Linux Utils Application Programming Interface (API) Reference Manual

linuxutils-d06

Generated by Doxygen 1.5.1

Mon Apr 20 11:45:33 2009

Contents

1 Linux Utils API Reference

1.1 Modules

The Linux Utils product contains several components:

- Contiguous Memory Manager
- EDMA Manager
- SDMA Manager
- VICP Manager

2 Linux Utils Application Programming Interface (API) Module Index

2.1 Linux Utils Application Programming Interface (API) Modules

Here is a list of all modules:

3 Linux Utils Application Programming Interface (API) Directory Hierarchy

3.1 Linux Utils Application Programming Interface (API) Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

4 Linux Utils Application Programming Interface (API) Data Structure Index

4.1 Linux Utils Application Programming Interface (API) Data Structures

Here are the data structures with brief descriptions:

5 Linux Utils Application Programming Interface (API) File Index

5.1 Linux Utils Application Programming Interface (API) File List

Here is a list of all files with brief descriptions:

6 Linux Utils Application Programming Interface (API) Page Index

6.1 Linux Utils Application Programming Interface (API) Related Pages

Here is a list of all related documentation pages:

7 Linux Utils Application Programming Interface (API) Module Documentation

7.1 Contiguous Memory Manager

7.1.1 Detailed Description

This is the API for the Contiguous Memory Manager.

Data Structures

- struct CMEM_AllocParams

  Parameters for CMEM_alloc(), CMEM_alloc2(), CMEM_allocPool(), CMEM_allocPool2(), CMEM_free().

- struct CMEM_BlockAttrs

Defines

- #define CMEM_VERSION 0x02300000U
- #define CMEM_WB 0x0100
- #define CMEM_INV 0x0200
- #define CMEM_HEAP 0x0400
- #define CMEM_POOL 0x0000
- #define CMEM_CACHED 0x0800
- #define CMEM_NONCACHED 0x0000
- #define CMEM_PHYS 0x1000
- #define CMEM_IOCALLOC 1
- #define CMEM_IOCALLOCHEAP 2
- #define CMEM_IOCFREE 3
- #define CMEM_IOCGETPHYS 4
- #define CMEM_IOCGETSIZE 5
- #define CMEM_IOCGETPOOL 6
- #define CMEM_IOCCACHE 7
- #define CMEM_IOCGETVERSION 8
- #define CMEM_IOCGETBLOCK 9
- #define CMEM_IOCREGUSER 10
- #define CMEM_IOCCACHEWBINV CMEM_IOCCACHE | CMEM_WB | CMEM_INV
- #define CMEM_IOCCACHEWB CMEM_IOCCACHE | CMEM_WB
- #define CMEM_IOCCACHEINV CMEM_IOCCACHE | CMEM_INV
- #define CMEM_IOCALLOCCACHED CMEM_IOCALLOC | CMEM_CACHED
- #define CMEM_IOCALLOCHEAPCACHED CMEM_IOCALLOCHEAP | CMEM_CACHED
- #define CMEM_IOCFREEHEAP CMEM_IOCFREE | CMEM_HEAP
- #define CMEM_IOCFREEPHYS CMEM_IOCFREE | CMEM_PHYS
- #define CMEM_IOCFREEHEAPPHYS CMEM_IOCFREE | CMEM_HEAP | CMEM_PHYS
- #define CMEM_IOCCMDMASK 0x000000ff

Functions

- int CMEM_init (void)

  Initialize the CMEM module. Must be called before other API calls.

- int CMEM_getPool (size_t size)

  Find the pool that best fits a given buffer size and has a buffer available.

- int CMEM_getPool2 (int blockid, size_t size)

Find the pool in memory block blockid that best fits a given buffer size and has a buffer available.

- void * CMEM_allocPool (int poolid, CMEM_AllocParams *params)

  Allocate memory from a specified pool.

- void * CMEM_allocPool2 (int blockid, int poolid, CMEM_AllocParams *params)

  Allocate memory from a specified pool in a specified memory block.

- void * CMEM_alloc (size_t size, CMEM_AllocParams *params)

  Allocate memory of a specified size.

- void * CMEM_alloc2 (int blockid, size_t size, CMEM_AllocParams *params)

  Allocate memory of a specified size from a specified memory block.

- void * CMEM_registerAlloc (unsigned long physp)

  Register shared usage of an already-allocated buffer.

- int CMEM_free (void *ptr, CMEM_AllocParams *params)

  Free a buffer previously allocated with CMEM_alloc()/CMEM_allocPool().

- int CMEM_unregister (void *ptr, CMEM_AllocParams *params)

  Unregister use of a buffer previously registered with CMEM_registerAlloc().

- unsigned long CMEM_getPhys (void *ptr)

  Get the physical address of a contiguous buffer.

- int CMEM_cacheWb (void *ptr, size_t size)

  Do a cache writeback of the block pointed to by `ptr/size`.

- int CMEM_cacheInv (void *ptr, size_t size)

  Do a cache invalidate of the block pointed to by `ptr/size`.

- int CMEM_cacheWbInv (void *ptr, size_t size)

  Do a cache writeback/invalidate of the block pointed to by `ptr/size`.

- int CMEM_getVersion (void)

  Retrieve version from CMEM driver.

- int CMEM_getBlock (unsigned long *pphys_base, size_t *psize)

  Retrieve memory block bounds from CMEM driver.

- int CMEM_getBlockAttrs (int blockid, CMEM_BlockAttrs *pattrs)
  Retrieve extended memory block attributes from CMEM driver.
- int CMEM_exit (void)
  Finalize the CMEM module.

Variables

- CMEM_AllocParams CMEM_DEFAULTPARAMS

7.1.2 Define Documentation

7.1.2.1 #define CMEM_VERSION 0x02300000U

7.1.2.2 #define CMEM_WB 0x0100

7.1.2.3 #define CMEM_INV 0x0200

7.1.2.4 #define CMEM_HEAP 0x0400

operation applies to heap

7.1.2.5 #define CMEM_POOL 0x0000

operation applies to a pool

7.1.2.6 #define CMEM_CACHED 0x0800

allocated buffer is cached

7.1.2.7 #define CMEM_NONCACHED 0x0000

allocated buffer is not cached

7.1.2.8 #define CMEM_PHYS 0x1000

7.1.2.9 #define CMEM_IOCALLOC 1

7.1.2.10 #define CMEM_IOCALLOCHEAP 2

7.1.2.11 #define CMEM_IOCFREE 3

7.1.2.12 #define CMEM_IOCGETPHYS 4

7.1.2.13 #define CMEM_IOCGETSIZE 5

7.1.2.14 #define CMEM_IOCGETPOOL 6

7.1.2.15 #define CMEM_IOCCACHE 7

7.1.2.16 #define CMEM_IOCGETVERSION 8

7.1.2.17 #define CMEM_IOCGETBLOCK 9

7.1.2.18 #define CMEM_IOCREGUSER 10

7.1.2.19 #define CMEM_IOCCACHEWBINV CMEM_IOCCACHE | CMEM_WB | CMEM_INV

7.1.2.20 #define CMEM_IOCCACHEWB CMEM_IOCCACHE | CMEM_WB

7.1.2.21 #define CMEM_IOCCACHEINV CMEM_IOCCACHE | CMEM_INV

7.1.2.22 #define CMEM_IOCALLOCCACHED CMEM_IOCALLOC | CMEM_CACHED

7.1.2.23 #define CMEM_IOCALLOCHEAPCACHED CMEM_IOCALLOCHEAP | CMEM_CACHED

7.1.2.24 #define CMEM_IOCFREEHEAP CMEM_IOCFREE | CMEM_HEAP

7.1.2.25 #define CMEM_IOCFREEPHYS CMEM_IOCFREE | CMEM_PHYS

7.1.2.26 #define CMEM_IOCFREEHEAPPHYS CMEM_IOCFREE | CMEM_HEAP | CMEM_PHYS

7.1.2.27 #define CMEM_IOCCMDMASK 0x000000ff

7.1.3 Function Documentation

7.1.3.1 int CMEM_init (void)

Initialize the CMEM module. Must be called before other API calls.

Returns:

0 for success or -1 for failure.

See also:

CMEM_exit

7.1.3.2 int CMEM_getPool (size_t *size*)

Find the pool that best fits a given buffer size and has a buffer available.

Parameters:

size The buffer size for which a pool is needed.

Returns:

A poolid that can be passed to CMEM_allocPool(), or -1 for error.

Precondition:

Must have called CMEM_init()

See also:

CMEM_allocPool()
CMEM_allocPool2()
CMEM_free()
CMEM_getPool2()

7.1.3.3 int CMEM_getPool2 (int *blockid*, size_t *size*)

Find the pool in memory block blockid that best fits a given buffer size and has a buffer available.

Parameters:

blockid Block number

size The buffer size for which a pool is needed.

Returns:

A poolid that can be passed to CMEM_allocPool2(), or -1 for error.

Precondition:

Must have called CMEM_init()

See also:

CMEM_allocPool()
CMEM_allocPool2()
CMEM_free()
CMEM_getPool()

7.1.3.4 void* CMEM_allocPool (int *poolid*, CMEM_AllocParams * *params*)

Allocate memory from a specified pool.

Parameters:

poolid The pool from which to allocate memory.

params Allocation parameters.

Remarks:

`params->type` is ignored - a pool will always be used.
`params->alignment` is unused, since pool buffers are already aligned to specific boundaries.

Returns:

A pointer to the allocated buffer, or NULL for failure.

Precondition:

Must have called CMEM_init()

See also:

CMEM_alloc()
CMEM_alloc2()
CMEM_allocPool2()
CMEM_registerAlloc()
CMEM_unregister()
CMEM_free()

7.1.3.5 void* CMEM_allocPool2 (int *blockid*, int *poolid*, CMEM_AllocParams * *params*)

Allocate memory from a specified pool in a specified memory block.

Parameters:

blockid The memory block from which to allocate.

poolid The pool from which to allocate memory.

params Allocation parameters.

Remarks:

`params->type` is ignored - a pool will always be used.
`params->alignment` is unused, since pool buffers are already aligned to specific boundaries.

Returns:

A pointer to the allocated buffer, or NULL for failure.

Precondition:

Must have called CMEM_init()

See also:

CMEM_alloc()
CMEM_alloc2()
CMEM_allocPool()
CMEM_registerAlloc()
CMEM_unregister()
CMEM_free()

7.1.3.6 void* CMEM_alloc (size_t *size*, CMEM_AllocParams * *params*)

Allocate memory of a specified size.

Parameters:

size The size of the buffer to allocate.

params Allocation parameters.

Remarks:

Used to allocate memory from either a pool or the heap. If doing a pool allocation, the pool that best fits the requested size will be selected. Use CMEM_allocPool() to allocate from a specific pool. Allocation will be cached or noncached, as specified by params. params->alignment valid only for heap allocation.

Returns:

A pointer to the allocated buffer, or NULL for failure.

Precondition:

Must have called CMEM_init()

See also:

CMEM_allocPool()
CMEM_allocPool2()
CMEM_alloc2()
CMEM_registerAlloc()
CMEM_unregister()
CMEM_free()

7.1.3.7 void* CMEM_alloc2 (int *blockid*, size_t *size*, CMEM_AllocParams * *params*)

Allocate memory of a specified size from a specified memory block.

Parameters:

blockid The memory block from which to allocate.

size The size of the buffer to allocate.

params Allocation parameters.

Remarks:

Used to allocate memory from either a pool or the heap. If doing a pool allocation, the pool that best fits the requested size will be selected. Use CMEM_allocPool() to allocate from a specific pool.
Allocation will be cached or noncached, as specified by params. params->alignment valid only for heap allocation.

Returns:

A pointer to the allocated buffer, or NULL for failure.

Precondition:

Must have called CMEM_init()

See also:

CMEM_allocPool()
CMEM_allocPool2()
CMEM_alloc()
CMEM_registerAlloc()
CMEM_unregister()
CMEM_free()

7.1.3.8 void* CMEM_registerAlloc (unsigned long *physp*)

Register shared usage of an already-allocated buffer.

Parameters:

physp Physical address of the already-allocated buffer.

Remarks:

Used to register the calling process for usage of an already-allocated buffer, for the purpose of shared usage of the buffer.
Allocation properties (such as cached/noncached or heap/pool) are inherited from original allocation call.

Returns:

A process-specific pointer to the allocated buffer, or NULL for failure.

Precondition:

Must have called some form of CMEM_alloc*()

See also:

CMEM_allocPool()
CMEM_allocPool2()
CMEM_alloc()
CMEM_free()
CMEM_unregister()

7.1.3.9 int CMEM_free (void * *ptr*, CMEM_AllocParams * *params*)

Free a buffer previously allocated with CMEM_alloc()/CMEM_allocPool().

Parameters:

ptr The pointer to the buffer.

params Allocation parameters.

Remarks:

Use the same CMEM_AllocParams as was used for the allocation. params->flags is "don't care". params->alignment is "don't care".

Returns:

0 for success or -1 for failure.

Precondition:

Must have called CMEM_init()

See also:

CMEM_alloc()
CMEM_alloc2()
CMEM_allocPool()
CMEM_allocPool2()
CMEM_registerAlloc()
CMEM_unregister()

7.1.3.10 int CMEM_unregister (void * *ptr*, CMEM_AllocParams * *params*)

Unregister use of a buffer previously registered with CMEM_registerAlloc().

Parameters:

ptr The pointer to the buffer.

params Allocation parameters.

Remarks:

Use the same CMEM_AllocParams as was used for the allocation. params->flags is "don't care". params->alignment is "don't care".

Returns:

0 for success or -1 for failure.

Precondition:

Must have called CMEM_init()

See also:

CMEM_alloc()
CMEM_alloc2()
CMEM_allocPool()
CMEM_allocPool2()
CMEM_registerAlloc()
CMEM_free()

7.1.3.11 unsigned long CMEM_getPhys (void * *ptr*)

Get the physical address of a contiguous buffer.

Parameters:

ptr The pointer to the buffer.

Returns:

The physical address of the buffer or 0 for failure.

Precondition:

Must have called CMEM_init()

7.1.3.12 int CMEM_cacheWb (void * *ptr*, size_t *size*)

Do a cache writeback of the block pointed to by `ptr/size`.

Parameters:

ptr Pointer to block to writeback

size Size in bytes of block to writeback.

Returns:

Success/failure boolean value

Precondition:

Must have called CMEM_init()

See also:

CMEM_cacheInv()
CMEM_cacheWbInv()

7.1.3.13 int CMEM_cacheInv (void * *ptr*, size_t *size*)

Do a cache invalidate of the block pointed to by `ptr/size`.

Parameters:

ptr Pointer to block to invalidate

size Size in bytes of block to invalidate

Returns:

Success/failure boolean value

Precondition:

Must have called CMEM_init()

See also:

CMEM_cacheWb()
CMEM_cacheWbInv()

7.1.3.14 int CMEM_cacheWbInv (void * *ptr*, size_t *size*)

Do a cache writeback/invalidate of the block pointed to by `ptr/size`.

Parameters:

ptr Pointer to block to writeback/invalidate

size Size in bytes of block to writeback/invalidate

Returns:

Success/failure boolean value

Precondition:

Must have called CMEM_init()

See also:

CMEM_cacheInv()
CMEM_cacheWb()

7.1.3.15 int CMEM_getVersion (void)

Retrieve version from CMEM driver.

Returns:

Installed CMEM driver's version number.

Precondition:

Must have called CMEM_init()

7.1.3.16 int CMEM_getBlock (unsigned long * *pphys_base*, size_t * *psize*)

Retrieve memory block bounds from CMEM driver.

Parameters:

pphys_base Pointer to storage for base physical address of CMEM's memory block

psize Pointer to storage for size of CMEM's memory block

Returns:

Success (0) or failure (-1).

Precondition:

Must have called CMEM_init()

See also:

CMEM_getBlockAttrs()

7.1.3.17 int CMEM_getBlockAttrs (int *blockid*, CMEM_BlockAttrs * *pattrs*)

Retrieve extended memory block attributes from CMEM driver.

Parameters:

blockid Block number

pattrs Pointer to CMEM_BlockAttrs struct

Returns:

Success (0) or failure (-1).

Remarks:

Currently this API returns the same values as CMEM_getBlock().

Precondition:

Must have called CMEM_init()

See also:

CMEM_getBlock()

7.1.3.18 int CMEM_exit (void)

Finalize the CMEM module.

Returns:

0 for success or -1 for failure.

Remarks:

After this function has been called, no other CMEM function may be called (unless CMEM is reinitialized).

Precondition:

Must have called CMEM_init()

See also:

CMEM_init()

7.1.4 Variable Documentation

7.1.4.1 CMEM_AllocParams CMEM_DEFAULTPARAMS

7.2 EDMA Manager

7.2.1 Detailed Description

This is the API for the EDMA manager.

Data Structures

- struct EDMA_requestDmaParams
  EDMAK ioctl input/output parameters.
- struct EDMA_releaseDmaParams
  EDMAK ioctl input parameters.

Defines

- #define EDMA_VERSION 0x01100000U
- #define EDMA_TCCANY 1001
  Values for dev_id parameter of EDMA_getResource().
- #define EDMA_TCCSYMM 1005
- #define EDMA_PARAMANY 1006
- #define EDMA_PARAMFIXEDEXACT 1007
- #define EDMA_PARAMFIXEDNOTEXACT 1008
- #define EDMA_EDMAANY 1003
- #define EDMA_QDMAANY 1004
- #define EDMA_QDMA0 512
- #define EDMA_QDMA1 EDMA_QDMA(1)
- #define EDMA_QDMA2 EDMA_QDMA(2)
- #define EDMA_QDMA3 EDMA_QDMA(3)
- #define EDMA_QDMA4 EDMA_QDMA(4)
- #define EDMA_QDMA5 EDMA_QDMA(5)
- #define EDMA_QDMA6 EDMA_QDMA(6)

- #define EDMA_QDMA7 EDMA_QDMA(7)
- #define EDMA_QDMA(n) (EDMA_QDMA0 + (n))

  macro used to translate from a QDMA channel # to the numerical range used by EDMA_getResource() & EDMA_freeResource() for representing a QDMA channel.

- #define EDMA_QDMA2NUM(qdma) (qdma - EDMA_QDMA0)

  macro to translate from EDMA_QDMA0 -> EDMA_QDMA7 namespace to actual QDMA number 0 -> 7.

Enumerations

- enum EDMA_commmands {
  EDMA_IOCREQUESTDMA = 1,
  EDMA_IOCRELEASEDMA,
  EDMA_IOCGETVERSION,
  EDMA_IOCGETBASEPHYSADDR,
  EDMA_IOCREGUSER }

  EDMAK ioctl() commands.

- enum EDMA_Status {
  EDMA_OK = 0,
  EDMA_EFAIL,
  EDMA_ENOCHANNEL,
  EDMA_ENOINIT }

  Status codes for EDMA API return values.

Functions

- int EDMA_init (void)

  EDMA initialization API.

- int EDMA_exit (void)

  EDMA finalization API.

- EDMA_Status EDMA_mapBaseAddress (void **pvirtAddr)

  Returns user space virtual address of EDMA base registers.

- EDMA_Status EDMA_getResource (int devId, int *tcc, int *channel, int *param, int nParams)

  Returns available logical channels.

- EDMA_Status EDMA_freeResource (int lch, int nParams)

  Frees previously requested logical channel.

- int EDMA_getVersion (void)

  Retrieve version from EDMA driver.

- EDMA_Status EDMA_registerResource (int lch)

  Registers calling process as a user of lch.

- EDMA_Status EDMA_unregister (int lch, int nParams)

  Unregisters calling process as a user of lch.

7.2.2 Define Documentation

7.2.2.1 #define EDMA_VERSION 0x01100000U

7.2.2.2 #define EDMA_TCCANY 1001

Values for dev_id parameter of EDMA_getResource().

7.2.2.3 #define EDMA_TCCSYMM 1005

7.2.2.4 #define EDMA_PARAMANY 1006

7.2.2.5 #define EDMA_PARAMFIXEDEXACT 1007

7.2.2.6 #define EDMA_PARAMFIXEDNOTEXACT 1008

7.2.2.7 #define EDMA_EDMAANY 1003

7.2.2.8 #define EDMA_QDMAANY 1004

7.2.2.9 #define EDMA_QDMA0 512

7.2.2.10 #define EDMA_QDMA1 EDMA_QDMA(1)

7.2.2.11 #define EDMA_QDMA2 EDMA_QDMA(2)

7.2.2.12 #define EDMA_QDMA3 EDMA_QDMA(3)

7.2.2.13 #define EDMA_QDMA4 EDMA_QDMA(4)

7.2.2.14 #define EDMA_QDMA5 EDMA_QDMA(5)

7.2.2.15 #define EDMA_QDMA6 EDMA_QDMA(6)

7.2.2.16 #define EDMA_QDMA7 EDMA_QDMA(7)

7.2.2.17 #define EDMA_QDMA(n) (EDMA_QDMA0 + (n))

macro used to translate from a QDMA channel # to the numerical range used by EDMA_getResource() & EDMA_freeResource() for representing a QDMA channel.

See also:

EDMA_QDMA2NUM()

7.2.2.18 #define EDMA_QDMA2NUM(qdma) (qdma - EDMA_QDMA0)

macro to translate from EDMA_QDMA0 -> EDMA_QDMA7 namespace to actual QDMA number 0 -> 7.

Remarks:

when requesting a QDMA channel from EDMA_getResource(), the returned channel # is not directly usable as a QDMA channel #. This macro translates from the "magic number" range to the actual QDMA channel # (0 -> 7).

See also:

EDMA_getResource()
EDMA_freeResource()
EMDA_QDMA()

7.2.3 Enumeration Type Documentation

7.2.3.1 enum EDMA_commmands

EDMAK ioctl() commands.

Enumerator:

EDMA_IOCREQUESTDMA
EDMA_IOCRELEASEDMA
EDMA_IOCGETVERSION
EDMA_IOCGETBASEPHYSADDR
EDMA_IOCREGUSER

7.2.3.2 enum EDMA_Status

Status codes for EDMA API return values.

Enumerator:

EDMA_OK OK
EDMA_EFAIL general failure
EDMA_ENOCHANNEL no channels available
EDMA_ENOINIT EDMA_init() not called

7.2.4 Function Documentation

7.2.4.1 int EDMA_init (void)

EDMA initialization API.

7.2.4.2 int EDMA_exit (void)

EDMA finalization API.

7.2.4.3 EDMA_Status EDMA_mapBaseAddress (void ** *pvirtAddr*)

Returns user space virtual address of EDMA base registers.

Parameters:

pvirtAddr pointer to storage for virtual address.

7.2.4.4 EDMA_Status EDMA_getResource (int *devId*, int * *tcc*, int * *channel*, int * *param*, int *nParams*)

Returns available logical channels.

Parameters:

devId EDMA channel # or EDMA_QDMA(0 -> # QDMAs) or EDMA_EDMAANY or EDMA_QDMAANY or EDMA_PARAMANY.

tcc pointer to TCC #. Serves as input or output or both, depending on the type of resource requested (devId).

channel pointer to channel #. Serves as output parameter only (explicit channel # request has channel # in devId parameter).

param pointer to PaRAM #. Serves as input or output or both, depending on the type of resource requested (devId).

nParams number of consecutive PaRAMs to allocate, applies only when devId is EDMA_PARAMANY.

Remarks:

Fails if the requested resource(s) is not available.
The following information shows which function parameters are valid for all types of allocations: devId input parameters output parameters —– ————– ————– EDMA # none channel, tcc, param∗# EDMA_EDMAANY tcc channel, tcc, param∗ EDMA_QDMA# tcc channel, tcc, param EDMA_QDMAANY tcc channel, tcc, param EDMA_PARAMANY nParams channel, tcc, param∗ ∗ channel and param are the same value # channel and tcc are the same value

See also:

EDMA_freeResource()
EDMA_registerResource()
EDMA_unregister()
EDMA_QDMA()
EDMA_QDMA2NUM()

7.2.4.5 EDMA_Status EDMA_freeResource (int *lch*, int *nParams*)

Frees previously requested logical channel.

Parameters:

lch channel number. EDMA channels range from 0 -> (# channels). PaRAM channels range from (# channels) -> (# PaRAMs). QDMA channels range from EDMA_QDMA(0 -> # QDMAs).

nParams number of consecutive PaRAMs to free, starting at lch, applies only when lch is a PaRAM identifier.

See also:

EDMA_getResource()

EDMA_registerResource()
EDMA_unregister()
EDMA_QDMA()
EDMA_QDMA2NUM()

7.2.4.6 int EDMA_getVersion (void)

Retrieve version from EDMA driver.

Returns:

Installed EDMA driver's version number.

Precondition:

Must have called EDMA_init()

7.2.4.7 EDMA_Status EDMA_registerResource (int *lch*)

Registers calling process as a user of lch.

Parameters:

lch Resource (channel) # returned by EDMA_getResource().

Remarks:

Fails if the requested resource is not already allocated.
Each call by the same process adds the process to the registration list a separate time, so that each call must be matched by a corresponding EDMA_unregister() call.

See also:

EDMA_unregister()
EDMA_getResource()
EDMA_freeResource()

7.2.4.8 EDMA_Status EDMA_unregister (int *lch*, int *nParams*)

Unregisters calling process as a user of lch.

Parameters:

lch Resource (channel) # returned by EDMA_getResource().

nParams number of PaRAMs, if lch represents a set of PaRAMS.

Remarks:

Fails if the calling process is not already registered for lch.
A process must match each EDMA_registerResource() call with a call to EDMA_unregister().

See also:

EDMA_freeResource()
EDMA_getResource()
EDMA_registerResource()

7.3 SDMA Manager

7.3.1 Detailed Description

This is the API for the SDMA Manager.

Data Structures

- struct SDMA_transferState

  State structure shared between SDMAK and SDMA user layers.

- union SDMA_requestDmaParams

  SDMAK ioctl input/output parameters.

- struct SDMA_ChannelDescriptor

  Descriptor for a channel granted with a SDMA_IOCREQUESTDMA ioctl().

Defines

- #define SDMA_VERSION 0x01000000U

Enumerations

- enum SDMA_commmands {
  SDMA_IOCREQUESTDMA = 1,
  SDMA_IOCRELEASEDMA,
  SDMA_IOCWAITFORCOMPLETION,
  SDMA_IOCGETVERSION }
  SDMAK ioctl() commands.

- enum SDMA_Status {
  SDMA_OK = 0,
  SDMA_EFAIL,
  SDMA_ENOCHANNEL,
  SDMA_ENOINIT }
  Status codes for SDMA API return values.

Functions

- int SDMA_init (void)
  SDMA initialization API.

- int SDMA_exit (void)
  SDMA finalization API.

- SDMA_Status SDMA_getChannels (int numChannels, SDMA_ChannelDescriptor chanArray[])
  Returns available logical channels.

- SDMA_Status SDMA_freeChannels (int numChannels, SDMA_ChannelDescriptor chanArray[])
  Frees previously requested logical channels.

- SDMA_Status SDMA_wait (SDMA_ChannelDescriptor *channel)
  Waits for the transfer corresponding to this descriptor to complete.

- SDMA_Status SDMA_check (SDMA_ChannelDescriptor *channel, int *pcompleted)
  *Checks the status of the transfer corresponding to this descriptor and returns it in the *pcompleted field.*

- int SDMA_getVersion (void)
  Retrieve version from SDMA driver.

7.3.2 Define Documentation

7.3.2.1 #define SDMA_VERSION 0x01000000U

7.3.3 Enumeration Type Documentation

7.3.3.1 enum SDMA_commmands

SDMAK ioctl() commands.

Enumerator:

SDMA_IOCREQUESTDMA
SDMA_IOCRELEASEDMA
SDMA_IOCWAITFORCOMPLETION
SDMA_IOCGETVERSION

7.3.3.2 enum SDMA_Status

Status codes for SDMA API return values.

Enumerator:

SDMA_OK OK
SDMA_EFAIL general failure
SDMA_ENOCHANNEL no channels available
SDMA_ENOINIT SDMA_init() not called

7.3.4 Function Documentation

7.3.4.1 int SDMA_init (void)

SDMA initialization API.

7.3.4.2 int SDMA_exit (void)

SDMA finalization API.

7.3.4.3 SDMA_Status SDMA_getChannels (int *numChannels*, SDMA_ChannelDescriptor *chanArray*[])

Returns available logical channels.

Parameters:

numChannels number of requested channels.

chanArray pointer to SDMA_ChannelDescriptor storage of sufficient size to hold numChannels descriptors. All structure array fields are filled by this API (in other words, no fields need to be filled by the caller prior to the call).

Remarks:

chanArray must be allocated from memory that persists over the life of the channel's usage, since the address of the chanArray will be used by the SDMA kernel driver. This implies that the contents of the passed chanArray should not be copied to another SDMA_ChannelDescriptor that is then used for the subsequent operations.
Fails if the requested number of channels is not available.

See also:

SDMA_freeChannels()

7.3.4.4 SDMA_Status SDMA_freeChannels (int *numChannels*, SDMA_ChannelDescriptor *chanArray*[])

Frees previously requested logical channels.

Parameters:

numChannels number of requested channels.

chanArray pointer to SDMA_ChannelDescriptor storage of sufficient size to hold numChannels descriptors. This will typically be the same pointer passed to SDMA_getChannels().

See also:

SDMA_getChannels()

7.3.4.5 SDMA_Status SDMA_wait (SDMA_ChannelDescriptor * *channel*)

Waits for the transfer corresponding to this descriptor to complete.

Parameters:

channel pointer to the single channel for which to wait.

Remarks:

SDMA_wait() first checks the transferComplete flag of the struct SDMA_transferState in channel. This flag is directly written by the SDMAK module's completion ISR. If this flag is not set to 1, SDMA_wait() will issue a `SDMA_IOCWAITFORCOMPLETION` ioctl(), which will block inside the SDMAK module and become unblocked by the completion ISR.

See also:

SDMA_check()

7.3.4.6 SDMA_Status SDMA_check (SDMA_ChannelDescriptor * *channel*, int * *pcompleted*)

Checks the status of the transfer corresponding to this descriptor and returns it in the *pcompleted field.

Parameters:

channel pointer to single channel to check.

pcompleted pointer to completion status.

Remarks:

copies the contents of the transferComplete flag of the struct SDMA_transferState in channel into *pcompleted (does not issue any ioctl()).

See also:

SDMA_wait()

7.3.4.7 int SDMA_getVersion (void)

Retrieve version from SDMA driver.

Returns:

Installed SDMA driver's version number.

Precondition:

Must have called SDMA_init()

7.4 VICP Manager

7.4.1 Detailed Description

This is the API for the VICP Manager.

Defines

- #define VICP_VERSION 0x01000000

Enumerations

- enum VICP_Status {
  VICP_OK = 0,
  VICP_EFAIL,
  VICP_ENOCHANNEL,
  VICP_ENOINIT }
  Status codes for VICP API return values.
- enum VICP_ResourceType {
  VICP_IMX0,
  VICP_IMX1,
  VICP_MJCP,

VICP_NSF,
VICP_HDVICP0,
VICP_HDVICP1 }

Resource types to be used for all APIs.

- enum VICP_InterruptLine {
  VICP_FIXED,
  VICP_FLEXIBLE }

  Source of VICP resource interrupt.

- enum VICP_InterruptType {
  VICP_IRQ,
  VICP_FIQ }

  Type of ARM interrupt support.

Functions

- int VICP_init (void)

  VICP initialization API.

- int VICP_exit (void)

  VICP finalization API.

- VICP_Status VICP_register (VICP_ResourceType resource, VICP_InterruptLine intrLine, VICP_InterruptType intrType)

  Registers resource with requested interrupt type.

- VICP_Status VICP_unregister (VICP_ResourceType resource)

  Frees previously requested resource.

- VICP_Status VICP_wait (VICP_ResourceType resource)

  Waits for the transfer corresponding to this resource to complete.

- VICP_Status VICP_done (VICP_ResourceType resource, int *done)

  Checks if the transfer corresponding to this resource is complete.

7.4.2 Define Documentation

7.4.2.1 #define VICP_VERSION 0x01000000

7.4.3 Enumeration Type Documentation

7.4.3.1 enum VICP_Status

Status codes for VICP API return values.

Enumerator:

VICP_OK OK
VICP_EFAIL general failure
VICP_ENOCHANNEL no channels available
VICP_ENOINIT VICP_init() not called

7.4.3.2 enum VICP_ResourceType

Resource types to be used for all APIs.

Enumerator:

VICP_IMX0
VICP_IMX1
VICP_MJCP
VICP_NSF
VICP_HDVICP0
VICP_HDVICP1

7.4.3.3 enum VICP_InterruptLine

Source of VICP resource interrupt.

Remarks:

VICP_FIXED tells the IRQ driver to use the interrupt that's dedicated for use with the particular VICP_ResourceType.
VICP_FLEXIBLE tells the IRQ driver to use the "flexible" VICP interrupt, which can be driven by all VICP_ResourceType except VICP_MJCP.

Enumerator:

VICP_FIXED
VICP_FLEXIBLE

7.4.3.4 enum VICP_InterruptType

Type of ARM interrupt support.

Enumerator:

VICP_IRQ
VICP_FIQ

7.4.4 Function Documentation

7.4.4.1 int VICP_init (void)

VICP initialization API.

7.4.4.2 int VICP_exit (void)

VICP finalization API.

7.4.4.3 VICP_Status VICP_register (VICP_ResourceType *resource*, VICP_InterruptLine *intrLine*, VICP_InterruptType *intrType*)

Registers resource with requested interrupt type.

7.4.4.4 VICP_Status VICP_unregister (VICP_ResourceType *resource*)

Frees previously requested resource.

7.4.4.5 VICP_Status VICP_wait (VICP_ResourceType *resource*)

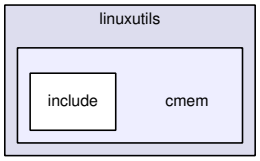
Waits for the transfer corresponding to this resource to complete.

7.4.4.6 VICP_Status VICP_done (VICP_ResourceType *resource*, int * *done*)

Checks if the transfer corresponding to this resource is complete.

8 Linux Utils Application Programming Interface (API) Directory Documentation

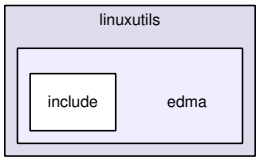
8.1 packages/ti/sdo/linuxutils/cmem/ Directory Reference



Directories

- directory include

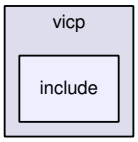
8.2 packages/ti/sdo/linuxutils/edma/ Directory Reference



Directories

- directory include

8.3 packages/ti/sdo/linuxutils/vicp/include/ Directory Reference

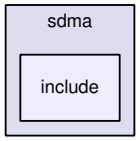


Files

- file vicp.h

  Describes the interface to the VICP manager.

8.4 packages/ti/sdo/linuxutils/sdma/include/ Directory Reference

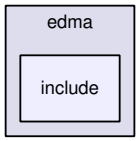


Files

- file sdma.h

  Describes the interface to the SDMA manager.

8.5 packages/ti/sdo/linuxutils/edma/include/ Directory Reference

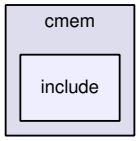


Files

- file edma.h

  Describes the interface to the EDMA manager.

8.6 packages/ti/sdo/linuxutils/cmem/include/ Directory Reference

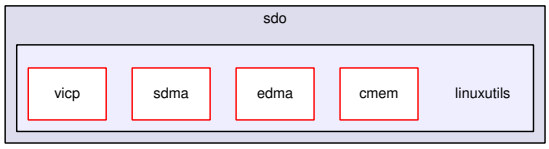


Files

- file cmem.h

  Describes the interface to the contiguous memory allocator.

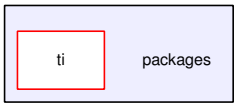
8.7 packages/ti/sdo/linuxutils/ Directory Reference



Directories

- directory cmem
- directory edma
- directory sdma
- directory vicp

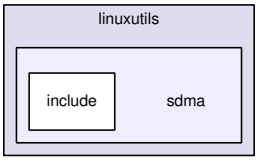
8.8 packages/ Directory Reference



Directories

- directory ti

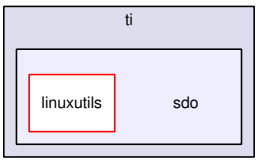
8.9 packages/ti/sdo/linuxutils/sdma/ Directory Reference



Directories

- directory include

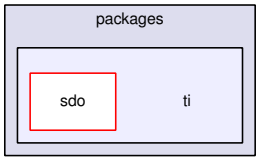
8.10 packages/ti/sdo/ Directory Reference



Directories

- directory linuxutils

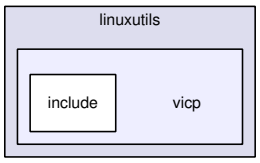
8.11 packages/ti/ Directory Reference



Directories

- directory sdo

8.12 packages/ti/sdo/linuxutils/vicp/ Directory Reference



Directories

- directory include

9 Linux Utils Application Programming Interface (API) Data Structure Documentation

9.1 CMEM_AllocParams Struct Reference

`#include <cmem.h>`

9.1.1 Detailed Description

Parameters for CMEM_alloc(), CMEM_alloc2(), CMEM_allocPool(), CMEM_allocPool2(), CMEM_free().

Data Fields

- int type
- int flags
- size_t alignment

9.1.2 Field Documentation

9.1.2.1 int CMEM_AllocParams::type

either CMEM_HEAP or CMEM_POOL

9.1.2.2 int CMEM_AllocParams::flags

either CMEM_CACHED or CMEM_NONCACHED

9.1.2.3 size_t CMEM_AllocParams::alignment

only used for heap allocations, must be power of 2

The documentation for this struct was generated from the following file:

- cmem.h

9.2 CMEM_BlockAttrs Struct Reference

`#include <cmem.h>`

Data Fields

- unsigned long phys_base
- size_t size

9.2.1 Field Documentation

9.2.1.1 unsigned long CMEM_BlockAttrs::phys_base

9.2.1.2 size_t CMEM_BlockAttrs::size

The documentation for this struct was generated from the following file:

- cmem.h

9.3 EDMA_releaseDmaParams Struct Reference

`#include <edma.h>`

9.3.1 Detailed Description

EDMAK ioctl input parameters.

Data Fields

- int channel
- int nParam

9.3.2 Field Documentation

9.3.2.1 int EDMA_releaseDmaParams::channel

9.3.2.2 int EDMA_releaseDmaParams::nParam

The documentation for this struct was generated from the following file:

- edma.h

9.4 EDMA_requestDmaParams Struct Reference

`#include <edma.h>`

9.4.1 Detailed Description

EDMAK ioctl input/output parameters.

Data Fields

- int dev_id
- int eventq_no
- int tcc
- int param
- int nParam
- int channel

9.4.2 Field Documentation

9.4.2.1 int EDMA_requestDmaParams::dev_id

requested resource EDMA #|EDMA_EDMAANY|EMDA_QDMA(0 -> 7)|EDMA_QDMAANY|EDMA_PARAMANY, input to EDMA_IOCREQUESTDMA ioctl()

9.4.2.2 int EDMA_requestDmaParams::eventq_no

event queue number, input to EDMA_IOCREQUESTDMA ioctl()

9.4.2.3 int EDMA_requestDmaParams::tcc

requested/granted TCC number, input and/or output to and/or from EDMA_IOCREQUESTDMA ioctl()

9.4.2.4 int EDMA_requestDmaParams::param

requested/granted PaRAM number, input and/or output to and/or from EDMA IOCREQUESTDMA ioctl()

9.4.2.5 int EDMA_requestDmaParams::nParam

9.4.2.6 int EDMA_requestDmaParams::channel

granted channel number, output from EDMA_IOCREQUESTDMA ioctl()

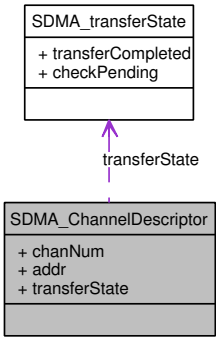
The documentation for this struct was generated from the following file:

- edma.h

9.5 SDMA_ChannelDescriptor Struct Reference

`#include <sdma.h>`

Collaboration diagram for SDMA_ChannelDescriptor:



9.5.1 Detailed Description

Descriptor for a channel granted with a SDMA_IOCREQUESTDMA ioctl().

Remarks:

The address of the transferState member is passed down to the SDMAK module during the `SDMA_IOCREQUESTDMA` ioctl(). The SDMAK module

maps this address and queries and writes to it during the completion ISR activity. It is therefore vital that the address of the descriptor passed to `SDMA_getChannels()` is in persistent memory that is live for the duration of SDMA channel usage.

See also:

SDMA_getChannels()
SDMA_freeChannels()
SDMA_wait()
SDMA_check()

Data Fields

- int chanNum
- unsigned int * addr
- SDMA_transferState transferState

9.5.2 Field Documentation

9.5.2.1 int SDMA_ChannelDescriptor::chanNum

channel number

9.5.2.2 unsigned int* SDMA_ChannelDescriptor::addr

user virtual address of granted DMA registers

9.5.2.3 struct SDMA_transferState SDMA_ChannelDescriptor::transferState

used for completion checking, whose address is passed down to sdmak kernel driver

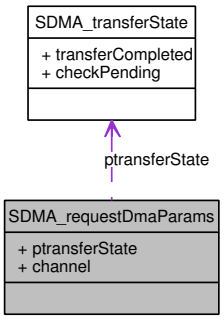
The documentation for this struct was generated from the following file:

- sdma.h

9.6 SDMA_requestDmaParams Union Reference

`#include <sdma.h>`

Collaboration diagram for SDMA_requestDmaParams:



9.6.1 Detailed Description

SDMAK ioctl input/output parameters.

Data Fields

- SDMA_transferState * ptransferState
- int channel

9.6.2 Field Documentation

9.6.2.1 struct SDMA_transferState* SDMA_requestDmaParams::ptransferState

user address of user-owned transfer state structure, input for REQEST_DMA ioctl()

9.6.2.2 int SDMA_requestDmaParams::channel

granted channel number, output from SDMA_IOCREQUESTDMA ioctl()

The documentation for this union was generated from the following file:

- sdma.h

9.7 SDMA_transferState Struct Reference

`#include <sdma.h>`

9.7.1 Detailed Description

State structure shared between SDMAK and SDMA user layers.

Remarks:

The address of a struct SDMA_transferState instance gets passed down to the SDMAK module during the `SDMA_IOCREQUESTDMA` ioctl() operation and is registered with the received channel. It is therefore vital that the structure address that is registered with the kernel module points to the actual location that is queried and written during a "wait" operation.

Data Fields

- volatile int transferCompleted
- volatile int checkPending

9.7.2 Field Documentation

9.7.2.1 volatile int SDMA_transferState::transferCompleted

flag for channel completion

9.7.2.2 volatile int SDMA_transferState::checkPending

completion check hand-shaking flag

The documentation for this struct was generated from the following file:

- sdma.h

10 Linux Utils Application Programming Interface (API) File Documentation

10.1 cmem.h File Reference

10.1.1 Detailed Description

Describes the interface to the contiguous memory allocator.

The cmem user interface library wraps file system calls to an associated kernel module (cmemk.ko), which needs to be loaded in order for calls to this library to succeed.

The following is an example of installing the cmem kernel module:

```
/sbin/insmod cmemk.ko pools=4x30000,2x500000 phys_start=0x0 phys_end=0x3000000
```

- phys_start and phys_end must be specified in hexadecimal format
- phys_start is "inclusive" and phys_end is "exclusive", i.e., phys_end should be "end address + 1".
- pools must be specified using decimal format (for both number and size), since using hexadecimal format would visually clutter the specification due to the use of "x" as a token separator

This particular command creates 2 pools. The first pool is created with 4 buffers of size 30000 bytes and the second pool is created with 2 buffers of size 500000 bytes. The CMEM pool buffers start at 0x0 and end at 0x2FFFFFF (max).

There is also support for a 2nd contiguous memory block to be specified, with all the same features supported for the 2nd block as with the 1st. This 2nd block is specified with *_1 parameters. The following example expands upon the first example above:

```
/sbin/insmod cmemk.ko pools=4x30000,2x500000 phys_start=0x0 phys_end=0x3000000
    pools_1=4x65536 phys_start_1=0x80000000 phys_end_1=0x80010000
```

This particular command, in addition to the pools explained above, creates 1 pool (with 4 buffers of size 64KB) in a 2nd memory block which starts at 0x80000000 and ends at 0x8000FFFF (specified as "end + 1" on the insmod command).

In order to access this 2nd memory block, new APIs have been added to CMEM which allow specification of the block ID.

Pool buffers are aligned on a module-dependent boundary, and their sizes are rounded up to this same boundary. This applies to each buffer within a pool. The total space used by an individual pool will therefore be greater than (or equal to) the exact amount requested in the installation of the module.

The poolid used in the driver calls would be 0 for the first pool and 1 for the second pool.

Pool allocations can be requested explicitly by pool number, or more generally by just a size. For size-based allocations, the pool which best fits the requested size is automatically chosen. Some CMEM APIs (newer ones) accept a blockid as a parameter, in order to specify which of the multiple blocks to operate on. For 'legacy' APIs (ones that existed before the support for multiple blocks) where a blockid is still needed, block 0 is assumed.

There is also support for a general purpose heap. In addition to the 2 pools described above, a general purpose heap block is created from which allocations of any size can be requested. Internally, allocation sizes are rounded up to a module-dependent boundary and allocation addresses are aligned either to this same boundary or to the requested alignment (whichever is greater).

The size of the heap block is the amount of CMEM memory remaining after all pool allocations. If more heap space is needed than is available after pool allocations, you must reduce the amount of CMEM memory granted to the pools.

Buffer allocation is tracked at the process level by way of a 'process registration' list. The initial allocator of a buffer (the process that calls CMEM_alloc()) is automatically added to the registration list, and further processes can become registered for the same buffer by way of the CMEM_registerAlloc() API (and unregister with the CMEM_unregister() API). This registration list for each buffer allows for buffer ownership tracking and cleanup on a per-process basis, so that when a process exits or dies without having explicitly freed/unregistered its buffers, they get automatically unregistered (and freed when no more registered processes exist). Only when the last registered process frees a buffer (either explictily, or by auto-cleanup) does a buffer actually get freed back to the kernel module.

Since the CMEM interface library doesn't use the GT tracing facility, there is one configuration option available for the CMEM module to control whether the debug or release interface library is used for building the application. This config parameter is named 'debug' and is of type bool, and the default value is 'false'.

The following line is an example of enabling usage of the debug interface library: var cmem = xdc.useModule('ti.sdo.linuxutils.cmem.CMEM'); cmem.debug = true; This will enable "CMEM Debug" statements to be printed to stdout.

Data Structures

- struct CMEM_AllocParams

  Parameters for CMEM_alloc(), CMEM_alloc2(), CMEM_allocPool(), CMEM_allocPool2(), CMEM_free().

- struct CMEM_BlockAttrs

Defines

- #define CMEM_VERSION 0x02300000U
- #define CMEM_WB 0x0100
- #define CMEM_INV 0x0200
- #define CMEM_HEAP 0x0400
- #define CMEM_POOL 0x0000
- #define CMEM_CACHED 0x0800
- #define CMEM_NONCACHED 0x0000
- #define CMEM_PHYS 0x1000
- #define CMEM_IOCALLOC 1
- #define CMEM_IOCALLOCHEAP 2
- #define CMEM_IOCFREE 3
- #define CMEM_IOCGETPHYS 4
- #define CMEM_IOCGETSIZE 5
- #define CMEM_IOCGETPOOL 6
- #define CMEM_IOCCACHE 7
- #define CMEM_IOCGETVERSION 8
- #define CMEM_IOCGETBLOCK 9
- #define CMEM_IOCREGUSER 10
- #define CMEM_IOCCACHEWBINV CMEM_IOCCACHE | CMEM_WB | CMEM_INV
- #define CMEM_IOCCACHEWB CMEM_IOCCACHE | CMEM_WB
- #define CMEM_IOCCACHEINV CMEM_IOCCACHE | CMEM_INV
- #define CMEM_IOCALLOCCACHED CMEM_IOCALLOC | CMEM_CACHED
- #define CMEM_IOCALLOCHEAPCACHED CMEM_IOCALLOCHEAP | CMEM_CACHED
- #define CMEM_IOCFREEHEAP CMEM_IOCFREE | CMEM_HEAP
- #define CMEM_IOCFREEPHYS CMEM_IOCFREE | CMEM_PHYS
- #define CMEM_IOCFREEHEAPPHYS CMEM_IOCFREE | CMEM_HEAP | CMEM_PHYS
- #define CMEM_IOCCMDMASK 0x000000ff

Functions

- int CMEM_init (void)

  Initialize the CMEM module. Must be called before other API calls.

- int CMEM_getPool (size_t size)

  Find the pool that best fits a given buffer size and has a buffer available.

- int CMEM_getPool2 (int blockid, size_t size)

Find the pool in memory block blockid that best fits a given buffer size and has a buffer available.

- void * CMEM_allocPool (int poolid, CMEM_AllocParams *params)
  Allocate memory from a specified pool.
- void * CMEM_allocPool2 (int blockid, int poolid, CMEM_AllocParams *params)
  Allocate memory from a specified pool in a specified memory block.
- void * CMEM_alloc (size_t size, CMEM_AllocParams *params)
  Allocate memory of a specified size.
- void * CMEM_alloc2 (int blockid, size_t size, CMEM_AllocParams *params)
  Allocate memory of a specified size from a specified memory block.
- void * CMEM_registerAlloc (unsigned long physp)
  Register shared usage of an already-allocated buffer.
- int CMEM_free (void *ptr, CMEM_AllocParams *params)
  Free a buffer previously allocated with CMEM_alloc()/CMEM_allocPool().
- int CMEM_unregister (void *ptr, CMEM_AllocParams *params)
  Unregister use of a buffer previously registered with CMEM_registerAlloc().
- unsigned long CMEM_getPhys (void *ptr)
  Get the physical address of a contiguous buffer.
- int CMEM_cacheWb (void *ptr, size_t size)
  Do a cache writeback of the block pointed to by `ptr/size`.
- int CMEM_cacheInv (void *ptr, size_t size)
  Do a cache invalidate of the block pointed to by `ptr/size`.
- int CMEM_cacheWbInv (void *ptr, size_t size)
  Do a cache writeback/invalidate of the block pointed to by `ptr/size`.
- int CMEM_getVersion (void)
  Retrieve version from CMEM driver.
- int CMEM_getBlock (unsigned long *pphys_base, size_t *psize)
  Retrieve memory block bounds from CMEM driver.

- int CMEM_getBlockAttrs (int blockid, CMEM_BlockAttrs *pattrs)
  Retrieve extended memory block attributes from CMEM driver.
- int CMEM_exit (void)
  Finalize the CMEM module.

Variables

- CMEM_AllocParams CMEM_DEFAULTPARAMS

10.2 disclaimer.dox File Reference

10.3 doxygen.txt File Reference

10.4 edma.h File Reference

10.4.1 Detailed Description

Describes the interface to the EDMA manager.

The edma user interface library wraps file system calls to an associated kernel module (edmak.ko), which needs to be loaded in order for calls to to this library to succeed.

To install edmak.ko, enter % insmod edmak.ko on a Linux command line.

EDMA channels are allocated (requested) from within the edmak kernel driver through the EDMA_getResource() API. This request is granted by way of the kernel's request_dma() functionality (davinci_request_dma() for the DM365), so it will work in conjuntion with independent kernel DMA usage without conflict or custom kernel configuration.

EDMA registers can be mapped to a process's address by way of the API EDMA_mapBaseAddress(). The EDMA registers are memory-mapped into the calling process' user address space to allow the application to directly access the EDMA registers.

Resource allocation is tracked at the process level by way of a 'process registration' list. The initial allocator of a resource (the process that calls EDMA_getResource()) is automatically added to the registration list, and further processes can become registered for the same resource by way of the EDMA_registerResource() API (and unregister with the EDMA_unregister() API). This registration list for each resource (logical channel) allows for resource ownership tracking and cleanup on a per-process basis, so that when a process exits or dies without having explicitly freed/unregistered its resources, they get automatically unregistered (and freed when no more registered processes exist). Only when the last registered process frees a resource (either explictily, or by auto-cleanup) does a resource actually get freed back to the kernel.

Since the EDMA interface library doesn't use the GT tracing facility, there is one configuration option available for the EDMA module to control whether the debug or release interface library is used for building the application. This config parameter is named 'debug' and is of type bool, and the default value is 'false'.

The following line is an example of enabling usage of the debug interface library: var edma = xdc.useModule('ti.sdo.linuxutils.edma.EDMA'); edma.debug = true; This will enable "EDMA Debug" statements to be printed to stdout.

Data Structures

- struct EDMA_requestDmaParams

  EDMAK ioctl input/output parameters.

- struct EDMA_releaseDmaParams

  EDMAK ioctl input parameters.

Defines

- #define EDMA_VERSION 0x01100000U
- #define EDMA_TCCANY 1001

  Values for dev_id parameter of EDMA_getResource().

- #define EDMA_TCCSYMM 1005
- #define EDMA_PARAMANY 1006
- #define EDMA_PARAMFIXEDEXACT 1007
- #define EDMA_PARAMFIXEDNOTEXACT 1008
- #define EDMA_EDMAANY 1003
- #define EDMA_QDMAANY 1004
- #define EDMA_QDMA0 512
- #define EDMA_QDMA1 EDMA_QDMA(1)

- #define EDMA_QDMA2 EDMA_QDMA(2)
- #define EDMA_QDMA3 EDMA_QDMA(3)
- #define EDMA_QDMA4 EDMA_QDMA(4)
- #define EDMA_QDMA5 EDMA_QDMA(5)
- #define EDMA_QDMA6 EDMA_QDMA(6)
- #define EDMA_QDMA7 EDMA_QDMA(7)
- #define EDMA_QDMA(n) (EDMA_QDMA0 + (n))

  macro used to translate from a QDMA channel # to the numerical range used by EDMA_getResource() & EDMA_freeResource() for representing a QDMA channel.

- #define EDMA_QDMA2NUM(qdma) (qdma - EDMA_QDMA0)

  macro to translate from EDMA_QDMA0 -> EDMA_QDMA7 namespace to actual QDMA number 0 -> 7.

Enumerations

- enum EDMA_commmands {
  EDMA_IOCREQUESTDMA = 1,
  EDMA_IOCRELEASEDMA,
  EDMA_IOCGETVERSION,
  EDMA_IOCGETBASEPHYSADDR,
  EDMA_IOCREGUSER }

  EDMAK ioctl() commands.

- enum EDMA_Status {
  EDMA_OK = 0,
  EDMA_EFAIL,
  EDMA_ENOCHANNEL,
  EDMA_ENOINIT }

  Status codes for EDMA API return values.

Functions

- int EDMA_init (void)

  EDMA initialization API.

- int EDMA_exit (void)

EDMA finalization API.

- EDMA_Status EDMA_mapBaseAddress (void **pvirtAddr)
  Returns user space virtual address of EDMA base registers.
- EDMA_Status EDMA_getResource (int devId, int *tcc, int *channel, int *param, int nParams)
  Returns available logical channels.
- EDMA_Status EDMA_freeResource (int lch, int nParams)
  Frees previously requested logical channel.
- int EDMA_getVersion (void)
  Retrieve version from EDMA driver.
- EDMA_Status EDMA_registerResource (int lch)
  Registers calling process as a user of lch.
- EDMA_Status EDMA_unregister (int lch, int nParams)
  Unregisters calling process as a user of lch.

10.5 sdma.h File Reference

10.5.1 Detailed Description

Describes the interface to the SDMA manager.

The sdma user interface library wraps file system calls to an associated kernel module (sdmak.ko), which needs to be loaded in order for calls to to this library to succeed.

To install sdmak.ko, enter % insmod sdmak.ko on a Linux command line.

SDMA channels are allocated (requested) from within the sdmak kernel driver through the SDMA_getChannels() API. This request is granted by way of the kernel's request_dma() functionality (omap_request_dma() for the OMAP35x), so it will work in conjuntion with independent kernel DMA usage without conflict or custom kernel configuration. The granted channel's registers are memory-mapped into the calling user address space to allow the application to directly access the DMA registers for that channel.

A DMA callback ISR is registered within the stock kernel DMA manager, and this ISR is used for completion notification (although it could be customized to

handle any DMA-generated interrupt). Notification to the user interface layer is accomplished by way of a user state structure whose address is passed to the sdmak kernel driver and which is directly written upon completion. Atomic hand-shaking allows the kernel driver to know if the application is going to wait for the completion by way of the corresponding sdmak ioctl(), as opposed to just checking the user state structure, and post a channel-specific mutex only when needed (otherwise just the completion flag in the state structure is set).

Since the SDMA interface library doesn't use the GT tracing facility, there is one configuration option available for the SDMA module to control whether the debug or release interface library is used for building the application. This config parameter is named 'debug' and is of type bool, and the default value is 'false'.

The following line is an example of enabling usage of the debug interface library: var sdma = xdc.useModule('ti.sdo.linuxutils.sdma.SDMA'); sdma.debug = true; This will enable "SDMA Debug" statements to be printed to stdout.

Data Structures

- struct SDMA_transferState

  State structure shared between SDMAK and SDMA user layers.

- union SDMA_requestDmaParams

  SDMAK ioctl input/output parameters.

- struct SDMA_ChannelDescriptor

  Descriptor for a channel granted with a SDMA_IOCREQUESTDMA ioctl().

Defines

- #define SDMA_VERSION 0x01000000U

Enumerations

- enum SDMA_commmands {
  SDMA_IOCREQUESTDMA = 1,
  SDMA_IOCRELEASEDMA,
  SDMA_IOCWAITFORCOMPLETION,
  SDMA_IOCGETVERSION }

  SDMAK ioctl() commands.

- enum SDMA_Status {
  SDMA_OK = 0,
  SDMA_EFAIL,
  SDMA_ENOCHANNEL,
  SDMA_ENOINIT }
  Status codes for SDMA API return values.

Functions

- int SDMA_init (void)
  SDMA initialization API.
- int SDMA_exit (void)
  SDMA finalization API.
- SDMA_Status SDMA_getChannels (int numChannels, SDMA_ChannelDescriptor chanArray[])
  Returns available logical channels.
- SDMA_Status SDMA_freeChannels (int numChannels, SDMA_ChannelDescriptor chanArray[])
  Frees previously requested logical channels.
- SDMA_Status SDMA_wait (SDMA_ChannelDescriptor *channel)
  Waits for the transfer corresponding to this descriptor to complete.
- SDMA_Status SDMA_check (SDMA_ChannelDescriptor *channel, int *pcompleted)
  *Checks the status of the transfer corresponding to this descriptor and returns it in the *pcompleted field.*
- int SDMA_getVersion (void)
  Retrieve version from SDMA driver.

10.6 vicp.h File Reference

10.6.1 Detailed Description

Describes the interface to the VICP manager.

The VICP user interface library wraps file system calls to an associated kernel module (irqk.ko), which needs to be loaded in order for calls to to this library to succeed.

To install irqk.ko, enter % insmod irqk.ko on a Linux command line.

Defines

- #define VICP_VERSION 0x01000000

Enumerations

- enum VICP_Status {
  VICP_OK = 0,
  VICP_EFAIL,
  VICP_ENOCHANNEL,
  VICP_ENOINIT }
  Status codes for VICP API return values.

- enum VICP_ResourceType {
  VICP_IMX0,
  VICP_IMX1,
  VICP_MJCP,
  VICP_NSF,
  VICP_HDVICP0,
  VICP_HDVICP1 }
  Resource types to be used for all APIs.

- enum VICP_InterruptLine {
  VICP_FIXED,
  VICP_FLEXIBLE }
  Source of VICP resource interrupt.

- enum VICP_InterruptType {
  VICP_IRQ,
  VICP_FIQ }
  Type of ARM interrupt support.

Functions

- int VICP_init (void)
  VICP initialization API.
- int VICP_exit (void)
  VICP finalization API.
- VICP_Status VICP_register (VICP_ResourceType resource, VICP_InterruptLine intrLine, VICP_InterruptType intrType)
  Registers resource with requested interrupt type.
- VICP_Status VICP_unregister (VICP_ResourceType resource)
  Frees previously requested resource.
- VICP_Status VICP_wait (VICP_ResourceType resource)
  Waits for the transfer corresponding to this resource to complete.
- VICP_Status VICP_done (VICP_ResourceType resource, int *done)
  Checks if the transfer corresponding to this resource is complete.

11 Linux Utils Application Programming Interface (API) Page Documentation

Index