h2d PCI Stream

Host API Guide

Table of Contents

h2d PCI Stream

1 Introduction

This library is the host side implementation of the Host-to-DSP PCI Stream communications protocol. The h2d pci stream allows the applications engineer to implement a many-streams architecture between the dsps and the host with the help of simple stream IO commands.

This streaming system is based on the DSP/BIOS Issue-Reclaim model. For a full explanation of this model please consult the Mango DSP Developer's Startup Guide; see the chapter on SIO Data Communications.

The main commands are PCI_STREAM_devInit (see page 13), PCI_STREAM_create (see page 10) ,PCI_STREAM_issue (see page 15) and PCI_STREAM_reclaim (see page 16). The PCI_STREAM_create (see page 10), PCI_STREAM_issue (see page 15) and PCI_STREAM_reclaim (see page 16) are modeled after the DSP BIOS functions SIO_create, SIO_issue, and SIO_reclaim, respectively.

PCI_STREAM_devInit (see page 13) is a host specific function to allow for the creation of different target DSPs.

PCI Streams use interrupts in both directions to prevent both the DSP and the Host from having to poll to wait for data. Enabling the DSP-to-host interrupts must be done by the application using the MANGOBIOS_isrConnect function. Please refer to the examples to see how to use it. This function will set the appropriate bits in the HDCR of the DSPs to enable DSP to host interrupts.

The isr_event parameter is architecture dependent.

On Linux, isr_event is the file descriptor of the handle. This is because interrupts work through the polling function on POSIX based systems, and the polling function needs the file descriptor. On Windows, isr_event is an event that must be created, and registered with an ISR using the MangoBIOS_isrConnect function.

The other parameters are self explanatory; please refer to examples.

2 Symbol Reference

Files

File	Description
h2d_pci_streamExp.h (see page 58)	Copyright 2005 MangoDSP, Inc., all rights reserved. Software and information contained herein is confidential and proprietary to MangoDSP, and any unauthorized copying, dissemination, or use is strictly prohibited. FILENAME: h2d_pci_streamExp.h GENERAL DESCRIPTION: h2d_pci_stream library exported api declarations DATE CREATED AUTHOR

Functions

Function	Description
h2d_pci_stream_Get_Version (see page 9)	Gets MANGOBIOS_version_t
PCI_STREAM_create (see page 10)	Creates a h2d_pci_stream channel
PCI_STREAM_ctrl (see page 11)	This is function PCI_STREAM_ctrl.
PCI_STREAM_delete (see page 12)	Deletes a h2d_pci_stream channel
PCI_STREAM_devInit (see page 13)	Creates a h2d_pci_stream target device
PCI_STREAM_devShutdown (see page 14)	Deletes a h2d_pci_stream target device
PCI_STREAM_issue (see page 15)	Issues a buffer to a stream
PCI_STREAM_reclaim (see page 16)	Reclaims a buffer from a stream
PCI_STREAM_select (see page 17)	Tests an array of channel's for data ready status

Macros

Macro	Description
__H2D_PCI_STREAMEXP_H__ (see page 31)	This is macro __H2D_PCI_STREAMEXP_H__.
ATOMIC (see page 32)	This is macro ATOMIC.
ATOMIC_CREATE (see page 33)	This is macro ATOMIC_CREATE.
ATOMIC_INC (see page 34)	This is macro ATOMIC_INC.
EVENT (see page 35)	This is macro EVENT.
PIPE (see page 36)	This is macro PIPE.
PIPE_CREATE (see page 37)	This is macro PIPE_CREATE.
PIPE_DESTROY (see page 38)	This is macro PIPE_DESTROY.
PIPE_PEND (see page 39)	This is macro PIPE_PEND.
PIPE_READ (see page 40)	This is macro PIPE_READ.
PIPE_WRITE (see page 41)	This is macro PIPE_WRITE.
SEM_CREATE (see page 42)	This is macro SEM_CREATE.
SEM_DESTROY (see page 43)	This is macro SEM_DESTROY.
SEM_GIVE (see page 44)	This is macro SEM_GIVE.
SEM_TAKE (see page 45)	This is macro SEM_TAKE.
SEM_TRY_TAKE (see page 46)	This is macro SEM_TRY_TAKE.
SEMAPHORE (see page 47)	This is macro SEMAPHORE.
THREAD (see page 48)	This is macro THREAD.
THREAD_CREATE (see page 49)	This is macro THREAD_CREATE.
THREAD_WAIT_EXIT (see page 50)	This is macro THREAD_WAIT_EXIT.
WAIT_FOR_PIPES (see page 51)	This is macro WAIT_FOR_PIPES.
WAKER (see page 52)	This is macro WAKER.
WAKER_CREATE (see page 53)	This is macro WAKER_CREATE.
WAKER_DESTROY (see page 54)	This is macro WAKER_DESTROY.
WAKER_GIVE (see page 55)	This is macro WAKER_GIVE.
WAKER_TAKE (see page 56)	This is macro WAKER_TAKE.

Types

Type	Description
h2d_pci_channel_params_t (⧉ see page 19)	This is type h2d_pci_channel_params_t.
h2d_pci_channel_t (⧉ see page 20)	This is type h2d_pci_channel_t.
h2d_pci_device_params_t (⧉ see page 21)	This is type h2d_pci_device_params_t.
h2d_pci_device_t (⧉ see page 22)	This is type h2d_pci_device_t.
PCI_H2D_INTERRUPT_t (⧉ see page 23)	This is type PCI_H2D_INTERRUPT_t.
PCI_READ_t (⧉ see page 24)	This is type PCI_READ_t.
PCI_STREAM_ptr_t (⧉ see page 25)	This is type PCI_STREAM_ptr_t.
PCI_WRITE_t (⧉ see page 26)	This is type PCI_WRITE_t.
pipe_t (⧉ see page 27)	This is type pipe_t.
QUE_Elem_t (⧉ see page 28)	This is type QUE_Elem_t.
waker_t (⧉ see page 29)	This is type waker_t.

Structs, Records, Enums

Struct, Record, Enum	Description
pipe_s (⧉ see page 5)	This is record pipe_s.
QUE_Elem_s (⧉ see page 6)	This is record QUE_Elem_s.
waker_s (⧉ see page 7)	This is record waker_s.

2.1 Structs, Records, Enums

Structs

Struct	Description
QUE_Elem_s (☑ see page 6)	This is record QUE_Elem_s.
waker_s (☑ see page 7)	This is record waker_s.

Unions

Union	Description
pipe_s (☑ see page 5)	This is record pipe_s.

2.1.1 pipe_s

```
union pipe_s {
  int _p[2];
};
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is record pipe_s.

2.1.2 QUE_Elem_s

```
struct QUE_Elem_s {
  struct QUE_Elem_s * next;
  struct QUE_Elem_s * prev;
};
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is record QUE_Elem_s.

2.1.3 waker_s

```
struct waker_s {
  int _p[2];
};
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is record waker_s.

2.2 Functions

Functions

Function	Description
h2d_pci_stream_Get_Version (see page 9)	Gets MANGOBIOS_version_t
PCI_STREAM_create (see page 10)	Creates a h2d_pci_stream channel
PCI_STREAM_ctrl (see page 11)	This is function PCI_STREAM_ctrl.
PCI_STREAM_delete (see page 12)	Deletes a h2d_pci_stream channel
PCI_STREAM_devInit (see page 13)	Creates a h2d_pci_stream target device
PCI_STREAM_devShutdown (see page 14)	Deletes a h2d_pci_stream target device
PCI_STREAM_issue (see page 15)	Issues a buffer to a stream
PCI_STREAM_reclaim (see page 16)	Reclaims a buffer from a stream
PCI_STREAM_select (see page 17)	Tests an array of channel's for data ready status

2.2.1 h2d_pci_stream_Get_Version

Gets MANGOBIOS_version_t

```
MANGOERROR_error_t h2d_pci_stream_Get_Version(MANGOBIOS_version_t * version);
```

File

h2d_pci_streamExp.h (see page 58)

Returns

Status

Return Values

Return Values	Description
MANGOERROR_SUCCESS	Success

Description

Gets version information for h2d_pci_stream Library

Remarks

None

Example

```
int errorCode;
MANGOBIOS_version_t version;

errorCode = h2d_pci_stream_Get_Version(
 &version
 );
```

2.2.2 PCI_STREAM_create

Creates a h2d_pci_stream channel

```
h2d_pci_channel_t * PCI_STREAM_create(h2d_pci_device_t * dev, char * name, int mode,
unsigned int bufsize, h2d_pci_channel_params_t * params);
```

File

h2d_pci_streamExp.h (see page 58)

Parameters

Parameters	Description
`h2d_pci_device_t * dev`	The device object
`char * name`	The name of the channel. This must be identical to the name used on the DSP.
`int mode`	PCI_STREAM_INPUT or PCI_STREAM_OUTPUT, see remarks
`unsigned int bufsize`	Maximum size in bytes of the chunks that this stream can transfer
`h2d_pci_channel_params_t * params`	The parameters for timeout and type

Returns

Pointer to the newly allocated h2d_pci_channel_t (see page 20)

NULL is returned if:

- params is NULL

- name is longer than MAX_CHANNEL_NAME_LENGTH

- Failed memory allocation

- an error occured with synchronization with the DSP (reset the DSP and host, and try again)

Description

Creates a h2d_pci_channel_t (see page 20) on 'dev'. This channel will connect to the corresponding channel created on the target dsp synchronizing on 'name' and 'mode'. See Remarks.

Remarks

Must be run only after DSP has performed device initialization. This occurs before the DSP reaches main().

Channels are created on the DSP side using SIO_create. 'name' needs to be identical to the name used on the dsp. This function will not initialize the channel with any empty buffers.

'bufsize' will be the maximum size of any buffer going through this channel. 'params' will determine the timeout of this channel, and the channel's transfer type (eg PCI_dsp_master..).

For a channel on the host to connect to a channel on the DSP, one side must use input, and the other side must use output. (On the DSP, the input is SIO_INPUT, and the output is SIO_OUTPUT)

Example

```
h2d_pci_channel_t * chan;
PCI_CHANNEL_PARAMS_t params;
params.type = PCI_dsp_master;
params.timeout = PCI_STREAM_FOREVER;
chan = PCI_STREAM_create(
 dev,
 "/dioPciDataChannel1",
 PCI_STREAM_OUTPUT,
 0x1000,
 &params
);
```

2.2.3 PCI_STREAM_ctrl

```
int PCI_STREAM_ctrl(h2d_pci_channel_t * chan, unsigned int cmd, void * args);
```

File

h2d_pci_streamExp.h (⧉ see page 58)

Description

This is function PCI_STREAM_ctrl.

2.2.4 PCI_STREAM_delete

Deletes a h2d_pci_stream channel

```
int PCI_STREAM_delete(h2d_pci_channel_t * chan);
```

File

h2d_pci_streamExp.h (see page 58)

Parameters

Parameters	Description
`h2d_pci_channel_t * chan`	The channel object

Returns

Status

Return Values

Return Values	Description
`PCI_OK`	Success
`PCI_ERROR`	'chan' is NULL

Description

Deletes a h2d_pci_channel_t (see page 20).

Remarks

None

Example

```
PCI_STREAM_delete(
 chan,
);
```

2.2.5 PCI_STREAM_devInit

Creates a h2d_pci_stream target device

```
h2d_pci_device_t * PCI_STREAM_devInit(h2d_pci_device_params_t * params);
```

File

h2d_pci_streamExp.h (see page 58)

Parameters

Parameters	Description
`h2d_pci_device_params_t * params`	The parameters for the target

Returns

Pointer to the newly allocated h2d_pci_device_t (see page 22)

NULL is returned if:

- params is NULL

- params->shared_memory_size is less than required for library

- Failed memory allocation

- params->fm_type is not PCI_FM_off

Description

Creates a h2d_pci_device_t (see page 22). This will be a future target for PCI_STREAM_create (see page 10) calls. h2d_pci_device_t (see page 22) holds all of the information for referring to a single DSP destination on a Mango DSP card.

Remarks

This can be called at any time. This does not require that the DSP is running to work.

Example

```
h2d_pci_device_t * dev;
PCI_DEVICE_PARAMS_t params;
params.card = &asb_handle;
params.dsp = 0;
params.fm_type = PCI_FM_off;
params.fm_event = NULL;
params.read_fxn = asb_handle.read_memory;
params.write_fxn = asb_handle.write_memory;
params.h2d_interrupt = asb_handle.h2d_interrupt;
params.shared_memory_start = 0x80000000;
params.shared_memory_size = 0x100;
dev = PCI_STREAM_devInit(
 &params
);
```

2.2.6 PCI_STREAM_devShutdown

Deletes a h2d_pci_stream target device

int `PCI_STREAM_devShutdown(h2d_pci_device_t * dev);`

File

h2d_pci_streamExp.h (see page 58)

Parameters

Parameters	Description
`h2d_pci_device_t * dev`	Pointer to h2d_pci_device_t (see page 22)

Returns

Status

Return Values

Return Values	Description
`MANGOERROR_SUCCESS`	Success
`MANGOERROR_INVALID_PARAMETER`	'dev' is NULL

Description

Deletes a h2d_pci_device_t (see page 22).

Remarks

None

Example

```
int errorCode;
errorCode = PCI_STREAM_devShutdown(
 dev
);
```

2.2.7 PCI_STREAM_issue

Issues a buffer to a stream

```
int PCI_STREAM_issue(h2d_pci_channel_t * chan, PCI_STREAM_ptr_t pbuf, unsigned int nmadus);
```

File

h2d_pci_streamExp.h (see page 58)

Parameters

Parameters	Description
`h2d_pci_channel_t * chan`	The channel object
`PCI_STREAM_ptr_t pbuf`	Pointer to the buffer structure
`unsigned int nmadus`	Size in bytes of the buffer

Returns

Status

Return Values

Return Values	Description
`PCI_OK`	Success
`PCI_ERROR`	'chan' is NULL 'pbuf'.local or 'pbuf'.pci is NULL Failed memory allocation

Description

Places a buffer in this stream's output que, and signals the DSP to update the same channel on the DSP. If there is a buffer issued on both ends, then a transaction takes place.

Remarks

Must be run only after DSP has performed device initialization. This occurs before the DSP reaches main().

Example

```
PCI_STREAM_ptr_t pbuf;
MANGOBIOS_memoryHandle_t handle;
MANGOBIOS_memoryAlloc(
 0x1000,
 &handle,
 &pbuf.pci,
 NULL
);
MANGOBIOS_memoryMap(
 handle,
 &pbuf.local,
 NULL
);
PCI_STREAM_issue(
 chan,
 pbuf,
 0x1000
);
```

2.2.8 PCI_STREAM_reclaim

Reclaims a buffer from a stream

```
int PCI_STREAM_reclaim(h2d_pci_channel_t * chan, PCI_STREAM_ptr_t * pbuf);
```

File

h2d_pci_streamExp.h (see page 58)

Parameters

Parameters	Description
`h2d_pci_channel_t * chan`	The channel object
`PCI_STREAM_ptr_t * pbuf`	Pointer to the buffer structure

Returns

Status

Return Values

Return Values	Description
`PCI_OK`	Success
`PCI_ERROR`	'chan' is NULL 'pbuf' is NULL No buffers are outstanding in this stream. ie: either no buffers have been issued, or every buffer issued has already been reclaimed.
`PCI_TIMEOUT`	No buffer was avaiable within the timeout set for this channel.

Description

Gets a buffer from this stream's output que if available. If there is no buffer available, then it pends for up to the timeout set for this channel.

Remarks

Must be run only after DSP has performed device initialization. This occurs before the DSP reaches main().

If the DSP side has not performed an SIO_create which connects to this channel, then the host will not receive any buffers, and either pend forever until the DSP creates the channel, and issues a buffer, or it will continously return a PCI_TIMEOUT

Example

```
int size = 0;
size = PCI_STREAM_reclaim(
 chan,
 &pbuf
);
if(size < 0) //error
```

2.2.9 PCI_STREAM_select

Tests an array of channel's for data ready status

```
int PCI_STREAM_select(h2d_pci_channel_t ** chans, int numChans, int timeout);
```

File

h2d_pci_streamExp.h (see page 58)

Parameters

Parameters	Description
`int numChans`	Number of channel's in the array. Each channel must be an initialized channel.
`int timeout`	PCI_STREAM_POLL or PCI_STREAM_FOREVER
`chan`	An array of channel pointers.

Returns

Bitmask of channel's that are ready. bitmask[0] corresponds to the 'chans'[0].

Description

Tests an array of channel's for data ready status. If a channel is ready, then a reclaim can be run without risk of pending.

Remarks

Must be run only after DSP has performed device initialization. This occurs before the DSP reaches main().

If the DSP side has not performed an SIO_create which connects to this channel, then the host will not receive any buffers, and either pend forever until the DSP creates the channel, and issues a buffer, or it will continously return a 0 if the timeout is PCI_STREAM_POLL.

This will return as soon as the first channel is ready.

Example

```
h2d_pci_channel_t * chans[2];
int mask = 0;
chans[0] = channel0;
chans[1] = channel1;
mask = PCI_STREAM_select(
 chans,
 2,
 PCI_STREAM_FOREVER
);
if(mask & 1) //channel0 has data
if(mask & 2) //channel1 has data
```

2.3 Types

Types

Type	Description
h2d_pci_channel_params_t (⧉ see page 19)	This is type h2d_pci_channel_params_t.
h2d_pci_channel_t (⧉ see page 20)	This is type h2d_pci_channel_t.
h2d_pci_device_params_t (⧉ see page 21)	This is type h2d_pci_device_params_t.
h2d_pci_device_t (⧉ see page 22)	This is type h2d_pci_device_t.
PCI_H2D_INTERRUPT_t (⧉ see page 23)	This is type PCI_H2D_INTERRUPT_t.
PCI_READ_t (⧉ see page 24)	This is type PCI_READ_t.
PCI_STREAM_ptr_t (⧉ see page 25)	This is type PCI_STREAM_ptr_t.
PCI_WRITE_t (⧉ see page 26)	This is type PCI_WRITE_t.
pipe_t (⧉ see page 27)	This is type pipe_t.
QUE_Elem_t (⧉ see page 28)	This is type QUE_Elem_t.
waker_t (⧉ see page 29)	This is type waker_t.

2.3.1 h2d_pci_channel_params_t

```
typedef struct {
  int timeout;
} h2d_pci_channel_params_t;
```

File

h2d_pci_streamExp.h (see page 58)

Members

Members	Description
`int timeout;`	when running PCI_STREAM_reclaim (see page 16), the timeout determines how long to wait before returning a timeout on the stream when there is no data - MANGOBIOS_FOREVER or MANGOBIOS_POLL

Description

This is type h2d_pci_channel_params_t.

2.3.2 h2d_pci_channel_t

```
typedef struct {
  QUE_Elem_t link;
  int chan_num;
  unsigned int remote_chan;
  SEMAPHORE mutex;
  SEMAPHORE connected;
  PIPE local_que_in, local_que_out;
  int mode;
  char name[MAX_CHANNEL_NAME_LENGTH];
  h2d_pci_device_t * dev;
  int timeout;
} h2d_pci_channel_t;
```

File

h2d_pci_streamExp.h (◰ see page 58)

Description

This is type h2d_pci_channel_t.

2.3.3 h2d_pci_device_params_t

```
typedef struct {
  void * card;
  int dsp;
  EVENT isr_event;
  PCI_READ_t read_fxn;
  PCI_WRITE_t write_fxn;
  PCI_H2D_INTERRUPT_t h2d_interrupt_fxn;
  unsigned int shared_memory_start;
  unsigned int shared_memory_size;
  int use_polling;
} h2d_pci_device_params_t;
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is type h2d_pci_device_params_t.

2.3.4 h2d_pci_device_t

```
typedef struct {
  void * card;
  int dsp;
  PCI_READ_t read_fxn;
  PCI_WRITE_t write_fxn;
  PCI_H2D_INTERRUPT_t h2d_interrupt_fxn;
  unsigned int db_sync_resp;
  unsigned int mb_sync_resp;
  unsigned int db_sync_resp_ack;
  unsigned int db_data_resp;
  unsigned int mb_data_resp;
  unsigned int db_data_resp_ack;
  unsigned int db_sync_init;
  unsigned int mb_sync_init;
  unsigned int db_sync_init_ack;
  unsigned int db_data_init;
  unsigned int mb_data_init;
  unsigned int db_data_init_ack;
  int db_s_resp_exp;
  int db_d_resp_exp;
  int db_s_init_ack_exp;
  int db_d_init_ack_exp;
  int db_s_init_exp;
  int db_d_init_exp;
  int db_s_resp_ack_exp;
  int db_d_resp_ack_exp;
  int use_polling;
  SEMAPHORE syncsem;
  SEMAPHORE datasem;
  THREAD thread;
  SEMAPHORE mutex;
  WAKER w_kill;
  WAKER w_wakeup[4];
  EVENT isr_event;
  QUE_Elem_t * channels;
} h2d_pci_device_t;
```

File

h2d_pci_streamExp.h (see page 58)

Members

Members	Description
`unsigned int db_sync_resp;`	from remote
`unsigned int mb_sync_resp;`	from remote
`unsigned int db_sync_resp_ack;`	to remote
`unsigned int db_data_resp;`	from remote
`unsigned int mb_data_resp;`	from remote
`unsigned int db_data_resp_ack;`	to remote
`unsigned int db_sync_init;`	to remote
`unsigned int mb_sync_init;`	to remote
`unsigned int db_sync_init_ack;`	from remote
`unsigned int db_data_init;`	to remote
`unsigned int mb_data_init;`	to remote
`unsigned int db_data_init_ack;`	from remote

Description

This is type h2d_pci_device_t.

2.3.5 PCI_H2D_INTERRUPT_t

```
typedef MANGOERROR_error_t (* PCI_H2D_INTERRUPT_t)(void * handle, int dsp);
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is type PCI_H2D_INTERRUPT_t.

2.3.6 PCI_READ_t

```
typedef MANGOERROR_error_t (* PCI_READ_t)(void * handle, int dsp, void * hst_adr, unsigned
int dsp_adr, unsigned int bytes_to_read);
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is type PCI_READ_t.

2.3.7 PCI_STREAM_ptr_t

```
typedef struct {
  void * local;
  unsigned int pci;
} PCI_STREAM_ptr_t;
```

File

h2d_pci_streamExp.h (⧉ see page 58)

Description

This is type PCI_STREAM_ptr_t.

2.3.8 PCI_WRITE_t

```
typedef MANGOERROR_error_t (* PCI_WRITE_t)(void * handle, int dsp, void * hst_adr, unsigned
int dsp_adr, unsigned int bytes_to_write);
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is type PCI_WRITE_t.

2.3.9 pipe_t

```
typedef union pipe_s {
  int _p[2];
} pipe_t;
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is type pipe_t.

2.3.10 QUE_Elem_t

```
typedef struct QUE_Elem_s {
  struct QUE_Elem_s * next;
  struct QUE_Elem_s * prev;
} QUE_Elem_t;
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is type QUE_Elem_t.

2.3.11 waker_t

```
typedef struct waker_s {
  int _p[2];
} waker_t;
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is type waker_t.

2.4 Macros

Macros

Macro	Description
__H2D_PCI_STREAMEXP_H__ (⧉ see page 31)	This is macro __H2D_PCI_STREAMEXP_H__.
ATOMIC (⧉ see page 32)	This is macro ATOMIC.
ATOMIC_CREATE (⧉ see page 33)	This is macro ATOMIC_CREATE.
ATOMIC_INC (⧉ see page 34)	This is macro ATOMIC_INC.
EVENT (⧉ see page 35)	This is macro EVENT.
PIPE (⧉ see page 36)	This is macro PIPE.
PIPE_CREATE (⧉ see page 37)	This is macro PIPE_CREATE.
PIPE_DESTROY (⧉ see page 38)	This is macro PIPE_DESTROY.
PIPE_PEND (⧉ see page 39)	This is macro PIPE_PEND.
PIPE_READ (⧉ see page 40)	This is macro PIPE_READ.
PIPE_WRITE (⧉ see page 41)	This is macro PIPE_WRITE.
SEM_CREATE (⧉ see page 42)	This is macro SEM_CREATE.
SEM_DESTROY (⧉ see page 43)	This is macro SEM_DESTROY.
SEM_GIVE (⧉ see page 44)	This is macro SEM_GIVE.
SEM_TAKE (⧉ see page 45)	This is macro SEM_TAKE.
SEM_TRY_TAKE (⧉ see page 46)	This is macro SEM_TRY_TAKE.
SEMAPHORE (⧉ see page 47)	This is macro SEMAPHORE.
THREAD (⧉ see page 48)	This is macro THREAD.
THREAD_CREATE (⧉ see page 49)	This is macro THREAD_CREATE.
THREAD_WAIT_EXIT (⧉ see page 50)	This is macro THREAD_WAIT_EXIT.
WAIT_FOR_PIPES (⧉ see page 51)	This is macro WAIT_FOR_PIPES.
WAKER (⧉ see page 52)	This is macro WAKER.
WAKER_CREATE (⧉ see page 53)	This is macro WAKER_CREATE.
WAKER_DESTROY (⧉ see page 54)	This is macro WAKER_DESTROY.
WAKER_GIVE (⧉ see page 55)	This is macro WAKER_GIVE.
WAKER_TAKE (⧉ see page 56)	This is macro WAKER_TAKE.

2.4.1 __H2D_PCI_STREAMEXP_H__

```
#define __H2D_PCI_STREAMEXP_H__
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is macro __H2D_PCI_STREAMEXP_H__.

2.4.2 ATOMIC

```
#define ATOMIC int
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is macro ATOMIC.

2.4.3 ATOMIC_CREATE

```
#define ATOMIC_CREATE(a) (a)
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is macro ATOMIC_CREATE.

2.4.4 ATOMIC_INC

```
#define ATOMIC_INC(a, val) {val = a; a++;}
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is macro ATOMIC_INC.

2.4.5 EVENT

```
#define EVENT int
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is macro EVENT.

2.4.6 PIPE

```
#define PIPE pipe_t
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is macro PIPE.

2.4.7 PIPE_CREATE

```
#define PIPE_CREATE(p) pipe((p)._p);
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is macro PIPE_CREATE.

2.4.8 PIPE_DESTROY

```
#define PIPE_DESTROY(p) {close(p._p[0]); close(p._p[1]);}
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is macro PIPE_DESTROY.

2.4.9 PIPE_PEND

```
#define PIPE_PEND(p, t, r) {struct timeval tv; fd_set rfds; FD_ZERO(&rfds);
FD_SET((p)._p[0], &rfds); tv.tv_sec = (t / 1000); tv.tv_usec = ((t % 1000) / 1000); r =
select((p)._p[0] + 1, &rfds, NULL, NULL, ((t) == MANGOBIOS_FOREVER) ? NULL : &tv); r = (r
== 1); }
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is macro PIPE_PEND.

2.4.10 PIPE_READ

```
#define PIPE_READ(p, d, s) read((p)._p[0], &d, s)
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is macro PIPE_READ.

2.4.11 PIPE_WRITE

```
#define PIPE_WRITE(p, d, s) write((p)._p[1], &d, s)
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is macro PIPE_WRITE.

2.4.12 SEM_CREATE

```
#define SEM_CREATE(e, v) sem_init(&(e), 0, (v))
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is macro SEM_CREATE.

2.4.13 SEM_DESTROY

```
#define SEM_DESTROY(e) sem_destroy(&(e))
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is macro SEM_DESTROY.

2.4.14 SEM_GIVE

```
#define SEM_GIVE(e) {while(sem_post(&(e)) != 0);}
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is macro SEM_GIVE.

2.4.15 SEM_TAKE

```
#define SEM_TAKE(e) {while(sem_wait(&(e)) != 0);}
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is macro SEM_TAKE.

2.4.16 SEM_TRY_TAKE

```
#define SEM_TRY_TAKE(e, ret) (ret = (WaitForSingleObject((e), 0) == WAIT_OBJECT_0))
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is macro SEM_TRY_TAKE.

2.4.17 SEMAPHORE

```
#define SEMAPHORE sem_t
```

File

h2d_pci_streamExp.h (⧉ see page 58)

Description

This is macro SEMAPHORE.

2.4.18 THREAD

```
#define THREAD pthread_t
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is macro THREAD.

2.4.19 THREAD_CREATE

```
#define THREAD_CREATE(t, f, p) pthread_create(&(t), NULL, f, (void*)p)
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is macro THREAD_CREATE.

2.4.20 THREAD_WAIT_EXIT

```
#define THREAD_WAIT_EXIT(t) pthread_join((t), NULL);
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is macro THREAD_WAIT_EXIT.

2.4.21 WAIT_FOR_PIPES

```
#define WAIT_FOR_PIPES(p, n, t, r) {int i; int high = 0; struct timeval tv; fd_set rfds;
FD_ZERO(&rfds); for(i = 0; i < (n); i++){ FD_SET((p)[i]->_p[0], &rfds); if((p)[i]->_p[0] >
high) high = (p)[i]->_p[0];} tv.tv_sec = (t / 1000); tv.tv_usec = ((t % 1000) / 1000); r =
select(high + 1, &rfds, NULL, NULL, ((t) == MANGOBIOS_FOREVER) ? NULL : &tv); r = (r >= 1);
}
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is macro WAIT_FOR_PIPES.

2.4.22 WAKER

```
#define WAKER waker_t
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is macro WAKER.

2.4.23 WAKER_CREATE

```
#define WAKER_CREATE(w) pipe((w)._p);
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is macro WAKER_CREATE.

2.4.24 WAKER_DESTROY

```
#define WAKER_DESTROY(w) {close((w)._p[0]); close((w)._p[1]);}
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is macro WAKER_DESTROY.

2.4.25 WAKER_GIVE

```
#define WAKER_GIVE(w) {char __junk; write((w)._p[1], &__junk, 1);}
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is macro WAKER_GIVE.

2.4.26 WAKER_TAKE

```
#define WAKER_TAKE(w) {char __junk; read((w)._p[0], &__junk, 1);}
```

File

h2d_pci_streamExp.h (see page 58)

Description

This is macro WAKER_TAKE.

2.5 Files

Files

File	Description
h2d_pci_streamExp.h (see page 58)	Copyright 2005 MangoDSP, Inc., all rights reserved. Software and information contained herein is confidential and proprietary to MangoDSP, and any unauthorized copying, dissemination, or use is strictly prohibited. FILENAME: h2d_pci_streamExp.h GENERAL DESCRIPTION: h2d_pci_stream library exported api declarations DATE CREATED AUTHOR

2.5.1 h2d_pci_streamExp.h

Copyright 2005 MangoDSP, Inc., all rights reserved. Software and information contained herein is confidential and proprietary to MangoDSP, and any unauthorized copying, dissemination, or use is strictly prohibited.

FILENAME: h2d_pci_streamExp.h

GENERAL DESCRIPTION: h2d_pci_stream library exported api declarations

DATE CREATED AUTHOR

Functions

Function	Description
h2d_pci_stream_Get_Version (⧉ see page 9)	Gets MANGOBIOS_version_t
PCI_STREAM_create (⧉ see page 10)	Creates a h2d_pci_stream channel
PCI_STREAM_ctrl (⧉ see page 11)	This is function PCI_STREAM_ctrl.
PCI_STREAM_delete (⧉ see page 12)	Deletes a h2d_pci_stream channel
PCI_STREAM_devInit (⧉ see page 13)	Creates a h2d_pci_stream target device
PCI_STREAM_devShutdown (⧉ see page 14)	Deletes a h2d_pci_stream target device
PCI_STREAM_issue (⧉ see page 15)	Issues a buffer to a stream
PCI_STREAM_reclaim (⧉ see page 16)	Reclaims a buffer from a stream
PCI_STREAM_select (⧉ see page 17)	Tests an array of channel's for data ready status

Macros

Macro	Description
__H2D_PCI_STREAMEXP_H__ (⧉ see page 31)	This is macro __H2D_PCI_STREAMEXP_H__.
ATOMIC (⧉ see page 32)	This is macro ATOMIC.
ATOMIC_CREATE (⧉ see page 33)	This is macro ATOMIC_CREATE.
ATOMIC_INC (⧉ see page 34)	This is macro ATOMIC_INC.
EVENT (⧉ see page 35)	This is macro EVENT.
PIPE (⧉ see page 36)	This is macro PIPE.
PIPE_CREATE (⧉ see page 37)	This is macro PIPE_CREATE.
PIPE_DESTROY (⧉ see page 38)	This is macro PIPE_DESTROY.
PIPE_PEND (⧉ see page 39)	This is macro PIPE_PEND.
PIPE_READ (⧉ see page 40)	This is macro PIPE_READ.
PIPE_WRITE (⧉ see page 41)	This is macro PIPE_WRITE.
SEM_CREATE (⧉ see page 42)	This is macro SEM_CREATE.
SEM_DESTROY (⧉ see page 43)	This is macro SEM_DESTROY.
SEM_GIVE (⧉ see page 44)	This is macro SEM_GIVE.
SEM_TAKE (⧉ see page 45)	This is macro SEM_TAKE.
SEM_TRY_TAKE (⧉ see page 46)	This is macro SEM_TRY_TAKE.
SEMAPHORE (⧉ see page 47)	This is macro SEMAPHORE.
THREAD (⧉ see page 48)	This is macro THREAD.
THREAD_CREATE (⧉ see page 49)	This is macro THREAD_CREATE.
THREAD_WAIT_EXIT (⧉ see page 50)	This is macro THREAD_WAIT_EXIT.
WAIT_FOR_PIPES (⧉ see page 51)	This is macro WAIT_FOR_PIPES.
WAKER (⧉ see page 52)	This is macro WAKER.
WAKER_CREATE (⧉ see page 53)	This is macro WAKER_CREATE.
WAKER_DESTROY (⧉ see page 54)	This is macro WAKER_DESTROY.
WAKER_GIVE (⧉ see page 55)	This is macro WAKER_GIVE.
WAKER_TAKE (⧉ see page 56)	This is macro WAKER_TAKE.

Structs

Struct	Description
QUE_Elem_s (see page 6)	This is record QUE_Elem_s.
waker_s (see page 7)	This is record waker_s.

Types

Type	Description
h2d_pci_channel_params_t (see page 19)	This is type h2d_pci_channel_params_t.
h2d_pci_channel_t (see page 20)	This is type h2d_pci_channel_t.
h2d_pci_device_params_t (see page 21)	This is type h2d_pci_device_params_t.
h2d_pci_device_t (see page 22)	This is type h2d_pci_device_t.
PCI_H2D_INTERRUPT_t (see page 23)	This is type PCI_H2D_INTERRUPT_t.
PCI_READ_t (see page 24)	This is type PCI_READ_t.
PCI_STREAM_ptr_t (see page 25)	This is type PCI_STREAM_ptr_t.
PCI_WRITE_t (see page 26)	This is type PCI_WRITE_t.
pipe_t (see page 27)	This is type pipe_t.
QUE_Elem_t (see page 28)	This is type QUE_Elem_t.
waker_t (see page 29)	This is type waker_t.

Unions

Union	Description
pipe_s (see page 5)	This is record pipe_s.

Index