Intel(R) MIC MYO API Reference Manual 0.3

Generated by Doxygen 1.6.1

Fri Feb 15 10:35:57 2013

Contents

1 Intel(R) MIC MYO API Reference Manual 0.3

Disclaimer and Legal Information

Document Number:

World Wide Web: `http://developer.intel.com`

Intel Confidential

2 Disclaimer and Legal Information

3 MYO Overview

MYO is a shared memory programming model for heterogeneous x86 processors.

MYO eliminates the need for explicit data transfers in distributed applications. In this document we discuss programming model and API's specific to share memory between CPU and CARD device(s). We use the term node while referring to either CPU or CARD to emphasize that the programming model is symmetric. Shared memory refer to having same virtual address for a variable or pointer on all nodes which is also referred to as Shared Virtual Memory (SVM). MYO stands for Mine, Yours and Ours which means shared virtual memory can be either owned by a given node (Mine), ownership released to the other nodes (Yours) or share ownership between all the nodes (Our). Our software based SVM will be referred to as MYO in the rest of this document. Acquiring and releasing ownership is done to reduce redundant data transfer between the nodes. If a particular node is not interested in the shared data then the memory consistency need not be maintained on that particular node.

Any shared memory allocated by the user is assigned from the MYO managed SVM space. Data or pointer located in the SVM is valid on all nodes and any update to the shared memory is made globally visible by the MYO consistency protocol. This allows sharing of complex data structures across compute domains without the need for any data marshalling. For example, a complex tree structure on the host is valid and accessible on CARD without any explicit conversions or data transfers. In the traditional offload model, complex structures have to be converted into simple data blocks for transfer and reconstruct the structure at the destination to begin computation in the new compute domain.

MYO supports Parallel co-processing and Scalar (Reverse) co-processing in a single application. In Parallel co-processing model, remote procedures calls (RPC) are called from host to card and in the Scalar co-processing model, RPC is called from card to host. This kind of mixed model gives flexibility to programmers to optimize applications by running functions which are inherently sequential on the host and parallel routines on card for optimal application performance.

4 Module Documentation

4.1 MYO

Modules

- MYO_API

  Description: External APIs of MYO runtime (MYO stands for Mine, Yours and Ours).

- MYOTYPES

  Description: Define the types used by APIs of MYO programming.

- MYOIMPL_API

  Description: Define APIs of MYO for compiler or pre-processor to transfer original programs.

4.2 MYO_API

Description: External APIs of MYO runtime (MYO stands for Mine, Yours and Ours).

Files

- file myo.h

Functions

- MYOACCESSAPI MyoError myoAcquire ()

  myoAcquire are the sync points for the default arena with "Release Consistency".

- MYOACCESSAPI MyoError myoAcquireOwnership ()

  Changes the ownership type of the default arena to MYO_ARENA_MINE.

- MYOACCESSAPI MyoError myoArenaAcquire (MyoArena in_Arena)

  myoArenaAcquire are the sync points for "OURS" arena with "Release Consistency".

- MYOACCESSAPI MyoError myoArenaAcquireOwnership (MyoArena in_Arena)

  Changes the ownership type of the arena to MYO_ARENA_MINE.

- MYOACCESSAPI void myoArenaAlignedFree (MyoArena in_Arena, void ∗in_pPtr)

  Frees the memory space got by myoArenaAlignedMalloc to the specified arena.

- MYOACCESSAPI void ∗ myoArenaAlignedMalloc (MyoArena in_Arena, size_t in_Size, size_t in_Alignment)

  Allocates size bytes from the specified arena.

- MYOACCESSAPI MyoError myoArenaCreate (MyoOwnershipType in_Type, int in_Property, MyoArena ∗out_pArena)

  Create an arena with specified ownership type and property.

- MYOACCESSAPI MyoError myoArenaDestroy (MyoArena in_Arena)

  Destroy an arena.

- MYOACCESSAPI void myoArenaFree (MyoArena in_Arena, void ∗in_pPtr)

  Frees the memory space got by myoArenaMalloc to the specified arena.

- MYOACCESSAPI MyoError myoArenaGetHandle (void ∗in_pPtr, MyoArena ∗out_pArena)

  Gets the arena handle of the arena which contains the memory space "in_pPtr" points to.

- MYOACCESSAPI void ∗ myoArenaMalloc (MyoArena in_Arena, size_t in_Size)

  Allocates size bytes from the specified arena.

- MYOACCESSAPI MyoError myoArenaRelease (MyoArena in_Arena)

  myoArenaRelease are the sync points for "OURS" arena with "Release Consistency".

- MYOACCESSAPI MyoError myoArenaReleaseOwnership (MyoArena in_Arena)

  Change the ownership type of the arena to MYO_ARENA_OURS.

- MYOACCESSAPI MyoError myoBarrierCreate (int in_Count, MyoBarrier ∗out_pBarrier)

  Create a barrier and return the barrier handle.

- MYOACCESSAPI MyoError myoBarrierDestroy (MyoBarrier in_Barrier)

  Destroy the barrier.

- MYOACCESSAPI MyoError myoBarrierWait (MyoBarrier in_Barrier)

  The caller will block until the required number of threads have called myoBarrierWait with the same barrier handle.

- MYOACCESSAPI MyoError myoMutexCreate (MyoMutex ∗out_pMutex)

  Create a mutex and return the mutex handle.

- MYOACCESSAPI MyoError myoMutexDestroy (MyoMutex in_Mutex)

  Destroy the mutex.

- MYOACCESSAPI MyoError myoMutexLock (MyoMutex in_Mutex)

  Lock the mutex.

- MYOACCESSAPI MyoError myoMutexTryLock (MyoMutex in_Mutex)

  Try to lock the mutex.

- MYOACCESSAPI MyoError myoMutexUnlock (MyoMutex in_Mutex)

  Release the locked mutex.

- MYOACCESSAPI MyoError myoRelease ()

  myoRelease are the sync points for the default arena with ∗ "Release Consistency".

- MYOACCESSAPI MyoError myoReleaseOwnership ()

  Change the ownership type of the default arena to the MYO_ARENA_OURS.

- MYOACCESSAPI MyoError myoSemCreate (int in_Value, MyoSem ∗out_pSem)

  Create a semaphore and return the semaphore handle.

- MYOACCESSAPI MyoError myoSemDestroy (MyoSem in_Sem)

  Destroy the semaphore.

- MYOACCESSAPI MyoError myoSemPost (MyoSem in_Sem)

  Increments (unlocks) the semaphore.

- MYOACCESSAPI MyoError myoSemTryWait (MyoSem in_Sem)

  Try to lock semaphore.

- MYOACCESSAPI MyoError myoSemWait (MyoSem in_Sem)

  Decrements (locks) the semaphore.

- MYOACCESSAPI void myoSharedAlignedFree (void ∗in_pPtr)

  Frees the memory space got by myoArenaAlignedMalloc to the default arena.

- MYOACCESSAPI void ∗ myoSharedAlignedMalloc (size_t in_Size, size_t in_Alignment)

  Allocates size bytes from the default arena.

- MYOACCESSAPI void myoSharedFree (void ∗in_pPtr)

  Frees the memory space got by myoArenaMalloc to the default arena.

- MYOACCESSAPI void ∗ myoSharedMalloc (size_t in_Size)

  There will be a default arena inside MYO runtime, which will be used when there is no specified arena.

4.2.1 Detailed Description

Description: External APIs of MYO runtime (MYO stands for Mine, Yours and Ours).

4.2.2 Function Documentation

4.2.2.1 MyoError myoAcquire ()

myoAcquire are the sync points for the default arena with "Release Consistency". myoAcquire is used to make sure that I will see all stores have been made globally visible prior to this.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.2.2.2 MyoError myoAcquireOwnership ()

Changes the ownership type of the default arena to MYO_ARENA_MINE.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.2.2.3 MyoError myoArenaAcquire (MyoArena *in_Arena*)

myoArenaAcquire are the sync points for "OURS" arena with "Release Consistency". myoArenaAcquire is used to guarantee all prior stores of this arena will be globally visible at this point. myoArenaAcquire is used to make sure that I will see all stores of this arena that have been made globally visible prior to this.

Parameters:

in_Arena Arena handle returned by previous call to myoArenaCreate.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.2.2.4 MyoError myoArenaAcquireOwnership (MyoArena *in_Arena*)

Changes the ownership type of the arena to MYO_ARENA_MINE.

Parameters:

in_Arena Arena handle returned by previous call to myoArenaCreate.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.2.2.5 void myoArenaAlignedFree (MyoArena *in_Arena*, void ∗ *in_pPtr*)

Frees the memory space got by myoArenaAlignedMalloc to the specified arena.

Parameters:

in_Arena Arena handle returned by previous call to myoArenaCreate.

in_pPtr The start address of the specified memory space, which must be retured by myoArenaAlignedMalloc.

Returns:

4.2.2.6 void ∗ myoArenaAlignedMalloc (MyoArena *in_Arena*, size_t *in_Size*, size_t *in_Alignment*)

Allocates size bytes from the specified arena. The start address of the allocated memory will be a multiple of alignment, which must be a power of two.

Parameters:

in_Arena Arena handle returned by previous call to myoArenaCreate.

in_Size Size (bytes) of the required memory space.

in_Alignment The alignment value, which must be an power of two.

Returns:

The start address of the allocated memory space. NULL: Failed.

4.2.2.7 MyoError myoArenaCreate (MyoOwnershipType *in_Type*, int *in_Property*, MyoArena ∗ *out_pArena*)

Create an arena with specified ownership type and property.

Parameters:

in_Type Specified ownership type (MYO_ARENA_OURS or MYO_ARENA_MINE).

in_Property Specified properties of the arena. Just keep it as 0 to use default properties.

MYO_RELEASE_CONSISTENCY or MYO_STRONG_RELEASE_CONSISTENCY or MYO_-STRONG_CONSISTENCY:

Consistency modes for Ours arenas. For MYO_RElEASE_CONSISTENCY, there are 2 functions, "acquire" and "release", are used to for memory ordering. "release" makes all local stores prior to the release globally visible; "acquire" syncs up the local memory with all stores that have been made globally visible. However, there is no definite answer to whether local stores must can be global visible before reaching a release point and whether newest global visible stores can be updated to local before reaching an acquire point. By using MYO_STRONG_RELEASE_CONSISTENCY, the definite answer to the two questions is "no". Sequential consistency model is maintained to the arena when using MYO_STRONG_-CONSISTENCY. MYO_RELEASE_CONSISTENCY is the default one.

MYO_UPDATE_ON_DEMAND or MYO_UPDATE_ON_ACQUIRE:

Only apply to "Release Consistency" Ours arenas. MYO_UPDATE_ON_ACQUIRE means that the shared pages of this arena will be updated on acquire point; MYO_UPDATE_ON_DEMAND means that the shared pages will not be updated until they are accessed. MYO_UPDATE_ON_DEMAND is the default one.

MYO_RECORD_DIRTY or MYO_NOT_RECORD_DIRTY:

Record dirty pages or not. There will be runtime overhead when recording dirty pages, while it can reduce the communication data. It is a trade-off for performance. Also when MYO_NOT_RECORD_DIRTY is set for Our arena, the runtime cannot guarantee the correctness when the host and card modify the same shared page between the same sync segment. MYO_RECORD_DIRTY is the default one.

MYO_ONE_VERSIONS or MYO_MULTI_VERSION:

Only apply to "Release Consistency" Ours arenas. When MYO_MULTI_VERSION is set, this arena can only be release on HOST side and acquire on CARD side. Releasing the arena on HOST will create a new versioned data and put it into a FIFO and acquiring the arena on CARD will get the versioned data from the FIFO one by one. MYO_ONE_VERSION is the default one.

MYO_CONSISTENCY or MYO_NO_CONSISTENCY:

Only apply to "Release Consistency" Ours arenas. When MYO_NO_CONSISTENCY is set, the consistency of the arena will not be maintained. That is to say that it is a no-op operation when calling acquire/release for such arenas. The default property is MYO_CONSISTENCY.

MYO_HOST_TO_DEVICE and MYO_DEVICE_TO_HOST:

When the programmers can make sure that there is only one communication direction for this arena, they can create this arena with only MYO_HOST_TO_DEVICE or MYO_DEVICE_TO_HOST so that the runtime can do some optimizations. The default property is MYO_HOST_TO_DEVICE | MYO_DEVICE_-TO_HOST.

Parameters:

out_pArena Used to store the handle of the created arena.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.2.2.8 MyoError myoArenaDestroy (MyoArena *in_Arena*)

Destroy an arena. As a result, the arena can not be refered any more.

Parameters:

in_Arena Arena handle returned by previous call to myoArenaCreate.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.2.2.9 void myoArenaFree (MyoArena *in_Arena*, void ∗ *in_pPtr*)

Frees the memory space got by myoArenaMalloc to the specified arena.

Parameters:

in_Arena Arena handle returned by previous call to myoArenaCreate.

in_pPtr The start address of the specified memory space, which must be retured by myoArenaMalloc.

Returns:

4.2.2.10 MyoError myoArenaGetHandle (void ∗ *in_pPtr*, MyoArena ∗ *out_pArena*)

Gets the arena handle of the arena which contains the memory space "in_pPtr" points to. This API can be used when you are not sure about which arena handle should be used for other arena related APIs.

Parameters:

in_pPtr The start address of a chunk of memory space.

out_pArena Handle of the arena.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.2.2.11 void * myoArenaMalloc (MyoArena *in_Arena*, size_t *in_Size*)

Allocates size bytes from the specified arena. and returns the start address of the allocated memory. The memory is not cleared.

Parameters:

in_Arena Arena handle returned by previous call to myoArenaCreate.

in_Size Size (bytes) of the required memory space.

Returns:

The start address of the allocated memory space. NULL: Failed.

4.2.2.12 MyoError myoArenaRelease (MyoArena *in_Arena*)

myoArenaRelease are the sync points for "OURS" arena with "Release Consistency". myoArenaRelease is used to guarantee all prior stores of this arena will be globally visible at this point.

Parameters:

in_Arena Arena handle returned by previous call to myoArenaCreate.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.2.2.13 MyoError myoArenaReleaseOwnership (MyoArena *in_Arena*)

Change the ownership type of the arena to MYO_ARENA_OURS.

Parameters:

in_Arena Arena handle returned by previous call to myoArenaCreate.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.2.2.14 MyoError myoBarrierCreate (int *in_Count*, MyoBarrier * *out_pBarrier*)

Create a barrier and return the barrier handle.

Parameters:

in_Count the number of threads that must call myoBarrierWait before any of them successfully return.

out_pBarrier Used to store the handle of the created barrier.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.2.2.15 MyoError myoBarrierDestroy (MyoBarrier *in_Barrier*)

Destroy the barrier.

Parameters:

in_Barrier the barrier handle returned by myoBarrierCreate.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.2.2.16 MyoError myoBarrierWait (MyoBarrier *in_Barrier*)

The caller will block until the required number of threads have called myoBarrierWait with the same barrier handle.

Parameters:

in_Barrier the barrier handle returned by myoBarrierCreate.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.2.2.17 MyoError myoMutexCreate (MyoMutex ∗ *out_pMutex*)

Create a mutex and return the mutex handle.

Parameters:

out_pMutex Used to store the handle of the created mutex.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.2.2.18 MyoError myoMutexDestroy (MyoMutex *in_Mutex*)

Destroy the mutex.

Parameters:

in_Mutex the mutex handle returned by myoMutexCreate.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.2.2.19 MyoError myoMutexLock (MyoMutex *in_Mutex*)

Lock the mutex. If the mutex is already locked by other peers, the calling shall block until the mutex becomes available. Currently, attempting to re-acquire the mutex caused deadlock.

Parameters:

in_Mutex the mutex handle returned by myoMutexCreate.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.2.2.20 MyoError myoMutexTryLock (MyoMutex *in_Mutex*)

Try to lock the mutex. myoMutexTryLock shall be equivalent to myoMutexLock, except that this function shall return immediately if the mutex is currently locked.

Parameters:

in_Mutex the mutex handle returned by myoMutexCreate.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.2.2.21 MyoError myoMutexUnlock (MyoMutex *in_Mutex*)

Release the locked mutex. Currently, attempting to release a unlocked mutex will cause undefined results.

Parameters:

in_Mutex the mutex handle returned by myoMutexCreate.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.2.2.22 MyoError myoRelease ()

myoRelease are the sync points for the default arena with $*$ "Release Consistency". myoRelease is used to guarantee all prior stores will be globally visible at this point.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.2.2.23 MyoError myoReleaseOwnership ()

Change the ownership type of the default arena to the MYO_ARENA_OURS.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.2.2.24 MyoError myoSemCreate (int *in_Value*, MyoSem $*$ *out_pSem*)

Create a semaphore and return the semaphore handle.

Parameters:

in_Value the initial value for the semaphore.

out_pSem Used to store the handle of the created semaphore.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.2.2.25 MyoError myoSemDestroy (MyoSem *in_Sem*)

Destroy the semaphore.

Parameters:

in_Sem the semaphore handle returned by myoSemCreate.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.2.2.26 MyoError myoSemPost (MyoSem *in_Sem*)

Increments (unlocks) the semaphore. If the semaphore value becomes greater than zero, one blocked myoSemWait called will be notified to return.

Parameters:

in_Sem the semaphore handle returned by myoSemCreate.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.2.2.27 MyoError myoSemTryWait (MyoSem *in_Sem*)

Try to lock semaphore. myoSemTryWait is the same as myoSemAcquire, except that if the decrement cannot be immediately performed, then the call returns instead of blocking.

Parameters:

in_Sem the semaphore handle returned by myoSemCreate.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.2.2.28 MyoError myoSemWait (MyoSem *in_Sem*)

Decrements (locks) the semaphore. If the semaphore value is greater than zero, then the decrement proceeds and the function returns immediately, or else the call blocks until the semaphore value rises above zero.

Parameters:

in_Sem the semaphore handle returned by myoSemCreate.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.2.2.29 void myoSharedAlignedFree (void ∗ *in_pPtr*)

Frees the memory space got by myoArenaAlignedMalloc to the default arena.

Parameters:

in_pPtr The start address of the specified memory space, which must be retured by myoArenaAlignedMalloc.

Returns:

4.2.2.30 void ∗ myoSharedAlignedMalloc (size_t *in_Size*, size_t *in_Alignment*)

Allocates size bytes from the default arena. The start address of the allocated memory will be a multiple of alignment, which must be a power of two.

Parameters:

in_Size Size (bytes) of the required memory space.

in_Alignment The alignment value, which must be an power of two.

Returns:

The start address of the allocated memory space. NULL: Failed.

4.2.2.31 void myoSharedFree (void ∗ *in_pPtr*)

Frees the memory space got by myoArenaMalloc to the default arena.

Parameters:

in_pPtr The start address of the specified memory space, which must be retured by myoSharedMalloc.

Returns:

4.2.2.32 void ∗ myoSharedMalloc (size_t *in_Size*)

There will be a default arena inside MYO runtime, which will be used when there is no specified arena. Allocates size bytes from the default arena. and returns the start address of the allocated memory. The memory is not cleared.

Parameters:

in_Size Size (bytes) of the required memory space.

Returns:

The start address of the allocated memory space. NULL: Failed.

4.3 MYOTYPES

Description: Define the types used by APIs of MYO programming.

Files

- file myotypes.h

Defines

- #define MYO_CONSISTENCY 0x200
- #define MYO_CONSISTENCY_MODE 0x3
- #define MYO_DEVICE_TO_HOST 0x1000
- #define MYO_HOST_TO_DEVICE 0x800
- #define MYO_HYBRID_UPDATE 0x2000
- #define MYO_MULTI_VERSIONS 0x100
- #define MYO_NO_CONSISTENCY 0x400
- #define MYO_NOT_RECORD_DIRTY 0x40
- #define MYO_ONE_VERSION 0x80
- #define MYO_RECORD_DIRTY 0x20
- #define MYO_RELEASE_CONSISTENCY 0x1
- #define MYO_STRONG_CONSISTENCY 0x3
- #define MYO_STRONG_RELEASE_CONSISTENCY 0x2
- #define MYO_UPDATE_ON_ACQUIRE 0x10
- #define MYO_UPDATE_ON_DEMAND 0x8

Typedefs

- typedef unsigned int MyoArena
- typedef void $*$ MyoBarrier
- typedef void $*$ MyoMutex
- typedef void $*$ MyoSem

Enumerations

- enum MyoError {
  MYO_SUCCESS = 0,
  MYO_ERROR,
  MYO_INVALID_ENV,
  MYO_INVALID_ARGUMENT,
  MYO_NOT_INITIALIZED,
  MYO_ALREADY_FINALIZED,
  MYO_BUF_ERROR,
  MYO_OUT_OF_RANGE,
  MYO_OUT_OF_MEMORY,
  MYO_ALREADY_EXISTS,
  MYO_EOF }
- enum MyoOwnershipType {
  MYO_ARENA_MINE = 1,
  MYO_ARENA_OURS }

4.3.1 Detailed Description

Description: Define the types used by APIs of MYO programming.

4.3.2 Define Documentation

4.3.2.1 #define MYO_CONSISTENCY 0x200

Definition at line 62 of file myotypes.h.

4.3.2.2 #define MYO_CONSISTENCY_MODE 0x3

Definition at line 52 of file myotypes.h.

4.3.2.3 #define MYO_DEVICE_TO_HOST 0x1000

Definition at line 65 of file myotypes.h.

4.3.2.4 #define MYO_HOST_TO_DEVICE 0x800

Definition at line 64 of file myotypes.h.

4.3.2.5 #define MYO_HYBRID_UPDATE 0x2000

Definition at line 66 of file myotypes.h.

4.3.2.6 #define MYO_MULTI_VERSIONS 0x100

Definition at line 61 of file myotypes.h.

4.3.2.7 #define MYO_NO_CONSISTENCY 0x400

Definition at line 63 of file myotypes.h.

4.3.2.8 #define MYO_NOT_RECORD_DIRTY 0x40

Definition at line 59 of file myotypes.h.

4.3.2.9 #define MYO_ONE_VERSION 0x80

Definition at line 60 of file myotypes.h.

4.3.2.10 #define MYO_RECORD_DIRTY 0x20

Definition at line 58 of file myotypes.h.

4.3.2.11 #define MYO_RELEASE_CONSISTENCY 0x1

Definition at line 53 of file myotypes.h.

4.3.2.12 #define MYO_STRONG_CONSISTENCY 0x3

Definition at line 55 of file myotypes.h.

4.3.2.13 #define MYO_STRONG_RELEASE_CONSISTENCY 0x2

Definition at line 54 of file myotypes.h.

4.3.2.14 #define MYO_UPDATE_ON_ACQUIRE 0x10

Definition at line 57 of file myotypes.h.

4.3.2.15 #define MYO_UPDATE_ON_DEMAND 0x8

Definition at line 56 of file myotypes.h.

4.3.3 Typedef Documentation

4.3.3.1 typedef unsigned int MyoArena

Definition at line 67 of file myotypes.h.

4.3.3.2 typedef void∗ MyoBarrier

Definition at line 71 of file myotypes.h.

4.3.3.3 typedef void∗ MyoMutex

Definition at line 69 of file myotypes.h.

4.3.3.4 typedef void∗ MyoSem

Definition at line 70 of file myotypes.h.

4.3.4 Enumeration Type Documentation

4.3.4.1 enum MyoError

MYO Status

Enumerator:

MYO_SUCCESS Success
MYO_ERROR Error
MYO_INVALID_ENV Invalid Env
MYO_INVALID_ARGUMENT Invalid Argument
MYO_NOT_INITIALIZED Not Initialized
MYO_ALREADY_FINALIZED Already Finalized
MYO_BUF_ERROR Buffer Error
MYO_OUT_OF_RANGE Out of Range
MYO_OUT_OF_MEMORY Out of Memory
MYO_ALREADY_EXISTS Already Exists
MYO_EOF EOF

Definition at line 23 of file myotypes.h.

4.3.4.2 enum MyoOwnershipType

Arena Ownership

Enumerator:

MYO_ARENA_MINE Arena MINE Ownership
MYO_ARENA_OURS Arena OURS Ownership

Definition at line 44 of file myotypes.h.

4.4 MYOIMPL_API

Description: Define APIs of MYO for compiler or pre-processor to transfer original programs.

Data Structures

- struct MyoiHostSharedFptrEntry

  Host Side Shared Function Pointer Entry Struct.

- struct MyoiHostSharedVarEntry

  Structure of the var table entry on host.

- struct MyoiMicSharedVarEntry

  Structure of the var table entry on card.

- struct MyoiSharedVarEntry

  This is structure of the Shared var table entry.

- struct MyoiTargetSharedFptrEntry

  Target Side Shared Function Pointer Entry Struct.

- struct MyoiUserParams

  Structure of the array element that is passed to myoiLibInit() to initialize a subset of the available cards.

Files

- file myoimpl.h

Defines

- #define EXTERN_C
- #define MYOACCESSAPI
- #define MYOI_USERPARAMS_DEVID 1
- #define MYOI_USERPARAMS_LAST_MSG -1

Typedefs

- typedef void ∗(∗ MyoiRemoteFuncType)(void ∗)
- typedef void ∗ MyoiRFuncCallHandle

Functions

- MyoError myoiCheckResult (MyoiRFuncCallHandle in_Handle)

  Check whether the remote call is done.

- MYOACCESSAPI MyoError myoiGetResult (MyoiRFuncCallHandle in_Handle)

  Wait till the remote call is done.

- MYOACCESSAPI MyoError myoiHostSharedMallocTableRegister (void ∗in_pAddrOfSVarTable, int in_NumEntry, int in_Ordered)

  Allocate shared memory for all shared variables in the table.

- MyoError myoiHostVarTablePropagate (void ∗in_pAddrOfSVarTable, int in_NumEntry)

  Send the host side var table to card side.

- MYOACCESSAPI void myoiLibFini ()

  Finalize the MYO library, all resources held by the runtime are released by this routine.

- MYOACCESSAPI MyoError myoiLibInit (void ∗in_args, void ∗userInitFunc)

  Init entry of the MYO library responsible for initializing the runtime.

- MYOACCESSAPI MyoiRFuncCallHandle myoiRemoteCall (const char ∗in_pFuncName, void ∗in_pArgs, int in_deviceNum)
- MYOACCESSAPI MyoError myoiRemoteFuncLookupByAddr (MyoiRemoteFuncType in_-pWrapFuncAddr, char ∗∗out_pFuncName)

  Get the name of a remote function by looking up the table by the address.

- MYOACCESSAPI MyoError myoiRemoteFuncLookupByName (char ∗in_pFuncName, MyoiRe-moteFuncType ∗out_pWrapFuncAddr)

  Get the address of the wrapper function by looking up the table by the name.

- MYOACCESSAPI MyoError myoiRemoteFuncRegister (MyoiRemoteFuncType in_pFuncAddr, const char ∗in_pFuncName)

  Register a function so that it can be remotely called.

- MYOACCESSAPI MyoError myoiSetMemConsistent (void ∗in_pAddr, size_t in_Size)

  Set part of the shared memory space to be consistent.

- MYOACCESSAPI MyoError myoiSetMemNonConsistent (void ∗in_pAddr, size_t in_Size)

  Set part of the shared memory space to be non-consistent.

- MyoError myoiTargetSharedMallocTableRegister (void ∗in_pAddrOfSVarTable, int in_NumEntry, int in_Ordered)

  Register the shared variables in target side.

- MYOACCESSAPI MyoError myoiVarRegister (void ∗in_pAddrOfLocalPtrToShared, const char ∗in_pSVarName)

Variables

- EXTERN_C MYOACCESSAPI volatile int myoiInitFlag
- EXTERN_C MYOACCESSAPI unsigned int myoiMyId

4.4.1 Detailed Description

Description: Define APIs of MYO for compiler or pre-processor to transfer original programs.

4.4.2 Define Documentation

4.4.2.1 #define EXTERN_C

Definition at line 20 of file myoimpl.h.

4.4.2.2 #define MYOACCESSAPI

Definition at line 32 of file myoimpl.h.

4.4.2.3 #define MYOI_USERPARAMS_DEVID 1

Definition at line 461 of file myoimpl.h.

4.4.2.4 #define MYOI_USERPARAMS_LAST_MSG -1

Definition at line 462 of file myoimpl.h.

4.4.3 Typedef Documentation

4.4.3.1 typedef void∗(∗ MyoiRemoteFuncType)(void ∗)

Definition at line 39 of file myoimpl.h.

4.4.3.2 typedef void∗ MyoiRFuncCallHandle

Definition at line 149 of file myoimpl.h.

4.4.4 Function Documentation

4.4.4.1 MyoError myoiCheckResult (MyoiRFuncCallHandle *in_Handle*)

Check whether the remote call is done.

Parameters:

in_Handle handle of the remote call.

Returns:

MYO_SUCCESS (done); or an error number to indicate the error.

4.4.4.2 MyoError myoiGetResult (MyoiRFuncCallHandle *in_Handle*)

Wait till the remote call is done.

Parameters:

in_Handle handle of the remote call.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.4.4.3 MyoError myoiHostSharedMallocTableRegister (void ∗ *in_pAddrOfSVarTable*, int *in_NumEntry*, int *in_Ordered*)

Allocate shared memory for all shared variables in the table. Also update local address of the shared variable with new shared address.

Parameters:

in_pAddrOfSVarTable start address of the shared varaible table. Assuming it follows the format of MyoiHostSharedVarEntry.

in_NumEntry number of entry in the table.

in_Ordered whether the table ordered by name.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.4.4.4 MyoError myoiHostVarTablePropagate (void ∗ *in_pAddrOfSVarTable*, int *in_NumEntry*)

Send the host side var table to card side. Card side will also have a copy of host side var table after this propagation, although it is in an internal format different than original host side var table due to implementation convenience.

Parameters:

in_pAddrOfSVarTable start address of the host side var table. Assuming it follows the format of MyoiSharedVarEntry.

in_NumEntry number of entry in the table.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.4.4.5 void myoiLibFini ()

Finalize the MYO library, all resources held by the runtime are released by this routine.

Returns:

4.4.4.6 MyoError myoiLibInit (void ∗ *in_args*, void ∗ *userInitFunc*)

Init entry of the MYO library responsible for initializing the runtime.

Parameters:

in_args mechanism to pass arguments to the Initialization routine.The default value of NULL would mean the host is blocked on the completion of myoiLibInit() on allthe nodes. A subset of the installed cards can be intialized by passing an array of MyoiUserParams's.For example, In a system with two cards, if you would like to run a MYO application only on the second card you would intialize the array as follows:

```
MyoiUserParams UserParas[64];
UserParas[0].type = MYOI_USERPARAMS_DEVID;
UserParas[0].nodeid = 2;
UserParas[1].type = MYOI_USERPARAMS_LAST_MSG;
if(MYO_SUCCESS != myoiLibInit(&UserParas, (void*)&myoiUserInit)) {
    printf("Failed to initialize MYO runtime\n");
    return -1;
}
```

This intialization is required only in the client/host side of the application. The server/card side executable should be executed only on the second card in this case.

userInitFunc Shared variables and remote funtions are registered in this routine which is called by the runtime during library initialization

Returns:

MYO_SUCCESS; MYO_ERROR;

4.4.4.7 MYOACCESSAPI MyoiRFuncCallHandle myoiRemoteCall (const char ∗ *in_pFuncName*, void ∗ *in_pArgs*, int *in_deviceNum*)

4.4.4.8 MyoError myoiRemoteFuncLookupByAddr (MyoiRemoteFuncType *in_pWrapFuncAddr*, char ∗∗ *out_pFuncName*)

Get the name of a remote function by looking up the table by the address. This API can be used when calling a remotely callable function by a function pointer.

Parameters:

in_pWrapFuncAddr address of the function.

out_pFuncName name of the function.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.4.4.9 MyoError myoiRemoteFuncLookupByName (char ∗ *in_pFuncName*, MyoiRemoteFuncType ∗ *out_pWrapFuncAddr*)

Get the address of the wrapper function by looking up the table by the name. This API can be used when before assigning a function pointer pointing to remotely callable functions.

Parameters:

in_pFuncName name of the function.
out_pWrapFuncAddr address of the wrapper function.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.4.4.10 MyoError myoiRemoteFuncRegister (MyoiRemoteFuncType *in_pFuncAddr*, const char ∗ *in_pFuncName*)

Register a function so that it can be remotely called. This should be done in myoiUserInit or before calling myoiLibInit. After myoiLibInit, there will be a table on all peers which containing the information for all remotely callable function.

Parameters:

in_pWrapFuncAddr address of the wrapper function.
in_pFuncName name of the function.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.4.4.11 MyoError myoiSetMemConsistent (void ∗ *in_pAddr*, size_t *in_Size*)

Set part of the shared memory space to be consistent. which means that the consistency of this part of shared memory space need to be maintained between HOST and cards.

Parameters:

in_pAddr The start address of the specified shared memory space;
in_size The size of the specified shared memory space;

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.4.4.12 MyoError myoiSetMemNonConsistent (void ∗ *in_pAddr*, size_t *in_Size*)

Set part of the shared memory space to be non-consistent. which means that the consistency of this part of shared memory space does not need to be maintained between HOST and cards.

Parameters:

in_pAddr The start address of the specified shared memory space;

in_Size The size of the specified shared memory space;

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.4.4.13 MyoError myoiTargetSharedMallocTableRegister (void ∗ *in_pAddrOfSVarTable*, int *in_NumEntry*, int *in_Ordered*)

Register the shared variables in target side.

Parameters:

in_pAddrOfSVarTable start address of the shared varaible table. Assuming it follows the format of MyoiMicSharedVarEntry.

in_NumEntry number of entry in the table.

in_Ordered whether the table ordered by name.

Returns:

MYO_SUCCESS; or an error number to indicate the error.

4.4.4.14 MYOACCESSAPI MyoError myoiVarRegister (void ∗ *in_pAddrOfLocalPtrToShared*, const char ∗ *in_pSVarName*)

4.4.5 Variable Documentation

4.4.5.1 EXTERN_C MYOACCESSAPI volatile int myoiInitFlag

Definition at line 450 of file myoimpl.h.

4.4.5.2 EXTERN_C MYOACCESSAPI unsigned int myoiMyId

Definition at line 449 of file myoimpl.h.

5 Data Structure Documentation

5.1 MyoiHostSharedFptrEntry Struct Reference

Host Side Shared Function Pointer Entry Struct.

Data Fields

- void ∗ funcAddr

  Function Address.

- const char ∗ funcName

  Function Name.

- void ∗ localThunkAddr

  Local Thunk Address.

5.1.1 Detailed Description

Host Side Shared Function Pointer Entry Struct.

Definition at line 83 of file myoimpl.h.

5.1.2 Field Documentation

5.1.2.1 void∗ MyoiHostSharedFptrEntry::funcAddr

Function Address.

Definition at line 87 of file myoimpl.h.

5.1.2.2 const char∗ MyoiHostSharedFptrEntry::funcName

Function Name.

Definition at line 85 of file myoimpl.h.

5.1.2.3 void∗ MyoiHostSharedFptrEntry::localThunkAddr

Local Thunk Address.

Definition at line 89 of file myoimpl.h.

5.2 MyoiHostSharedVarEntry Struct Reference

Structure of the var table entry on host.

Data Fields

- void ∗ ptrToLocalPtrToShared

  Local pointer to Shared var.

- int size

  Variable Size.

- const char ∗ varName

  Variable Name.

5.2.1 Detailed Description

Structure of the var table entry on host.

Definition at line 308 of file myoimpl.h.

5.2.2 Field Documentation

5.2.2.1 void∗ MyoiHostSharedVarEntry::ptrToLocalPtrToShared

Local pointer to Shared var.

Definition at line 314 of file myoimpl.h.

5.2.2.2 int MyoiHostSharedVarEntry::size

Variable Size.

Definition at line 312 of file myoimpl.h.

5.2.2.3 const char∗ MyoiHostSharedVarEntry::varName

Variable Name.

Definition at line 310 of file myoimpl.h.

5.3 MyoiMicSharedVarEntry Struct Reference

Structure of the var table entry on card.

Data Fields

- void ∗ ptrToLocalPtrToShared

  Local pointer to Shared var.

- const char ∗ varName

  Variable Name.

5.3.1 Detailed Description

Structure of the var table entry on card.

Definition at line 318 of file myoimpl.h.

5.3.2 Field Documentation

5.3.2.1 void∗ MyoiMicSharedVarEntry::ptrToLocalPtrToShared

Local pointer to Shared var.

Definition at line 322 of file myoimpl.h.

5.3.2.2 const char∗ MyoiMicSharedVarEntry::varName

Variable Name.

Definition at line 320 of file myoimpl.h.

5.4 MyoiSharedVarEntry Struct Reference

This is structure of the Shared var table entry.

Data Fields

- void ∗ sharedAddr

  Shared Address.

- const char ∗ varName

  Variable Name.

5.4.1 Detailed Description

This is structure of the Shared var table entry. This table contains the shared address and name of each shared variable

Definition at line 300 of file myoimpl.h.

5.4.2 Field Documentation

5.4.2.1 void∗ MyoiSharedVarEntry::sharedAddr

Shared Address.

Definition at line 304 of file myoimpl.h.

5.4.2.2 const char∗ MyoiSharedVarEntry::varName

Variable Name.

Definition at line 302 of file myoimpl.h.

5.5 MyoiTargetSharedFptrEntry Struct Reference

Target Side Shared Function Pointer Entry Struct.

Data Fields

- void ∗ funcAddr

  Function Address.

- const char ∗ funcName

  Function Name.

- void ∗ localThunkAddr

  Locak Thunk Address.

- void ∗ wrapFuncAddr

  Wrap Function Address.

5.5.1 Detailed Description

Target Side Shared Function Pointer Entry Struct.

Definition at line 93 of file myoimpl.h.

5.5.2 Field Documentation

5.5.2.1 void∗ MyoiTargetSharedFptrEntry::funcAddr

Function Address.

Definition at line 97 of file myoimpl.h.

5.5.2.2 const char∗ MyoiTargetSharedFptrEntry::funcName

Function Name.

Definition at line 95 of file myoimpl.h.

5.5.2.3 void∗ MyoiTargetSharedFptrEntry::localThunkAddr

Locak Thunk Address.

Definition at line 101 of file myoimpl.h.

5.5.2.4 void∗ MyoiTargetSharedFptrEntry::wrapFuncAddr

Wrap Function Address.

Definition at line 99 of file myoimpl.h.

5.6 MyoiUserParams Struct Reference

Structure of the array element that is passed to myoiLibInit() to initialize a subset of the available cards.

Data Fields

- int nodeid

  nodeid refers to the card index.

- int type

  type = MYOI_USERPARAMS_DEVID for each element in the array except the last element ; type = MYOI_-USERPARAMS_LAST_MSG for the last element in the array.

5.6.1 Detailed Description

Structure of the array element that is passed to myoiLibInit() to initialize a subset of the available cards.

Definition at line 454 of file myoimpl.h.

5.6.2 Field Documentation

5.6.2.1 int MyoiUserParams::nodeid

nodeid refers to the card index.

Definition at line 458 of file myoimpl.h.

5.6.2.2 int MyoiUserParams::type

type = MYOI_USERPARAMS_DEVID for each element in the array except the last element ; type = MYOI_USERPARAMS_LAST_MSG for the last element in the array.

Definition at line 456 of file myoimpl.h.

6 File Documentation

6.1 myo.h File Reference

Functions

- MYOACCESSAPI MyoError myoAcquire ()

  myoAcquire are the sync points for the default arena with "Release Consistency".

- MYOACCESSAPI MyoError myoAcquireOwnership ()

  Changes the ownership type of the default arena to MYO_ARENA_MINE.

- MYOACCESSAPI MyoError myoArenaAcquire (MyoArena in_Arena)

  myoArenaAcquire are the sync points for "OURS" arena with "Release Consistency".

- MYOACCESSAPI MyoError myoArenaAcquireOwnership (MyoArena in_Arena)

  Changes the ownership type of the arena to MYO_ARENA_MINE.

- MYOACCESSAPI void myoArenaAlignedFree (MyoArena in_Arena, void ∗in_pPtr)

  Frees the memory space got by myoArenaAlignedMalloc to the specified arena.

- MYOACCESSAPI void ∗ myoArenaAlignedMalloc (MyoArena in_Arena, size_t in_Size, size_t in_Alignment)

  Allocates size bytes from the specified arena.

- MYOACCESSAPI MyoError myoArenaCreate (MyoOwnershipType in_Type, int in_Property, MyoArena ∗out_pArena)

  Create an arena with specified ownership type and property.

- MYOACCESSAPI MyoError myoArenaDestroy (MyoArena in_Arena)

  Destroy an arena.

- MYOACCESSAPI void myoArenaFree (MyoArena in_Arena, void ∗in_pPtr)

  Frees the memory space got by myoArenaMalloc to the specified arena.

- MYOACCESSAPI MyoError myoArenaGetHandle (void ∗in_pPtr, MyoArena ∗out_pArena)

  Gets the arena handle of the arena which contains the memory space "in_pPtr" points to.

- MYOACCESSAPI void ∗ myoArenaMalloc (MyoArena in_Arena, size_t in_Size)

  Allocates size bytes from the specified arena.

- MYOACCESSAPI MyoError myoArenaRelease (MyoArena in_Arena)

  myoArenaRelease are the sync points for "OURS" arena with "Release Consistency".

- MYOACCESSAPI MyoError myoArenaReleaseOwnership (MyoArena in_Arena)

  Change the ownership type of the arena to MYO_ARENA_OURS.

- MYOACCESSAPI MyoError myoBarrierCreate (int in_Count, MyoBarrier ∗out_pBarrier)

  Create a barrier and return the barrier handle.

- MYOACCESSAPI MyoError myoBarrierDestroy (MyoBarrier in_Barrier)

  Destroy the barrier.

- MYOACCESSAPI MyoError myoBarrierWait (MyoBarrier in_Barrier)

  The caller will block until the required number of threads have called myoBarrierWait with the same barrier handle.

- MYOACCESSAPI MyoError myoMutexCreate (MyoMutex ∗out_pMutex)

  Create a mutex and return the mutex handle.

- MYOACCESSAPI MyoError myoMutexDestroy (MyoMutex in_Mutex)

  Destroy the mutex.

- MYOACCESSAPI MyoError myoMutexLock (MyoMutex in_Mutex)

  Lock the mutex.

- MYOACCESSAPI MyoError myoMutexTryLock (MyoMutex in_Mutex)

  Try to lock the mutex.

- MYOACCESSAPI MyoError myoMutexUnlock (MyoMutex in_Mutex)

  Release the locked mutex.

- MYOACCESSAPI MyoError myoRelease ()

  myoRelease are the sync points for the default arena with ∗ "Release Consistency".

- MYOACCESSAPI MyoError myoReleaseOwnership ()

  Change the ownership type of the default arena to the MYO_ARENA_OURS.

- MYOACCESSAPI MyoError myoSemCreate (int in_Value, MyoSem ∗out_pSem)

  Create a semaphore and return the semaphore handle.

- MYOACCESSAPI MyoError myoSemDestroy (MyoSem in_Sem)

  Destroy the semaphore.

- MYOACCESSAPI MyoError myoSemPost (MyoSem in_Sem)

  Increments (unlocks) the semaphore.

- MYOACCESSAPI MyoError myoSemTryWait (MyoSem in_Sem)

  Try to lock semaphore.

- MYOACCESSAPI MyoError myoSemWait (MyoSem in_Sem)

Decrements (locks) the semaphore.

- MYOACCESSAPI void myoSharedAlignedFree (void ∗in_pPtr)

  Frees the memory space got by myoArenaAlignedMalloc to the default arena.

- MYOACCESSAPI void ∗ myoSharedAlignedMalloc (size_t in_Size, size_t in_Alignment)

  Allocates size bytes from the default arena.

- MYOACCESSAPI void myoSharedFree (void ∗in_pPtr)

  Frees the memory space got by myoArenaMalloc to the default arena.

- MYOACCESSAPI void ∗ myoSharedMalloc (size_t in_Size)

  There will be a default arena inside MYO runtime, which will be used when there is no specified arena.

6.1.1 Detailed Description

Definition in file myo.h.

6.2 myoimpl.h File Reference

Data Structures

- struct MyoiHostSharedFptrEntry

  Host Side Shared Function Pointer Entry Struct.

- struct MyoiHostSharedVarEntry

  Structure of the var table entry on host.

- struct MyoiMicSharedVarEntry

  Structure of the var table entry on card.

- struct MyoiSharedVarEntry

  This is structure of the Shared var table entry.

- struct MyoiTargetSharedFptrEntry

  Target Side Shared Function Pointer Entry Struct.

- struct MyoiUserParams

  Structure of the array element that is passed to myoiLibInit() to initialize a subset of the available cards.

Defines

- #define EXTERN_C
- #define MYOACCESSAPI
- #define MYOI_USERPARAMS_DEVID 1
- #define MYOI_USERPARAMS_LAST_MSG -1

Typedefs

- typedef void ∗(∗ MyoiRemoteFuncType)(void ∗)
- typedef void ∗ MyoiRFuncCallHandle

Functions

- MyoError myoiCheckResult (MyoiRFuncCallHandle in_Handle)

  Check whether the remote call is done.

- MYOACCESSAPI MyoError myoiGetResult (MyoiRFuncCallHandle in_Handle)

  Wait till the remote call is done.

- MYOACCESSAPI MyoError myoiHostSharedMallocTableRegister (void ∗in_pAddrOfSVarTable, int in_NumEntry, int in_Ordered)

  Allocate shared memory for all shared variables in the table.

- MyoError myoiHostVarTablePropagate (void ∗in_pAddrOfSVarTable, int in_NumEntry)

  Send the host side var table to card side.

- MYOACCESSAPI void myoiLibFini ()

  Finalize the MYO library, all resources held by the runtime are released by this routine.

- MYOACCESSAPI MyoError myoiLibInit (void ∗in_args, void ∗userInitFunc)

  Init entry of the MYO library responsible for initializing the runtime.

- MYOACCESSAPI MyoiRFuncCallHandle myoiRemoteCall (const char ∗in_pFuncName, void ∗in_pArgs, int in_deviceNum)
- MYOACCESSAPI MyoError myoiRemoteFuncLookupByAddr (MyoiRemoteFuncType in_-pWrapFuncAddr, char ∗∗out_pFuncName)

  Get the name of a remote function by looking up the table by the address.

- MYOACCESSAPI MyoError myoiRemoteFuncLookupByName (char ∗in_pFuncName, MyoiRe-moteFuncType ∗out_pWrapFuncAddr)

  Get the address of the wrapper function by looking up the table by the name.

- MYOACCESSAPI MyoError myoiRemoteFuncRegister (MyoiRemoteFuncType in_pFuncAddr, const char ∗in_pFuncName)

  Register a function so that it can be remotely called.

- MYOACCESSAPI MyoError myoiSetMemConsistent (void ∗in_pAddr, size_t in_Size)

  Set part of the shared memory space to be consistent.

- MYOACCESSAPI MyoError myoiSetMemNonConsistent (void ∗in_pAddr, size_t in_Size)

  Set part of the shared memory space to be non-consistent.

- MyoError myoiTargetSharedMallocTableRegister (void ∗in_pAddrOfSVarTable, int in_NumEntry, int in_Ordered)

  Register the shared variables in target side.

- MYOACCESSAPI MyoError myoiVarRegister (void ∗in_pAddrOfLocalPtrToShared, const char ∗in_pSVarName)

Variables

- EXTERN_C MYOACCESSAPI volatile int myoiInitFlag
- EXTERN_C MYOACCESSAPI unsigned int myoiMyId

6.2.1 Detailed Description

Definition in file myoimpl.h.

6.3 myotypes.h File Reference

Defines

- #define MYO_CONSISTENCY 0x200
- #define MYO_CONSISTENCY_MODE 0x3
- #define MYO_DEVICE_TO_HOST 0x1000
- #define MYO_HOST_TO_DEVICE 0x800
- #define MYO_HYBRID_UPDATE 0x2000
- #define MYO_MULTI_VERSIONS 0x100
- #define MYO_NO_CONSISTENCY 0x400
- #define MYO_NOT_RECORD_DIRTY 0x40
- #define MYO_ONE_VERSION 0x80
- #define MYO_RECORD_DIRTY 0x20
- #define MYO_RELEASE_CONSISTENCY 0x1
- #define MYO_STRONG_CONSISTENCY 0x3
- #define MYO_STRONG_RELEASE_CONSISTENCY 0x2
- #define MYO_UPDATE_ON_ACQUIRE 0x10
- #define MYO_UPDATE_ON_DEMAND 0x8

Typedefs

- typedef unsigned int MyoArena
- typedef void ∗ MyoBarrier
- typedef void ∗ MyoMutex
- typedef void ∗ MyoSem

Enumerations

- enum MyoError {
  MYO_SUCCESS = 0,
  MYO_ERROR,
  MYO_INVALID_ENV,
  MYO_INVALID_ARGUMENT,
  MYO_NOT_INITIALIZED,
  MYO_ALREADY_FINALIZED,
  MYO_BUF_ERROR,
  MYO_OUT_OF_RANGE,

MYO_OUT_OF_MEMORY,
MYO_ALREADY_EXISTS,
MYO_EOF }

- enum MyoOwnershipType {
  MYO_ARENA_MINE = 1,
  MYO_ARENA_OURS }

6.3.1 Detailed Description

Definition in file myotypes.h.

Index