . . . 7

4.1.2.1 dataBreakpointMode . . . 7

4.1.2.2 dataSize . . . 7

4.1.2.3 pAddress . . . 8

4.2 HwDbgDim3 Struct Reference . . . 8

4.2.1 Detailed Description . . . 8

4.2.2 Field Documentation . . . 8

4.2.2.1 x . . . 8

4.2.2.2 y . . . 8

4.2.2.3 z . . . 8

4.3 HwDbgLoaderSegmentDescriptor Struct Reference . . . 9

4.3.1 Detailed Description . . . 9
4.3.2 Field Documentation . . . 9
4.3.2.1 codeObjectStorageOffset . . . 9
4.3.2.2 codeObjectStorageSize . . . 9
4.3.2.3 codeObjectStorageType . . . 10
4.3.2.4 device . . . 10
4.3.2.5 executable . . . 10
4.3.2.6 pCodeObjectStorageBase . . . 10
4.3.2.7 pSegmentBase . . . 10
4.3.2.8 segmentSize . . . 10
4.4 HwDbgState Struct Reference . . . 11
4.4.1 Detailed Description . . . 11
4.4.2 Field Documentation . . . 11
4.4.2.1 behaviorFlags . . . 11
4.4.2.2 packetId . . . 11
4.4.2.3 pDevice . . . 11
4.4.2.4 pPacket . . . 11
4.5 HwDbgWavefrontInfo Struct Reference . . . 12
4.5.1 Detailed Description . . . 12
4.5.2 Field Documentation . . . 12
4.5.2.1 breakpointType . . . 12
4.5.2.2 codeAddress . . . 13
4.5.2.3 dataBreakpointHandle . . . 13
4.5.2.4 executionMask . . . 13
4.5.2.5 pOtherData . . . 13
4.5.2.6 wavefrontAddress . . . 13
4.5.2.7 workGroupId . . . 13
4.5.2.8 workItemId . . . 13

5 File Documentation **15**

5.1 AMDGPUDebug.h File Reference 15

5.1.1 Detailed Description 17

5.1.2 Macro Definition Documentation 18

5.1.2.1 AMDGPUDEBUG_VERSION_BUILD 18

5.1.2.2 AMDGPUDEBUG_VERSION_MAJOR 18

5.1.2.3 AMDGPUDEBUG_VERSION_MINOR 18

5.1.2.4 HWDBG_API_CALL 18

5.1.2.5 HWDBG_API_ENTRY 18

5.1.2.6 HWDBG_WAVEFRONT_SIZE 18

5.1.3 Typedef Documentation 18

5.1.3.1 HwDbgCodeAddress 18

5.1.3.2 HwDbgCodeBreakpointHandle 19

5.1.3.3 HwDbgContextHandle 19

5.1.3.4 HwDbgDataBreakpointHandle 19

5.1.3.5 HwDbgLoggingCallback 19

5.1.3.6 HwDbgWavefrontAddress 19

5.1.4 Enumeration Type Documentation 19

5.1.4.1 HwDbgAPIType 19

5.1.4.2 HwDbgBehaviorType 20

5.1.4.3 HwDbgBreakpointType 20

5.1.4.4 HwDbgCommand 20

5.1.4.5 HwDbgDataBreakpointMode 21

5.1.4.6 HwDbgEventType 21

5.1.4.7 HwDbgLoaderCodeObjectStorageType 21

5.1.4.8 HwDbgLogType 22

5.1.4.9 HwDbgStatus 22

5.1.5 Function Documentation 22

5.1.5.1 HwDbgBeginDebugContext(const HwDbgState state, HwDbgContextHandle *pDebugContextOut) 22

5.1.5.2 HwDbgBreakAll(const HwDbgContextHandle hDebugContext) 23

5.1.5.3 HwDbgContinueEvent(HwDbgContextHandle hDebugContext, const HwDbgCommand command) . . . 24

5.1.5.4 HwDbgCreateCodeBreakpoint(HwDbgContextHandle hDebugContext, const HwDbgCodeAddress codeAddress, HwDbgCodeBreakpointHandle *pBreakpointOut) . . . 24

5.1.5.5 HwDbgCreateDataBreakpoint(HwDbgContextHandle hDebugContext, const HwDbgDataBreakpointInfo breakpointInfo, HwDbgDataBreakpointHandle *pDataBreakpointOut) . . . 25

5.1.5.6 HwDbgDeleteAllCodeBreakpoints(HwDbgContextHandle hDebugContext) . . . 26

5.1.5.7 HwDbgDeleteAllDataBreakpoints(HwDbgContextHandle hDebugContext) . . . 26

5.1.5.8 HwDbgDeleteCodeBreakpoint(HwDbgContextHandle hDebugContext, HwDbgCodeBreakpointHandle hBreakpoint) . . . 27

5.1.5.9 HwDbgDeleteDataBreakpoint(HwDbgContextHandle hDebugContext, HwDbgDataBreakpointHandle hDataBreakpoint) . . . 28

5.1.5.10 HwDbgEndDebugContext(HwDbgContextHandle hDebugContext) . . . 28

5.1.5.11 HwDbgGetActiveWavefronts(const HwDbgContextHandle hDebugContext, const HwDbgWavefrontInfo **ppWavefrontInfoOut, uint32_t *pNumWavefrontsOut) . . . 29

5.1.5.12 HwDbgGetAPIType(HwDbgAPIType *pAPITypeOut) . . . 30

5.1.5.13 HwDbgGetAPIVersion(uint32_t *pVersionMajorOut, uint32_t *pVersionMinorOut, uint32_t *pVersionBuildOut) . . . 30

5.1.5.14 HwDbgGetCodeBreakpointAddress(const HwDbgContextHandle hDebugContext, const HwDbgCodeBreakpointHandle hBreakpoint, HwDbgCodeAddress *pCodeAddressOut) . . . 31

5.1.5.15 HwDbgGetDataBreakpointInfo(const HwDbgContextHandle hDebugContext, const HwDbgDataBreakpointHandle hDataBreakpoint, HwDbgDataBreakpointInfo *pDataBreakpointInfoOut) . . . 31

5.1.5.16 HwDbgGetDispatchedKernelName(const HwDbgContextHandle hDebugContext, const char **ppKernelNameOut) . . . 32

5.1.5.17 HwDbgGetKernelBinary(const HwDbgContextHandle hDebugContext, const void **ppBinaryOut, size_t *pBinarySizeOut) . . . 33

5.1.5.18 HwDbgGetLoadedSegmentDescriptors(HwDbgLoaderSegmentDescriptor *pSegmentDescriptorListOut, size_t *pSegmentDescriptorCountOut) . . . 34

5.1.5.19 HwDbgInit(void *pApiTable) . . . 34

5.1.5.20 HwDbgKillAll(const HwDbgContextHandle hDebugContext) . . . 35

5.1.5.21 HwDbgReadMemory(const HwDbgContextHandle hDebugContext, const uint32_t memoryRegion, const HwDbgDim3 workGroupId, const HwDbgDim3 workItemId, const size_t offset, const size_t numBytesToRead, void *pMemOut, size_t *pNumBytesOut) . . . 35

5.1.5.22 HwDbgSetLoggingCallback(uint32_t types, HwDbgLoggingCallback pCallback, void *pUserData) . . . 36

5.1.5.23 HwDbgShutDown() . . . 37

5.1.5.24 HwDbgWaitForEvent(HwDbgContextHandle hDebugContext, const uint32_t timeout, HwDbgEventType *pEventTypeOut) . . . 37

Index **39**

Chapter 1

Introduction

1.1 Overview

This document describes a set of interfaces which can be used by debugger or application developers to incorporate GPU kernel debugging functionality into their debugger or application running on AMD Graphics Core Next GPUs (or APUs).

The AMD GPU Kernel Debugging API has been designed to hide the multiple driver API specific implementations and the internal architecture of a particular GPU device. It has evolved starting from a minimal set of GPU debugging APIs that can be currently supported by AMD GPUs and software stacks. As more GPU debug features are implemented and validated, the API will evolve further. It is still a work-in-progress.

For HSA, this API together with the AMD HSA binary interface, AMD HSA debug info and AMD HSA API and dispatch interception mechanism form the AMD HSA GPU Debugging Architecture. Refer to the "AMD HSA GPU Debugging Architecture" document for more information.

1.2 Assumptions

The AMD GPU Kernel Debugging API is an "in-process" debug API. That is, the API must be called from the same process address space as the program being debugged and will have direct access to all process resources. No OS provided inter-process debug mechanisms are required, but it should be reasonably straightforward for tool developers to create a client/server remote debugging model through the introduction of a simple communication protocol.

To inject these kernel debugging API calls into the debugged application process address space, the API and kernel dispatch interception mechanism provided through amd_hsa_tools_interfaces.h can be used.

1.3 Requirements

For HSA:

1. AMD Kaveri and Carrizo APUs
2. HSA Runtime and HSAIL 1.0 Final

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

HwDbgDataBreakpointInfo 7
HwDbgDim3 8
HwDbgLoaderSegmentDescriptor 9
HwDbgState 11
HwDbgWavefrontInfo 12

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

AMDGPUDebug.h
The AMD GPU Kernel Debugging API to implement device kernel debugging on AMD Graphics Core Next (GCN) GPUs 15

Chapter 4

Data Structure Documentation

4.1 HwDbgDataBreakpointInfo Struct Reference

`#include <AMDGPUDebug.h>`

Data Fields

- HwDbgDataBreakpointMode dataBreakpointMode
- uint64_t dataSize
- void * pAddress

4.1.1 Detailed Description

A structure to hold all the info required to create a single data breakpoint.

Warning

This is not yet supported

Definition at line 247 of file AMDGPUDebug.h.

4.1.2 Field Documentation

4.1.2.1 HwDbgDataBreakpointMode HwDbgDataBreakpointInfo::dataBreakpointMode

the relevant mode for the data breakpoint

Definition at line 250 of file AMDGPUDebug.h.

4.1.2.2 uint64_t HwDbgDataBreakpointInfo::dataSize

the size of data in bytes being watched

Definition at line 253 of file AMDGPUDebug.h.

4.1.2.3 void* HwDbgDataBreakpointInfo::pAddress

the memory address to be watched

Definition at line 256 of file AMDGPUDebug.h.

4.2 HwDbgDim3 Struct Reference

`#include <AMDGPUDebug.h>`

Data Fields

- uint32_t x
- uint32_t y
- uint32_t z

4.2.1 Detailed Description

A three dimensional type, used by work-group and work-item ids.

Definition at line 238 of file AMDGPUDebug.h.

4.2.2 Field Documentation

4.2.2.1 uint32_t HwDbgDim3::x

x dimension

Definition at line 240 of file AMDGPUDebug.h.

4.2.2.2 uint32_t HwDbgDim3::y

y dimension

Definition at line 241 of file AMDGPUDebug.h.

4.2.2.3 uint32_t HwDbgDim3::z

z dimension

Definition at line 242 of file AMDGPUDebug.h.

4.3 HwDbgLoaderSegmentDescriptor Struct Reference

`#include <AMDGPUDebug.h>`

Data Fields

- uint64_t device
- uint64_t executable
- HwDbgLoaderCodeObjectStorageType codeObjectStorageType
- const void * pCodeObjectStorageBase
- size_t codeObjectStorageSize
- size_t codeObjectStorageOffset
- const void * pSegmentBase
- size_t segmentSize

4.3.1 Detailed Description

A structure to hold information related to each loaded segment.

Definition at line 260 of file AMDGPUDebug.h.

4.3.2 Field Documentation

4.3.2.1 size_t HwDbgLoaderSegmentDescriptor::codeObjectStorageOffset

If the storage type of the code object that is backing underlying memory segment is:

- HWDBG_LOADER_CODE_OBJECT_STORAGE_TYPE_NONE, then 0;
- other, then offset, in bytes, from the beginning of the code object to the first byte in the code object data is copied from.

Definition at line 297 of file AMDGPUDebug.h.

4.3.2.2 size_t HwDbgLoaderSegmentDescriptor::codeObjectStorageSize

If the storage type of the code object that is backing underlying memory segment is:

- HWDBG_LOADER_CODE_OBJECT_STORAGE_TYPE_NONE, then 0;
- HWDBG_LOADER_CODE_OBJECT_STORAGE_TYPE_FILE, then the length of the filepath to the code object (including null-terminating character);
- HWDBG_LOADER_CODE_OBJECT_STORAGE_TYPE_MEMORY, then the size, in bytes, of the memory occupied by the code object.

Definition at line 289 of file AMDGPUDebug.h.

4.3.2.3 HwDbgLoaderCodeObjectStorageType HwDbgLoaderSegmentDescriptor::codeObjectStorageType

Storage type of the code object that is backing underlying memory segment.

Definition at line 270 of file AMDGPUDebug.h.

4.3.2.4 uint64_t HwDbgLoaderSegmentDescriptor::device

Device underlying memory segment is allocated on. If the code object that is backing underlying memory segment is program code object, then 0.

Definition at line 264 of file AMDGPUDebug.h.

4.3.2.5 uint64_t HwDbgLoaderSegmentDescriptor::executable

Executable that is managing this underlying memory segment.

Definition at line 267 of file AMDGPUDebug.h.

4.3.2.6 const void* HwDbgLoaderSegmentDescriptor::pCodeObjectStorageBase

If the storage type of the code object that is backing underlying memory segment is:

- HWDBG_LOADER_CODE_OBJECT_STORAGE_TYPE_NONE, then null;
- HWDBG_LOADER_CODE_OBJECT_STORAGE_TYPE_FILE, then null-terminated filepath to the code object;
- HWDBG_LOADER_CODE_OBJECT_STORAGE_TYPE_MEMORY, then host accessible pointer to the first byte of the code object.

Definition at line 279 of file AMDGPUDebug.h.

4.3.2.7 const void* HwDbgLoaderSegmentDescriptor::pSegmentBase

Starting address of the underlying memory segment.

Definition at line 300 of file AMDGPUDebug.h.

4.3.2.8 size_t HwDbgLoaderSegmentDescriptor::segmentSize

Size, in bytes, of the underlying memory segment.

Definition at line 303 of file AMDGPUDebug.h.

4.4 HwDbgState Struct Reference

`#include <AMDGPUDebug.h>`

Data Fields

- void * pDevice
- void * pPacket
- uint64_t packetId
- uint32_t behaviorFlags

4.4.1 Detailed Description

A structure to hold the device state as an input to the HwDbgBeginDebugContext

Definition at line 353 of file AMDGPUDebug.h.

4.4.2 Field Documentation

4.4.2.1 uint32_t HwDbgState::behaviorFlags

flags that the control the behavior of the debug context

Definition at line 366 of file AMDGPUDebug.h.

4.4.2.2 uint64_t HwDbgState::packetId

set to packet_id from the pre-dispatch callback function

Warning

This is not yet supported

Definition at line 363 of file AMDGPUDebug.h.

4.4.2.3 void* HwDbgState::pDevice

set to hsa_agent_t.handle from the pre-dispatch callback function

Definition at line 356 of file AMDGPUDebug.h.

4.4.2.4 void* HwDbgState::pPacket

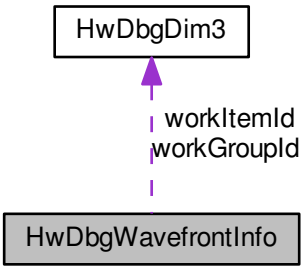
set to hsa_kernel_dispatch_packet_t* from the pre-dispatch callback function

Definition at line 359 of file AMDGPUDebug.h.

4.5 HwDbgWavefrontInfo Struct Reference

`#include <AMDGPUDebug.h>`

Collaboration diagram for HwDbgWavefrontInfo:



Data Fields

- HwDbgDim3 workGroupId
- HwDbgDim3 workItemId [HWDBG_WAVEFRONT_SIZE]
- uint64_t executionMask
- HwDbgWavefrontAddress wavefrontAddress
- HwDbgCodeAddress codeAddress
- HwDbgDataBreakpointHandle dataBreakpointHandle
- HwDbgBreakpointType breakpointType
- void * pOtherData

4.5.1 Detailed Description

A structure to hold the active wave info returned by HwDbgGetActiveWavefronts API

Definition at line 309 of file AMDGPUDebug.h.

4.5.2 Field Documentation

4.5.2.1 HwDbgBreakpointType HwDbgWavefrontInfo::breakpointType

the type of breakpoint that was signaled

Warning

This is not yet supported

Definition at line 332 of file AMDGPUDebug.h.

4.5.2.2 HwDbgCodeAddress HwDbgWavefrontInfo::codeAddress

the byte offset in the ISA binary for the wavefront

Definition at line 324 of file AMDGPUDebug.h.

4.5.2.3 HwDbgDataBreakpointHandle HwDbgWavefrontInfo::dataBreakpointHandle

the data breakpoint handle

Warning

This is not yet supported

Definition at line 328 of file AMDGPUDebug.h.

4.5.2.4 uint64_t HwDbgWavefrontInfo::executionMask

the execution mask of the work-items

Definition at line 318 of file AMDGPUDebug.h.

4.5.2.5 void* HwDbgWavefrontInfo::pOtherData

additional data that can be returned

Definition at line 335 of file AMDGPUDebug.h.

4.5.2.6 HwDbgWavefrontAddress HwDbgWavefrontInfo::wavefrontAddress

the hardware wavefront slot address (not unique for a dispatch)

Definition at line 321 of file AMDGPUDebug.h.

4.5.2.7 HwDbgDim3 HwDbgWavefrontInfo::workGroupId

the work-group id

Definition at line 312 of file AMDGPUDebug.h.

4.5.2.8 HwDbgDim3 HwDbgWavefrontInfo::workItemId[HWDBG_WAVEFRONT_SIZE]

the work-item id (local id within a work-group)

Definition at line 315 of file AMDGPUDebug.h.

Chapter 5

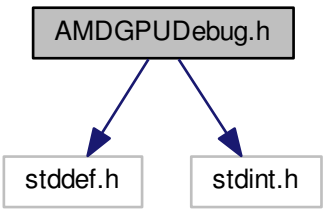
File Documentation

5.1 AMDGPUDebug.h File Reference

The AMD GPU Kernel Debugging API to implement device kernel debugging on AMD Graphics Core Next (GCN) GPUs.

```
#include <stddef.h>
#include <stdint.h>
```

Include dependency graph for AMDGPUDebug.h:



Data Structures

- struct HwDbgDim3
- struct HwDbgDataBreakpointInfo
- struct HwDbgLoaderSegmentDescriptor
- struct HwDbgWavefrontInfo
- struct HwDbgState

Macros

- #define HWDBG_API_ENTRY
- #define HWDBG_API_CALL
- #define AMDGPUDEBUG_VERSION_MAJOR 1
- #define AMDGPUDEBUG_VERSION_MINOR 4
- #define AMDGPUDEBUG_VERSION_BUILD 194
- #define HWDBG_WAVEFRONT_SIZE 64

Typedefs

- typedef uint64_t HwDbgCodeAddress
- typedef uint32_t HwDbgWavefrontAddress
- typedef void * HwDbgContextHandle
- typedef void * HwDbgCodeBreakpointHandle
- typedef void * HwDbgDataBreakpointHandle
- typedef void(* HwDbgLoggingCallback) (void *pUserData, const HwDbgLogType type, const char *const pMessage)

Enumerations

- enum HwDbgStatus {
  HWDBG_STATUS_SUCCESS = 0x0, HWDBG_STATUS_ERROR = 0x01, HWDBG_STATUS_DEVICE_↩
  ERROR = 0x02, HWDBG_STATUS_DRIVER_ERROR = 0x03,
  HWDBG_STATUS_DUPLICATE_BREAKPOINT = 0x04, HWDBG_STATUS_INVALID_ADDRESS_ALIG↩
  NMENT = 0x05, HWDBG_STATUS_INVALID_HANDLE = 0x06, HWDBG_STATUS_INVALID_PARAMET↩
  ER = 0x07,
  HWDBG_STATUS_NULL_POINTER = 0x08, HWDBG_STATUS_OUT_OF_RANGE_ADDRESS = 0x09,
  HWDBG_STATUS_OUT_OF_MEMORY = 0x0A, HWDBG_STATUS_OUT_OF_RESOURCES = 0x0B,
  HWDBG_STATUS_REGISTRATION_ERROR = 0x0C, HWDBG_STATUS_UNDEFINED = 0x0D, HWDB↩
  G_STATUS_UNSUPPORTED = 0x0E, HWDBG_STATUS_NOT_INITIALIZED = 0x0F,
  HWDBG_STATUS_INVALID_BEHAVIOR_STATE = 0x10 }
- enum HwDbgCommand { HWDBG_COMMAND_CONTINUE = 0x0 }
- enum HwDbgAPIType { HWDBG_API_HSA = 0x0 }
- enum HwDbgBreakpointType { HWDBG_BREAKPOINT_TYPE_NONE = 0x0, HWDBG_BREAKPOINT_T↩
  YPE_CODE = 0x1, HWDBG_BREAKPOINT_TYPE_DATA = 0x2 }
- enum HwDbgEventType { HWDBG_EVENT_INVALID = 0x0, HWDBG_EVENT_TIMEOUT = 0x1, HWDBG↩
  _EVENT_POST_BREAKPOINT = 0x2, HWDBG_EVENT_END_DEBUGGING = 0x3 }
- enum HwDbgDataBreakpointMode { HWDBG_DATABREAKPOINT_MODE_READ = 0x1, HWDBG_DAT↩
  ABREAKPOINT_MODE_NONREAD = 0x2, HWDBG_DATABREAKPOINT_MODE_ATOMIC = 0x4, HWD↩
  BG_DATABREAKPOINT_MODE_ALL = 0x7 }
- enum HwDbgLoaderCodeObjectStorageType { HWDBG_LOADER_CODE_OBJECT_STORAGE_TYPE_↩
  NONE = 0, HWDBG_LOADER_CODE_OBJECT_STORAGE_TYPE_FILE = 1, HWDBG_LOADER_COD↩
  E_OBJECT_STORAGE_TYPE_MEMORY = 2 }
- enum HwDbgBehaviorType { HWDBG_BEHAVIOR_NONE = 0x00, HWDBG_BEHAVIOR_DISABLE_DIS↩
  PATCH_DEBUGGING = 0x01 }
- enum HwDbgLogType {
  HWDBG_LOG_TYPE_NONE = 0x00, HWDBG_LOG_TYPE_ASSERT = 0x01, HWDBG_LOG_TYPE_ER↩
  ROR = 0x02, HWDBG_LOG_TYPE_TRACE = 0x04,
  HWDBG_LOG_TYPE_MESSAGE = 0x08, HWDBG_LOG_TYPE_ALL = 0x0f }

Functions

- HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgSetLoggingCallback (uint32_t types, Hw↩
  DbgLoggingCallback pCallback, void *pUserData)
- HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgGetAPIVersion (uint32_t *pVersionMajor↩
  Out, uint32_t *pVersionMinorOut, uint32_t *pVersionBuildOut)
- HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgGetAPIType (HwDbgAPIType *pAPI↩
  TypeOut)
- HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgInit (void *pApiTable)
- HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgShutDown ()
- HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgBeginDebugContext (const HwDbgState state, HwDbgContextHandle *pDebugContextOut)

- HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgEndDebugContext (HwDbgContextHandle hDebugContext)
- HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgWaitForEvent (HwDbgContextHandle hDebugContext, const uint32_t timeout, HwDbgEventType *pEventTypeOut)
- HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgContinueEvent (HwDbgContextHandle hDebugContext, const HwDbgCommand command)
- HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgCreateCodeBreakpoint (HwDbgContextHandle hDebugContext, const HwDbgCodeAddress codeAddress, HwDbgCodeBreakpointHandle *pBreakpointOut)
- HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgDeleteCodeBreakpoint (HwDbgContextHandle hDebugContext, HwDbgCodeBreakpointHandle hBreakpoint)
- HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgDeleteAllCodeBreakpoints (HwDbgContextHandle hDebugContext)
- HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgGetCodeBreakpointAddress (const HwDbgContextHandle hDebugContext, const HwDbgCodeBreakpointHandle hBreakpoint, HwDbgCodeAddress *pCodeAddressOut)
- HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgGetKernelBinary (const HwDbgContextHandle hDebugContext, const void **ppBinaryOut, size_t *pBinarySizeOut)
- HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgGetDispatchedKernelName (const HwDbgContextHandle hDebugContext, const char **ppKernelNameOut)
- HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgGetLoadedSegmentDescriptors (HwDbgLoaderSegmentDescriptor *pSegmentDescriptorListOut, size_t *pSegmentDescriptorCountOut)
- HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgGetActiveWavefronts (const HwDbgContextHandle hDebugContext, const HwDbgWavefrontInfo **ppWavefrontInfoOut, uint32_t *pNumWavefrontsOut)
- HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgReadMemory (const HwDbgContextHandle hDebugContext, const uint32_t memoryRegion, const HwDbgDim3 workGroupId, const HwDbgDim3 workItemId, const size_t offset, const size_t numBytesToRead, void *pMemOut, size_t *pNumBytesOut)
- HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgBreakAll (const HwDbgContextHandle hDebugContext)
- HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgKillAll (const HwDbgContextHandle hDebugContext)
- HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgCreateDataBreakpoint (HwDbgContextHandle hDebugContext, const HwDbgDataBreakpointInfo breakpointInfo, HwDbgDataBreakpointHandle *pDataBreakpointOut)
- HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgDeleteDataBreakpoint (HwDbgContextHandle hDebugContext, HwDbgDataBreakpointHandle hDataBreakpoint)
- HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgDeleteAllDataBreakpoints (HwDbgContextHandle hDebugContext)
- HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgGetDataBreakpointInfo (const HwDbgContextHandle hDebugContext, const HwDbgDataBreakpointHandle hDataBreakpoint, HwDbgDataBreakpointInfo *pDataBreakpointInfoOut)

5.1.1 Detailed Description

The AMD GPU Kernel Debugging API to implement device kernel debugging on AMD Graphics Core Next (GCN) GPUs.

Copyright (c) 2015-2016 Advanced Micro Devices, Inc. All rights reserved.

Author

AMD Developer Tools

5.1.2 Macro Definition Documentation

5.1.2.1 #define AMDGPUDEBUG_VERSION_BUILD 194

The AMD GPU Debug API build number.

Definition at line 83 of file AMDGPUDebug.h.

5.1.2.2 #define AMDGPUDEBUG_VERSION_MAJOR 1

The AMD GPU Debug API major version.

Definition at line 79 of file AMDGPUDebug.h.

5.1.2.3 #define AMDGPUDEBUG_VERSION_MINOR 4

The AMD GPU Debug API minor version.

Definition at line 81 of file AMDGPUDebug.h.

5.1.2.4 #define HWDBG_API_CALL

The API calling convention on linux.

Definition at line 75 of file AMDGPUDebug.h.

5.1.2.5 #define HWDBG_API_ENTRY

No symbol visibility control for GCC older than 4.0.

Definition at line 72 of file AMDGPUDebug.h.

5.1.2.6 #define HWDBG_WAVEFRONT_SIZE 64

The maximum number of lanes in a wavefront for the GPU device.

Definition at line 86 of file AMDGPUDebug.h.

5.1.3 Typedef Documentation

5.1.3.1 typedef uint64_t HwDbgCodeAddress

The code location type (in bytes).

Definition at line 219 of file AMDGPUDebug.h.

5.1.3.2 typedef void* HwDbgCodeBreakpointHandle

A unique handle for a code breakpoint (returned by HwDbgCreateCodeBreakpoint).

Definition at line 228 of file AMDGPUDebug.h.

5.1.3.3 typedef void* HwDbgContextHandle

A unique handle for the kernel debug context (returned by HwDbgBeginDebugContext).

Definition at line 225 of file AMDGPUDebug.h.

5.1.3.4 typedef void* HwDbgDataBreakpointHandle

A unique handle for a data breakpoint (returned by HwDbgCreateDataBreakpoint).

Warning

This is not yet supported

Definition at line 232 of file AMDGPUDebug.h.

5.1.3.5 typedef void(* HwDbgLoggingCallback) (void *pUserData, const HwDbgLogType type, const char *const pMessage)

The user provided logging callback function to be registered.

This function will be called when the message with the type registered by the user is generated by the library.

Parameters

in	*pUserData*	The pointer specified by the user during registration
in	*type*	The type of log message being passed back
in	*pMessage*	The log message being passed back

Definition at line 393 of file AMDGPUDebug.h.

5.1.3.6 typedef uint32_t HwDbgWavefrontAddress

The hardware wavefront location type.

Definition at line 222 of file AMDGPUDebug.h.

5.1.4 Enumeration Type Documentation

5.1.4.1 enum HwDbgAPIType

The enumeration values of possible driver software stacks supported by the library

Enumerator

HWDBG_API_HSA the library is built for HSA software stack

Definition at line 156 of file AMDGPUDebug.h.

5.1.4.2 enum HwDbgBehaviorType

The enumerated bitfield values of supported behavior, the flags can be used internally to optimize behavior

Enumerator

HWDBG_BEHAVIOR_NONE Default flag, used to debug GPU dispatches

HWDBG_BEHAVIOR_DISABLE_DISPATCH_DEBUGGING Disable GPU dispatch debugging. However this behavior mode allows extraction of kernel binaries and breakpoint management. Allowed API calls are HwDbg[Begin or End]DebugContext, HwDbgGetKernelBinary, HwDbg[*CodeBreakpoint*] and Hw↩ Dbg[*DataBreakpoint*]

Definition at line 339 of file AMDGPUDebug.h.

5.1.4.3 enum HwDbgBreakpointType

The enumeration values of possible breakpoint types supported by the library.

Warning

This is not yet supported

Enumerator

HWDBG_BREAKPOINT_TYPE_NONE no breakpoint type

HWDBG_BREAKPOINT_TYPE_CODE instruction-based breakpoint type

HWDBG_BREAKPOINT_TYPE_DATA memory-based or data breakpoint type

Definition at line 163 of file AMDGPUDebug.h.

5.1.4.4 enum HwDbgCommand

The list of debugger commands for the HwDbgContinueEvent API to advance to the next state in the GPU debug engine.

Enumerator

HWDBG_COMMAND_CONTINUE resume the device execution

Definition at line 150 of file AMDGPUDebug.h.

5.1.4.5 enum HwDbgDataBreakpointMode

The list of possible access modes of data breakpoints supported.

Warning

This is not yet supported

Enumerator

HWDBG_DATABREAKPOINT_MODE_READ read operations only

HWDBG_DATABREAKPOINT_MODE_NONREAD write or atomic operations only

HWDBG_DATABREAKPOINT_MODE_ATOMIC atomic operations only

HWDBG_DATABREAKPOINT_MODE_ALL read, write or atomic operations

Definition at line 182 of file AMDGPUDebug.h.

5.1.4.6 enum HwDbgEventType

The enumeration values of possible event types returned by the HwDbgWaitForEvent API.

Enumerator

HWDBG_EVENT_INVALID an invalid event

HWDBG_EVENT_TIMEOUT has reached the user timeout value

HWDBG_EVENT_POST_BREAKPOINT has reached a breakpoint

HWDBG_EVENT_END_DEBUGGING has completed kernel execution

Definition at line 172 of file AMDGPUDebug.h.

5.1.4.7 enum HwDbgLoaderCodeObjectStorageType

The list of code object storage types supported by the loader.

Enumerator

HWDBG_LOADER_CODE_OBJECT_STORAGE_TYPE_NONE Loaded memory segment is not backed by any code object (anonymous), as the case would be with BSS (uninitialized data).

HWDBG_LOADER_CODE_OBJECT_STORAGE_TYPE_FILE Loaded memory segment is backed by the code object that is stored in the file.

HWDBG_LOADER_CODE_OBJECT_STORAGE_TYPE_MEMORY Loaded memory segment is backed by the code object that is stored in the memory.

Definition at line 198 of file AMDGPUDebug.h.

5.1.4.8 enum HwDbgLogType

The enumerated bitfield values of supported logging message types

Enumerator

HWDBG_LOG_TYPE_NONE do not register for any message
HWDBG_LOG_TYPE_ASSERT register for assert messages
HWDBG_LOG_TYPE_ERROR register for error messages
HWDBG_LOG_TYPE_TRACE register for trace messages
HWDBG_LOG_TYPE_MESSAGE register for generic messages
HWDBG_LOG_TYPE_ALL register for all messages

Definition at line 373 of file AMDGPUDebug.h.

5.1.4.9 enum HwDbgStatus

The enumeration values of the possible return status from the provided API.

Warning

Not all the enum values are supported currently

Enumerator

HWDBG_STATUS_SUCCESS the API was executed successfully
HWDBG_STATUS_ERROR a debugger internal error occurred
HWDBG_STATUS_DEVICE_ERROR the GPU device does not support debugging
HWDBG_STATUS_DRIVER_ERROR the driver is not compatible with the API
HWDBG_STATUS_DUPLICATE_BREAKPOINT a duplicate breakpoint is detected
HWDBG_STATUS_INVALID_ADDRESS_ALIGNMENT invalid address alignment was provided
HWDBG_STATUS_INVALID_HANDLE an invalid debug context handle was provided
HWDBG_STATUS_INVALID_PARAMETER invalid input arguments were provided
HWDBG_STATUS_NULL_POINTER expected a non NULL input argument
HWDBG_STATUS_OUT_OF_RANGE_ADDRESS out of range address was provided
HWDBG_STATUS_OUT_OF_MEMORY failed to allocate memory
HWDBG_STATUS_OUT_OF_RESOURCES ran out of hardware resources (for data breakpoints)
HWDBG_STATUS_REGISTRATION_ERROR started debugging on more than one application process
HWDBG_STATUS_UNDEFINED an undefined operation was detected (i.e. an incorrect call order)
HWDBG_STATUS_UNSUPPORTED the API has not been implemented
HWDBG_STATUS_NOT_INITIALIZED HwDbgInit has not been called
HWDBG_STATUS_INVALID_BEHAVIOR_STATE The debug context was created with unsupported behavior flags for the API

Definition at line 93 of file AMDGPUDebug.h.

5.1.5 Function Documentation

5.1.5.1 HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgBeginDebugContext (const HwDbgState *state,* HwDbgContextHandle * *pDebugContextOut*)

Mark the start debugging of a kernel dispatch.

This function should be called right before the execution of the kernel to be debugged (such as within the pre-dispatch callback function). Only one kernel dispatch should be between HwDbgBeginDebugContext and HwDbg↩EndDebugContext. Only one process can be debugged at a time in the system.

Parameters

in	*state*	specifies the input debug state
out	*pDebugContextOut*	returns the handle that identifies the particular kernel debug context

Returns

HwDbgStatus

Return values

HWDBG_STATUS_SUCCESS	On success
HWDBG_STATUS_DEVICE_ERROR	If the device does not support debugging
HWDBG_STATUS_ERROR	If an internal error occurs (check the log output for details)
HWDBG_STATUS_NOT_INITIALIZED	If called prior to a HwDbgInit call
HWDBG_STATUS_NULL_POINTER	If the input argument is NULL
HWDBG_STATUS_OUT_OF_MEMORY	If fail to allocate necessary memory
HWDBG_STATUS_REGISTRATION_ERROR	If more than 1 debug process is detected
HWDBG_STATUS_UNSUPPORTED	If the API has not been implemented

See also

HwDbgEndDebugContext

5.1.5.2 HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgBreakAll (const HwDbgContextHandle *hDebugContext*)

Break kernel execution of all active wavefronts for a kernel dispatch.

Can be called at any time after a HwDbgBeginDebugContext call.

Parameters

in	*hDebugContext*	specifies the context handle received from HwDbgBeginDebugContext API

Returns

HwDbgStatus

Return values

HWDBG_STATUS_SUCCESS	On success
HWDBG_STATUS_ERROR	If an internal error occurs (check the log output for details)
HWDBG_STATUS_INVALID_BEHAVIOR	If the context behavior flags are invalid
HWDBG_STATUS_INVALID_HANDLE	If the input hDebugContext is invalid
HWDBG_STATUS_NOT_INITIALIZED	If called prior to a HwDbgInit call
HWDBG_STATUS_UNSUPPORTED	If the API has not been implemented

5.1.5.3 HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgContinueEvent (HwDbgContextHandle *hDebugContext,* const HwDbgCommand *command*)

Continue to the next operation (resume device execution, run to the next breakpoint).

This is performed after receiving an event from HwDbgWaitForEvent. This is an asynchronous call, subsequent calls are undefined until the next HwDbgWaitEvent call.

Parameters

in	*hDebugContext*	specifies the context handle received from HwDbgBeginDebugContext API
in	*command*	specifies the debugger command to execute next

Returns

HwDbgStatus

Return values

HWDBG_STATUS_SUCCESS	On success
HWDBG_STATUS_ERROR	If an internal error occurs (check the log output for details)
HWDBG_STATUS_INVALID_BEHAVIOR	If the context behavior flags are invalid
HWDBG_STATUS_INVALID_HANDLE	If the input hDebugContext is invalid
HWDBG_STATUS_INVALID_PARAMETER	If the command argument is invalid
HWDBG_STATUS_NOT_INITIALIZED	If called prior to a HwDbgInit call
HWDBG_STATUS_UNDEFINED	If the kernel has completed execution
HWDBG_STATUS_UNSUPPORTED	If the API has not been implemented

See also

HwDbgWaitForEvent

5.1.5.4 HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgCreateCodeBreakpoint (HwDbgContextHandle *hDebugContext,* const HwDbgCodeAddress *codeAddress,* HwDbgCodeBreakpointHandle * *pBreakpointOut*)

Create a breakpoint at a specified program counter.

Parameters

in	*hDebugContext*	specifies the context handle received from HwDbgBeginDebugContext API
in	*codeAddress*	specifies the byte offset into the ISA binary indicating where to set the breakpoint. This has to be 4-byte aligned for AMD GPUs.
out	*pBreakpointOut*	returns the handle of the newly created instruction-based breakpoint. It is valid for use anywhere after creation. However, it is undefined to change the breakpoint state outside the HwDbgWaitForEvent/ HwDbgContinueEvent pair associated with the kernel dispatch that the breakpoint was created for

Returns

HwDbgStatus

Return values

HWDBG_STATUS_SUCCESS	On success
HWDBG_STATUS_ERROR	If the codeAddress is invalid (not 4-byte aligned or out of range) or has been inserted before
HWDBG_STATUS_INVALID_BEHAVIOR	If the context behavior flags are invalid
HWDBG_STATUS_INVALID_HANDLE	If the input hDebugContext is invalid
HWDBG_STATUS_NOT_INITIALIZED	If called prior to a HwDbgInit call
HWDBG_STATUS_NULL_POINTER	If the input argument is NULL
HWDBG_STATUS_UNSUPPORTED	If the API has not been implemented

See also

HwDbgDeleteCodeBreakpoint, HwDbgDeleteAllCodeBreakpoints, HwDbgGetCodeBreakpointAddress

5.1.5.5 HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgCreateDataBreakpoint (HwDbgContextHandle *hDebugContext,* const HwDbgDataBreakpointInfo *breakpointInfo,* HwDbgDataBreakpointHandle * *pDataBreakpointOut*)

Create a data breakpoint.

Warning

This is not yet supported

Parameters

`in`	*hDebugContext*	specifies the context handle received from HwDbgBeginDebugContext API
`in`	*breakpointInfo*	specifies the structure containing information where to set the data breakpoint
`out`	*pDataBreakpointOut*	returns the handle of the newly created data breakpoint. It is valid for use anywhere after creation. However, it is undefined to change the breakpoint state outside the HwDbgWaitForEvent/ HwDbgContinueEvent pair associated with the shader dispatch that the breakpoint was created for

Returns

HwDbgStatus

Return values

HWDBG_STATUS_SUCCESS	On success
HWDBG_STATUS_ERROR	If an internal error occurs (check the log output for details)
HWDBG_STATUS_INVALID_BEHAVIOR	If the context behavior flags are invalid
HWDBG_STATUS_INVALID_HANDLE	If the input hDebugContext is invalid
HWDBG_STATUS_NOT_INITIALIZED	If called prior to a HwDbgInit call

Return values

HWDBG_STATUS_NULL_POINTER	If the input argument or address is NULL
HWDBG_STATUS_OUT_OF_RESOURCES	If cannot be created due to hw limits
HWDBG_STATUS_UNSUPPORTED	If the API has not been implemented

See also

HwDbgDeleteDataBreakpoint, HwDbgDeleteAllDataBreakpoints, HwDbgGetDataBreakpointInfo

5.1.5.6 HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgDeleteAllCodeBreakpoints (HwDbgContextHandle *hDebugContext*)

Delete all instruction-based breakpoints.

Parameters

in	*hDebugContext*	specifies the context handle received from HwDbgBeginDebugContext API

Returns

HwDbgStatus

Return values

HWDBG_STATUS_SUCCESS	On success
HWDBG_STATUS_ERROR	If an internal error occurs (check the log output for details)
HWDBG_STATUS_INVALID_BEHAVIOR	If the context behavior flags are invalid
HWDBG_STATUS_INVALID_HANDLE	If the input hDebugContext is invalid
HWDBG_STATUS_NOT_INITIALIZED	If called prior to a HwDbgInit call
HWDBG_STATUS_UNSUPPORTED	If the API has not been implemented

See also

HwDbgCreateCodeBreakpoint, HwDbgDeleteCodeBreakpoint, HwDbgGetCodeBreakpointAddress

5.1.5.7 HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgDeleteAllDataBreakpoints (HwDbgContextHandle *hDebugContext*)

Delete all data breakpoints.

Warning

This is not yet supported

After this call, all data breakpoint handles created prior for the debug context will be invalid.

Parameters

in	*hDebugContext*	specifies the context handle received from HwDbgBeginDebugContext API

Returns

HwDbgStatus

Return values

HWDBG_STATUS_SUCCESS	On success
HWDBG_STATUS_ERROR	If an internal error occurs (check the log output for details)
HWDBG_STATUS_INVALID_BEHAVIOR	If the context behavior flags are invalid
HWDBG_STATUS_INVALID_HANDLE	If the input hDebugContext is invalid
HWDBG_STATUS_NOT_INITIALIZED	If called prior to a HwDbgInit call
HWDBG_STATUS_UNSUPPORTED	If the API has not been implemented

See also

HwDbgCreateDataBreakpoint, HwDbgDeleteDataBreakpoint, HwDbgGetDataBreakpointInfo

5.1.5.8 HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgDeleteCodeBreakpoint (HwDbgContextHandle *hDebugContext,* HwDbgCodeBreakpointHandle *hBreakpoint*)

Delete a instruction-based breakpoint.

Parameters

in	*hDebugContext*	specifies the context handle received from HwDbgBeginDebugContext API
in	*hBreakpoint*	specifies the breakpoint handle. The handle is invalid after this call and may be returned in future calls to HwDbgCreateCodeBreakpoint

Returns

HwDbgStatus

Return values

HWDBG_STATUS_SUCCESS	On success
HWDBG_STATUS_ERROR	If breakpoint handle is invalid or contains an invalid code address
HWDBG_STATUS_INVALID_BEHAVIOR	If the context behavior flags are invalid
HWDBG_STATUS_INVALID_HANDLE	If the input hDebugContext is invalid
HWDBG_STATUS_NOT_INITIALIZED	If called prior to a HwDbgInit call
HWDBG_STATUS_UNSUPPORTED	If the API has not been implemented

See also

HwDbgCreateCodeBreakpoint, HwDbgDeleteAllCodeBreakpoints, HwDbgGetCodeBreakpointAddress

5.1.5.9 HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgDeleteDataBreakpoint (HwDbgContextHandle *hDebugContext,* HwDbgDataBreakpointHandle *hDataBreakpoint*)

Delete a data breakpoint.

Warning

This is not yet supported

Parameters

in	*hDebugContext*	specifies the context handle received from HwDbgBeginDebugContext API
in	*hDataBreakpoint*	specifies the data breakpoint handle. The handle is invalid after this call and may be returned in future calls to HwDbgCreateCodeBreakpoint

Returns

HwDbgStatus

Return values

HWDBG_STATUS_SUCCESS	On success
HWDBG_STATUS_ERROR	If an internal error occurs (check the log output for details)
HWDBG_STATUS_INVALID_BEHAVIOR	If the context behavior flags are invalid
HWDBG_STATUS_INVALID_HANDLE	If the input hDebugContext is invalid
HWDBG_STATUS_NOT_INITIALIZED	If called prior to a HwDbgInit call
HWDBG_STATUS_UNSUPPORTED	If the API has not been implemented

See also

HwDbgCreateDataBreakpoint, HwDbgDeleteAllDataBreakpoints, HwDbgGetDataBreakpointInfo

5.1.5.10 HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgEndDebugContext (HwDbgContextHandle *hDebugContext*)

Mark the end debugging of a kernel dispatch.

This function must be called after the kernel has complete execution. Only one kernel dispatch should be between HwDbgBeginDebugContext and HwDbgEndDebugContext. Only one process can be debugged at a time in the system.

Parameters

in	*hDebugContext*	specifies the context handle received from HwDbgBeginDebugContext API. If it is NULL, then all sessions in flight will be terminated and deleted

Returns

HwDbgStatus

Return values

HWDBG_STATUS_SUCCESS	On success
HWDBG_STATUS_INVALID_HANDLE	If hDebugContext is an invalid handle
HWDBG_STATUS_NOT_INITIALIZED	If called prior to a HwDbgInit call
HWDBG_STATUS_UNDEFINED	If kernel execution has not yet completed
HWDBG_STATUS_UNSUPPORTED	If the API has not been implemented

See also

HwDbgBeginDebugContext

5.1.5.11 HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgGetActiveWavefronts (const HwDbgContextHandle *hDebugContext,* const HwDbgWavefrontInfo ** *ppWavefrontInfoOut,* uint32_t * *pNumWavefrontsOut*)

Retrieve the list of active wavefronts for the kernel dispatch in the GPU device.

Must only be called after receiving a HWDBG_EVENT_POST_BREAKPOINT event from HwDbgWaitForEvent API.

Parameters

in	*hDebugContext*	specifies the context handle received from HwDbgBeginDebugContext API
out	*ppWavefrontInfoOut*	returns a pointer to HwDbgWavefrontInfo structures. It contains the work-group ids, work- item ids, code adress, etc for each wavefront
out	*pNumWavefrontsOut*	returns the number of active wavefronts

Returns

HwDbgStatus

Return values

HWDBG_STATUS_SUCCESS	On success
HWDBG_STATUS_ERROR	If an internal error occurs (check the log output for details)
HWDBG_STATUS_INVALID_BEHAVIOR	If the context behavior flags are invalid
HWDBG_STATUS_INVALID_HANDLE	If the input hDebugContext is invalid
HWDBG_STATUS_NOT_INITIALIZED	If called prior to a HwDbgInit call
HWDBG_STATUS_NULL_POINTER	If the ppWaveInfoOut is NULL

Return values

HWDBG_STATUS_UNDEFINED	If it is called after not receiving a HWDBG_EVENT_POST_BREAKPOINT event
HWDBG_STATUS_UNSUPPORTED	If the API has not been implemented

5.1.5.12 HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgGetAPIType (HwDbgAPIType * *pAPITypeOut*)

Retrieve the driver API type of the loaded library.

This function can be called prior to a HwDbgInit call.

Parameters

out	*pAPITypeOut*	returns the API type of the library

Returns

HwDbgStatus

Return values

HWDBG_STATUS_SUCCESS	On success
HWDBG_STATUS_NULL_POINTER	If the input argument is NULL
HWDBG_STATUS_UNSUPPORTED	If the API is not yet implemented

5.1.5.13 HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgGetAPIVersion (uint32_t * *pVersionMajorOut,* uint32_t * *pVersionMinorOut,* uint32_t * *pVersionBuildOut*)

Retrieve the library version (major, minor and build) number.

This function can be called prior to a HwDbgInit call.

Parameters

out	*pVersionMajorOut*	returns the API version major number
out	*pVersionMinorOut*	returns API version minor number
out	*pVersionBuildOut*	returns API build number

Returns

HwDbgStatus

Return values

HWDBG_STATUS_SUCCESS	On success

Return values

HWDBG_STATUS_NULL_POINTER	If an input argument is NULL
HWDBG_STATUS_UNSUPPORTED	If the API is not yet implemented

5.1.5.14 HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgGetCodeBreakpointAddress (const HwDbgContextHandle *hDebugContext*, const HwDbgCodeBreakpointHandle *hBreakpoint*, HwDbgCodeAddress * *pCodeAddressOut*)

Retrieve the code location from an instruction-based breakpoint handle.

Parameters

`in`	*hDebugContext*	specifies the context handle received from HwDbgBeginDebugContext API
`in`	*hBreakpoint*	specifies the breakpoint handle
`out`	*pCodeAddressOut*	returns the code address (program counter)

Returns

HwDbgStatus
HWDBG_STATUS_SUCCESS On success

Return values

HWDBG_STATUS_ERROR	If an internal error occurs (check the log output for details)
HWDBG_STATUS_INVALID_BEHAVIOR	If the context behavior flags are invalid
HWDBG_STATUS_INVALID_HANDLE	If the input hDebugContext is invalid
HWDBG_STATUS_NOT_INITIALIZED	If called prior to a HwDbgInit call
HWDBG_STATUS_NULL_POINTER	If the input argument is NULL
HWDBG_STATUS_UNSUPPORTED	If the API has not been implemented

See also

HwDbgCreateCodeBreakpoint, HwDbgDeleteCodeBreakpoint, HwDbgDeleteAllCodeBreakpoints

5.1.5.15 HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgGetDataBreakpointInfo (const HwDbgContextHandle *hDebugContext*, const HwDbgDataBreakpointHandle *hDataBreakpoint*, HwDbgDataBreakpointInfo * *pDataBreakpointInfoOut*)

Retrieve the data breakpoint information from a data breakpoint handle.

Warning

This is not yet supported

Parameters

in	*hDebugContext*	specifies the context handle received from HwDbgBeginDebugContext API
in	*hDataBreakpoint*	specifies the data breakpoint handle
out	*pDataBreakpointInfoOut*	returns a structure containing information of the data breakpoint

Return values

HWDBG_STATUS_NOT_INITIALIZED	If called prior to a HwDbgInit call

Returns

HwDbgStatus
HWDBG_STATUS_SUCCESS On success

Return values

HWDBG_STATUS_ERROR	If an internal error occurs (check the log output for details)
HWDBG_STATUS_INVALID_BEHAVIOR	If the context behavior flags are invalid
HWDBG_STATUS_INVALID_HANDLE	If the input hDebugContext is invalid
HWDBG_STATUS_NULL_POINTER	If the input argument is NULL
HWDBG_STATUS_UNSUPPORTED	If the API has not been implemented

See also

HwDbgCreateDataBreakpoint, HwDbgDeleteDataBreakpoint, HwDbgDeleteAllDataBreakpoints

5.1.5.16 HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgGetDispatchedKernelName (const HwDbgContextHandle *hDebugContext*, const char ** *ppKernelNameOut*)

Retrieve the dispatched kernel name.

Parameters

in	*hDebugContext*	specifies the context handle received from HwDbgBeginDebugContext API
out	*ppKernelNameOut*	returns a pointer to a null-terminated character array The lifetime of the character array is within the debug context (i.e. after HwDbgBeginDebugContext call until the HwDbgEndDebugContext call)

Returns

HwDbgStatus

Return values

HWDBG_STATUS_SUCCESS	On success
HWDBG_STATUS_DRIVER_ERROR	If the retrieved kernel name is NULL

Return values

HWDBG_STATUS_ERROR	If an internal error occurs (check the log output for details)
HWDBG_STATUS_INVALID_BEHAVIOR	If the context behavior flags are invalid
HWDBG_STATUS_INVALID_HANDLE	If the input hDebugContext is invalid
HWDBG_STATUS_NULL_POINTER	If the input argument is NULL
HWDBG_STATUS_NOT_INITIALIZED	If called prior to a HwDbgInit call
HWDBG_STATUS_UNSUPPORTED	If the API has not been implemented

5.1.5.17 HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgGetKernelBinary (const HwDbgContextHandle *hDebugContext,* const void ** *ppBinaryOut,* size_t * *pBinarySizeOut*)

Retrieve the kernel binary (in ELF) of the kernel dispatch.

For HSA, the binary is the loaded and relocated code object. The binary contains the debugging information (in DWARF) from high level source to ISA (can be multiple level of DWARFs such as one DWARF to represent the mapping from a high level kernel source to BRIG and another DWARF to represent the mapping from BRIG to ISA).

Note

Refer to the following two documentation for more information:

1. HSA Application Binary Interface - AMD GPU Architecture document for the complete ABI.
2. HSA Debug Information document for the HSA DWARF extension

Parameters

`in`	*hDebugContext*	specifies the context handle received from HwDbgBeginDebugContext API
`out`	*ppBinaryOut*	returns a pointer to a buffer containing the binary kernel code object The lifetime of the buffer is within the debug context (i.e. after HwDbgBeginDebugContext call until the HwDbgEndDebugContext call)
`out`	*pBinarySizeOut*	returns the binary size in bytes

Returns

HwDbgStatus

Return values

HWDBG_STATUS_SUCCESS	On success
HWDBG_STATUS_DRIVER_ERROR	If the retrieved kernel binary is NULL or the binary size is 0
HWDBG_STATUS_ERROR	If an internal error occurs (check the log output for details)
HWDBG_STATUS_INVALID_BEHAVIOR	If the context behavior flags are invalid
HWDBG_STATUS_INVALID_HANDLE	If the input hDebugContext is invalid
HWDBG_STATUS_NOT_INITIALIZED	If called prior to a HwDbgInit call
HWDBG_STATUS_NULL_POINTER	If the input argument is NULL
HWDBG_STATUS_UNSUPPORTED	If the API has not been implemented

5.1.5.18 HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgGetLoadedSegmentDescriptors (HwDbgLoaderSegmentDescriptor * *pSegmentDescriptorListOut,* size_t * *pSegmentDescriptorCountOut*)

Query the loaded memory segment descriptors, or the total number of loaded memory segment descriptors. If pSegmentDescriptorListOut is nullptr, then the segment count is returned. This function is intended to be called twice, once to get the number of loaded segments and once to populate the segments.

Parameters

out	*pSegmentDescriptorListOut*	Application managed buffer, returns a list of the loaded segments
out	*pSegmentDescriptorCountOut*	Application managed buffer, returns the number of loaded segments

Returns

HwDbgStatus

Return values

HWDBG_STATUS_SUCCESS	On success
HWDBG_STATUS_ERROR	If an internal error occurs (check the log output for details)
HWDBG_STATUS_INVALID_PARAMETER	If both input parameters are null
HWDBG_STATUS_NOT_INITIALIZED	If called prior to a HwDbgInit call
HWDBG_STATUS_UNSUPPORTED	If the functionality is not supported

5.1.5.19 HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgInit (void * *pApiTable*)

Initialize the GPU debug engine.

This function should be called right after the debugged process starts. For hsa, this is in the HSA Runtime's OnLoad callback.

Parameters

in	*pApiTable*	Used by HSA: Pass in the pointer to the hsa api table provided by the HSA Runtime's OnLoad callback. Can be NULL (won't support full DBE functionality).

Returns

HwDbgStatus

Return values

HWDBG_STATUS_SUCCESS	On success
HWDBG_STATUS_ERROR	If called multiple times without a corresponding HwDbgShutDown
HWDBG_STATUS_OUT_OF_MEMORY	If fail to allocate necessary memory

See also

HwDbgShutDown

5.1.5.20 HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgKillAll (const HwDbgContextHandle *hDebugContext*)

Terminate the kernel dispatch execution.

Can be called at any time after a HwDbgBeginDebugContext call. Can be called multiple times to terminate a large kernel dispatch.

Parameters

in	*hDebugContext*	specifies the context handle received from HwDbgBeginDebugContext API

Returns

HwDbgStatus

Return values

HWDBG_STATUS_SUCCESS	On success
HWDBG_STATUS_ERROR	If an internal error occurs (check the log output for details)
HWDBG_STATUS_INVALID_BEHAVIOR	If the context behavior flags are invalid
HWDBG_STATUS_INVALID_HANDLE	If the input hDebugContext is invalid
HWDBG_STATUS_NOT_INITIALIZED	If called prior to a HwDbgInit call
HWDBG_STATUS_UNSUPPORTED	If the API has not been implemented

5.1.5.21 HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgReadMemory (const HwDbgContextHandle *hDebugContext*, const uint32_t *memoryRegion*, const HwDbgDim3 *workGroupId*, const HwDbgDim3 *workItemId*, const size_t *offset*, const size_t *numBytesToRead*, void * *pMemOut*, size_t * *pNumBytesOut*)

Read data from a memory region.

Warning

Only private memory region (IMR_Scratch = 1) is currently supported.

Must only be called after receiving a HWDBG_EVENT_POST_BREAKPOINT event from HwDbgWaitForEvent API.

Parameters

in	*hDebugContext*	specifies the context handle received from HwDbgBeginDebugContext API
in	*memoryRegion*	specifies the target memory region to read from. This should be set to an enum value stored in DW_AT_HSA_isa_memory_region attribute of DW_TAG_variable tag in ISA DWARF.
in	*workGroupId*	specifies the work-group id of interest (from HwDbgGetActiveWavefronts)

Parameters

in	*workItemId*	specifies the work-item id of interest (from HwDbgGetActiveWavefronts)
in	*offset*	specifies a byte offset for the logical location that should be retrieved. On GPU, this must be a multiple of 4 bytes (align on a DWORD boundary)
in	*numBytesToRead*	specifies the number of bytes to retrieve On GPU, this must be a multiple of 4 bytes
out	*pMemOut*	returns a pointer to a memory chunk of at least "numBytesToRead" bytes long
out	*pNumBytesOut*	returns the number of bytes written into pMemOut

Returns

HwDbgStatus

Return values

HWDBG_STATUS_SUCCESS	On success
HWDBG_STATUS_INVALID_BEHAVIOR	If the context behavior flags are invalid
HWDBG_STATUS_INVALID_HANDLE	If the input hDebugContext is invalid
HWDBG_STATUS_NOT_INITIALIZED	If called prior to a HwDbgInit call
HWDBG_STATUS_NULL_POINTER	If an input argument is NULL
HWDBG_STATUS_UNSUPPORTED	If the API has not been implemented

5.1.5.22 HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgSetLoggingCallback (uint32_t *types*, HwDbgLoggingCallback *pCallback*, void * *pUserData*)

Register a logging callback function.

Extra diagnostics output about the operation of the AMD GPU Debug API may be enabled by registering a client callback function through this API.

This function can be called prior to a HwDbgInit call.

Parameters

in	*types*	specifies the logging message types to be registered (a combination of HwDbgLogType enum value)
in	*pCallback*	specifies the logging callback function Set to a callback function function to enable logging Set to NULL to disable logging
in	*pUserData*	specifies a pointer to data that can be accessed by the user specified logging callback function

Returns

HwDbgStatus

Return values

HWDBG_STATUS_SUCCESS	If the callback can be set successfully
HWDBG_STATUS_ERROR	If an error is encountered
HWDBG_STATUS_UNSUPPORTED	If the API has not been implemented

5.1.5.23 HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgShutDown ( )

Shut down the GPU debug engine.

This function should be called before the debugged process ends. For hsa, this should be called right before calling the hsa_shut_down API.

Returns

HwDbgStatus

Return values

HWDBG_STATUS_SUCCESS	On success
HWDBG_STATUS_NOT_INITIALIZED	If called without a corresponding HwDbgInit

See also

HwDbgInit

5.1.5.24 HWDBG_API_ENTRY HwDbgStatus HWDBG_API_CALL HwDbgWaitForEvent (HwDbgContextHandle *hDebugContext*, const uint32_t *timeout*, HwDbgEventType * *pEventTypeOut*)

Wait on a debug event from the GPU device.

This is a synchronous function that will not return until it receives an event or reaches the specified timeout value.

Parameters

in	*hDebugContext*	specifies the context handle received from HwDbgBeginDebugContext API
in	*timeout*	specifies how long to wait in milliseconds before timing out
out	*pEventTypeOut*	The resulting event type

Returns

HwDbgStatus

Return values

HWDBG_STATUS_SUCCESS	On success
HWDBG_STATUS_ERROR	If an internal error occurs (check the log output for details)
HWDBG_STATUS_INVALID_BEHAVIOR	If the context behavior flags are invalid
HWDBG_STATUS_INVALID_HANDLE	If the input hDebugContext is invalid
HWDBG_STATUS_NOT_INITIALIZED	If called prior to a HwDbgInit call
HWDBG_STATUS_NULL_POINTER	If an input argument is NULL
HWDBG_STATUS_UNDEFINED	If the kernel has completed execution
HWDBG_STATUS_UNSUPPORTED	If the API has not been implemented

See also

HwDbgContinueEvent

Index

AMDGPUDEBUG_VERSION_BUILD
AMDGPUDebug.h, 18
AMDGPUDEBUG_VERSION_MAJOR
AMDGPUDebug.h, 18
AMDGPUDEBUG_VERSION_MINOR
AMDGPUDebug.h, 18
AMDGPUDebug.h, 15
AMDGPUDEBUG_VERSION_BUILD, 18
AMDGPUDEBUG_VERSION_MAJOR, 18
AMDGPUDEBUG_VERSION_MINOR, 18
HWDBG_API_CALL, 18
HWDBG_API_ENTRY, 18
HWDBG_API_HSA, 20
HWDBG_BEHAVIOR_DISABLE_DISPATCH_D↩
EBUGGING, 20
HWDBG_BEHAVIOR_NONE, 20
HWDBG_BREAKPOINT_TYPE_CODE, 20
HWDBG_BREAKPOINT_TYPE_DATA, 20
HWDBG_BREAKPOINT_TYPE_NONE, 20
HWDBG_COMMAND_CONTINUE, 20
HWDBG_DATABREAKPOINT_MODE_ALL, 21
HWDBG_DATABREAKPOINT_MODE_ATOMIC,
21
HWDBG_DATABREAKPOINT_MODE_NONRE↩
AD, 21
HWDBG_DATABREAKPOINT_MODE_READ, 21
HWDBG_EVENT_END_DEBUGGING, 21
HWDBG_EVENT_INVALID, 21
HWDBG_EVENT_POST_BREAKPOINT, 21
HWDBG_EVENT_TIMEOUT, 21
HWDBG_LOADER_CODE_OBJECT_STORAG↩
E_TYPE_FILE, 21
HWDBG_LOADER_CODE_OBJECT_STORAG↩
E_TYPE_MEMORY, 21
HWDBG_LOADER_CODE_OBJECT_STORAG↩
E_TYPE_NONE, 21
HWDBG_LOG_TYPE_ALL, 22
HWDBG_LOG_TYPE_ASSERT, 22
HWDBG_LOG_TYPE_ERROR, 22
HWDBG_LOG_TYPE_MESSAGE, 22
HWDBG_LOG_TYPE_NONE, 22
HWDBG_LOG_TYPE_TRACE, 22
HWDBG_STATUS_DEVICE_ERROR, 22
HWDBG_STATUS_DRIVER_ERROR, 22
HWDBG_STATUS_DUPLICATE_BREAKPOINT,
22
HWDBG_STATUS_ERROR, 22
HWDBG_STATUS_INVALID_ADDRESS_ALIG↩
NMENT, 22
HWDBG_STATUS_INVALID_BEHAVIOR_STA↩
TE, 22
HWDBG_STATUS_INVALID_HANDLE, 22
HWDBG_STATUS_INVALID_PARAMETER, 22
HWDBG_STATUS_NOT_INITIALIZED, 22
HWDBG_STATUS_NULL_POINTER, 22
HWDBG_STATUS_OUT_OF_MEMORY, 22
HWDBG_STATUS_OUT_OF_RANGE_ADDRE↩
SS, 22
HWDBG_STATUS_OUT_OF_RESOURCES, 22
HWDBG_STATUS_REGISTRATION_ERROR, 22
HWDBG_STATUS_SUCCESS, 22
HWDBG_STATUS_UNDEFINED, 22
HWDBG_STATUS_UNSUPPORTED, 22
HWDBG_WAVEFRONT_SIZE, 18
HwDbgAPIType, 19
HwDbgBeginDebugContext, 22
HwDbgBehaviorType, 20
HwDbgBreakAll, 23
HwDbgBreakpointType, 20
HwDbgCodeAddress, 18
HwDbgCodeBreakpointHandle, 18
HwDbgCommand, 20
HwDbgContextHandle, 19
HwDbgContinueEvent, 24
HwDbgCreateCodeBreakpoint, 24
HwDbgCreateDataBreakpoint, 25
HwDbgDataBreakpointHandle, 19
HwDbgDataBreakpointMode, 20
HwDbgDeleteAllCodeBreakpoints, 26
HwDbgDeleteAllDataBreakpoints, 26
HwDbgDeleteCodeBreakpoint, 27
HwDbgDeleteDataBreakpoint, 28
HwDbgEndDebugContext, 28
HwDbgEventType, 21
HwDbgGetAPIType, 30
HwDbgGetAPIVersion, 30
HwDbgGetActiveWavefronts, 29
HwDbgGetCodeBreakpointAddress, 31
HwDbgGetDataBreakpointInfo, 31
HwDbgGetDispatchedKernelName, 32
HwDbgGetKernelBinary, 33
HwDbgGetLoadedSegmentDescriptors, 33
HwDbgInit, 34
HwDbgKillAll, 35
HwDbgLoaderCodeObjectStorageType, 21
HwDbgLogType, 21
HwDbgLoggingCallback, 19
HwDbgReadMemory, 35

HwDbgSetLoggingCallback, 36
HwDbgShutDown, 37
HwDbgStatus, 22
HwDbgWaitForEvent, 37
HwDbgWavefrontAddress, 19

behaviorFlags
HwDbgState, 11
breakpointType
HwDbgWavefrontInfo, 12

codeAddress
HwDbgWavefrontInfo, 12
codeObjectStorageOffset
HwDbgLoaderSegmentDescriptor, 9
codeObjectStorageSize
HwDbgLoaderSegmentDescriptor, 9
codeObjectStorageType
HwDbgLoaderSegmentDescriptor, 9

dataBreakpointHandle
HwDbgWavefrontInfo, 13
dataBreakpointMode
HwDbgDataBreakpointInfo, 7
dataSize
HwDbgDataBreakpointInfo, 7
device
HwDbgLoaderSegmentDescriptor, 10

executable
HwDbgLoaderSegmentDescriptor, 10
executionMask
HwDbgWavefrontInfo, 13

HWDBG_API_CALL
AMDGPUDebug.h, 18
HWDBG_API_ENTRY
AMDGPUDebug.h, 18
HWDBG_API_HSA
AMDGPUDebug.h, 20
HWDBG_BEHAVIOR_DISABLE_DISPATCH_DEBU↩
GGING
AMDGPUDebug.h, 20
HWDBG_BEHAVIOR_NONE
AMDGPUDebug.h, 20
HWDBG_BREAKPOINT_TYPE_CODE
AMDGPUDebug.h, 20
HWDBG_BREAKPOINT_TYPE_DATA
AMDGPUDebug.h, 20
HWDBG_BREAKPOINT_TYPE_NONE
AMDGPUDebug.h, 20
HWDBG_COMMAND_CONTINUE
AMDGPUDebug.h, 20
HWDBG_DATABREAKPOINT_MODE_ALL
AMDGPUDebug.h, 21
HWDBG_DATABREAKPOINT_MODE_ATOMIC
AMDGPUDebug.h, 21
HWDBG_DATABREAKPOINT_MODE_NONREAD
AMDGPUDebug.h, 21
HWDBG_DATABREAKPOINT_MODE_READ
AMDGPUDebug.h, 21
HWDBG_EVENT_END_DEBUGGING
AMDGPUDebug.h, 21
HWDBG_EVENT_INVALID
AMDGPUDebug.h, 21
HWDBG_EVENT_POST_BREAKPOINT
AMDGPUDebug.h, 21
HWDBG_EVENT_TIMEOUT
AMDGPUDebug.h, 21
HWDBG_LOADER_CODE_OBJECT_STORAGE_TY↩
PE_FILE
AMDGPUDebug.h, 21
HWDBG_LOADER_CODE_OBJECT_STORAGE_TY↩
PE_MEMORY
AMDGPUDebug.h, 21
HWDBG_LOADER_CODE_OBJECT_STORAGE_TY↩
PE_NONE
AMDGPUDebug.h, 21
HWDBG_LOG_TYPE_ALL
AMDGPUDebug.h, 22
HWDBG_LOG_TYPE_ASSERT
AMDGPUDebug.h, 22
HWDBG_LOG_TYPE_ERROR
AMDGPUDebug.h, 22
HWDBG_LOG_TYPE_MESSAGE
AMDGPUDebug.h, 22
HWDBG_LOG_TYPE_NONE
AMDGPUDebug.h, 22
HWDBG_LOG_TYPE_TRACE
AMDGPUDebug.h, 22
HWDBG_STATUS_DEVICE_ERROR
AMDGPUDebug.h, 22
HWDBG_STATUS_DRIVER_ERROR
AMDGPUDebug.h, 22
HWDBG_STATUS_DUPLICATE_BREAKPOINT
AMDGPUDebug.h, 22
HWDBG_STATUS_ERROR
AMDGPUDebug.h, 22
HWDBG_STATUS_INVALID_ADDRESS_ALIGNMENT
AMDGPUDebug.h, 22
HWDBG_STATUS_INVALID_BEHAVIOR_STATE
AMDGPUDebug.h, 22
HWDBG_STATUS_INVALID_HANDLE
AMDGPUDebug.h, 22
HWDBG_STATUS_INVALID_PARAMETER
AMDGPUDebug.h, 22
HWDBG_STATUS_NOT_INITIALIZED
AMDGPUDebug.h, 22
HWDBG_STATUS_NULL_POINTER
AMDGPUDebug.h, 22
HWDBG_STATUS_OUT_OF_MEMORY
AMDGPUDebug.h, 22
HWDBG_STATUS_OUT_OF_RANGE_ADDRESS
AMDGPUDebug.h, 22
HWDBG_STATUS_OUT_OF_RESOURCES
AMDGPUDebug.h, 22
HWDBG_STATUS_REGISTRATION_ERROR

AMDGPUDebug.h, 22
HWDBG_STATUS_SUCCESS
AMDGPUDebug.h, 22
HWDBG_STATUS_UNDEFINED
AMDGPUDebug.h, 22
HWDBG_STATUS_UNSUPPORTED
AMDGPUDebug.h, 22
HWDBG_WAVEFRONT_SIZE
AMDGPUDebug.h, 18
HwDbgAPIType
AMDGPUDebug.h, 19
HwDbgBeginDebugContext
AMDGPUDebug.h, 22
HwDbgBehaviorType
AMDGPUDebug.h, 20
HwDbgBreakAll
AMDGPUDebug.h, 23
HwDbgBreakpointType
AMDGPUDebug.h, 20
HwDbgCodeAddress
AMDGPUDebug.h, 18
HwDbgCodeBreakpointHandle
AMDGPUDebug.h, 18
HwDbgCommand
AMDGPUDebug.h, 20
HwDbgContextHandle
AMDGPUDebug.h, 19
HwDbgContinueEvent
AMDGPUDebug.h, 24
HwDbgCreateCodeBreakpoint
AMDGPUDebug.h, 24
HwDbgCreateDataBreakpoint
AMDGPUDebug.h, 25
HwDbgDataBreakpointHandle
AMDGPUDebug.h, 19
HwDbgDataBreakpointInfo, 7
dataBreakpointMode, 7
dataSize, 7
pAddress, 7
HwDbgDataBreakpointMode
AMDGPUDebug.h, 20
HwDbgDeleteAllCodeBreakpoints
AMDGPUDebug.h, 26
HwDbgDeleteAllDataBreakpoints
AMDGPUDebug.h, 26
HwDbgDeleteCodeBreakpoint
AMDGPUDebug.h, 27
HwDbgDeleteDataBreakpoint
AMDGPUDebug.h, 28
HwDbgDim3, 8
x, 8
y, 8
z, 8
HwDbgEndDebugContext
AMDGPUDebug.h, 28
HwDbgEventType
AMDGPUDebug.h, 21
HwDbgGetAPIType
AMDGPUDebug.h, 30
HwDbgGetAPIVersion
AMDGPUDebug.h, 30
HwDbgGetActiveWavefronts
AMDGPUDebug.h, 29
HwDbgGetCodeBreakpointAddress
AMDGPUDebug.h, 31
HwDbgGetDataBreakpointInfo
AMDGPUDebug.h, 31
HwDbgGetDispatchedKernelName
AMDGPUDebug.h, 32
HwDbgGetKernelBinary
AMDGPUDebug.h, 33
HwDbgGetLoadedSegmentDescriptors
AMDGPUDebug.h, 33
HwDbgInit
AMDGPUDebug.h, 34
HwDbgKillAll
AMDGPUDebug.h, 35
HwDbgLoaderCodeObjectStorageType
AMDGPUDebug.h, 21
HwDbgLoaderSegmentDescriptor, 9
codeObjectStorageOffset, 9
codeObjectStorageSize, 9
codeObjectStorageType, 9
device, 10
executable, 10
pCodeObjectStorageBase, 10
pSegmentBase, 10
segmentSize, 10
HwDbgLogType
AMDGPUDebug.h, 21
HwDbgLoggingCallback
AMDGPUDebug.h, 19
HwDbgReadMemory
AMDGPUDebug.h, 35
HwDbgSetLoggingCallback
AMDGPUDebug.h, 36
HwDbgShutDown
AMDGPUDebug.h, 37
HwDbgState, 11
behaviorFlags, 11
pDevice, 11
pPacket, 11
packetId, 11
HwDbgStatus
AMDGPUDebug.h, 22
HwDbgWaitForEvent
AMDGPUDebug.h, 37
HwDbgWavefrontAddress
AMDGPUDebug.h, 19
HwDbgWavefrontInfo, 12
breakpointType, 12
codeAddress, 12
dataBreakpointHandle, 13
executionMask, 13
pOtherData, 13
wavefrontAddress, 13

workGroupId, 13
workItemId, 13

pAddress
HwDbgDataBreakpointInfo, 7
pCodeObjectStorageBase
HwDbgLoaderSegmentDescriptor, 10
pDevice
HwDbgState, 11
pOtherData
HwDbgWavefrontInfo, 13
pPacket
HwDbgState, 11
pSegmentBase
HwDbgLoaderSegmentDescriptor, 10
packetId
HwDbgState, 11

segmentSize
HwDbgLoaderSegmentDescriptor, 10

wavefrontAddress
HwDbgWavefrontInfo, 13
workGroupId
HwDbgWavefrontInfo, 13
workItemId
HwDbgWavefrontInfo, 13

x
HwDbgDim3, 8

y
HwDbgDim3, 8

z
HwDbgDim3, 8