STAPLER Reference Manual

Generated by Doxygen 1.4.7

Wed Feb 23 13:34:28 2011

Contents

1 ST Abstraction PLatform LayER.

STAPLER: Provides an OS-abstraction layer for security drivers

Version:

1.7.3

1.1 API sections

API types

API constants

Stapler management functions

Driver registration and lookup functions

Memory management functions

Interrupt functions

Semaphore functions

Mutex functions

Timer functions

Task Functions

Chip functions

Stapler queue functions

2 STAPLER Module Index

2.1 STAPLER Modules

Here is a list of all modules:

3 STAPLER Class Index

3.1 STAPLER Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

4 STAPLER Module Documentation

4.1 API types

Defines

- #define STAPLER_Time_t unsigned long

  Type used to store time values.

- #define STAPLER_DEFAULT_MODE STAPLER_INTERRUPT_POLL

  The default mode stapler will be running in.

Typedefs

- typedef U32 STAPLER_DriverHandle_t
  Handle used to reference a driver.
- typedef U32 STAPLER_PartitionHandle_t
  Handle used to reference a memory partition.
- typedef U32 STAPLER_SemaphoreHandle_t
  Handle used to reference a semaphore.
- typedef U32 STAPLER_MutexHandle_t
  Handle used to reference a mutex.
- typedef U32 STAPLER_Task_t
  Handle used to reference a task.
- typedef U32 STAPLER_Size_t
  type used to reference a size
- typedef U32 STAPLER_TDesc_t
  type used to reference a task descriptor
- typedef U32 STAPLER_Task_Flags_t
  type used to reference task flags
- typedef U32 STAPLER_Queue_t
  type used to reference a Queue

4.2 API constants

Classes

- struct STAPLER_Init_t
  Initialsation paramameters - currently only one!

Defines

- #define STAPLER_REVISION "STAPLER-REL_1.7.3"
- #define **STAPLER_BAD_DRIVER_HANDLE** 0

Enumerations

- enum STAPLER_Error_t {
  STAPLER_NO_ERROR = 0,
  STAPLER_ERROR_BAD_PARAMETER,
  STAPLER_ERROR_NO_MEMORY,

STAPLER_ERROR_UNKNOWN_DEVICE,
STAPLER_ERROR_ALREADY_INITIALIZED,
STAPLER_ERROR_NO_FREE_HANDLES,
STAPLER_ERROR_OPEN_HANDLE,
STAPLER_ERROR_INVALID_HANDLE,
STAPLER_ERROR_FEATURE_NOT_SUPPORTED,
STAPLER_ERROR_INTERRUPT_INSTALL,
STAPLER_ERROR_INTERRUPT_UNINSTALL,
STAPLER_ERROR_TIMEOUT,
STAPLER_ERROR_DEVICE_BUSY,
STAPLER_ERROR_NOT_SUPPORTED = 0x1000,
STAPLER_ERROR_DRIVER_NOT_INITIALIZED,
STAPLER_ERROR_INTERRUPT_ENABLE,
STAPLER_ERROR_INTERRUPT_DISABLE,
STAPLER_ERROR_CAN_NOT_FIND_INTERRUPT,
STAPLER_ERROR_NO_FREE_SLOT,
STAPLER_ERROR_SEMAPHORE_MAKE,
STAPLER_ERROR_MUTEX_MAKE,
STAPLER_ERROR_MUTEX_LOCKED }

STAPLER driver-specific return codes.

- enum STAPLER_Interrupt_Mode_t {
  STAPLER_INTERRUPT_POLL,
  STAPLER_INTERRUPT_REAL }

  The interrupt mode in which STAPLER is running.

- enum STAPLER_Build_Mode_t {
  STAPLER_BUILD_MODE_STOS,
  STAPLER_BUILD_MODE_OS20,
  STAPLER_BUILD_MODE_OS21,
  STAPLER_BUILD_MODE_BARE }

  The mode in which STAPLER was built.

- enum STAPLER_Semaphore_Types_t {
  STAPLER_SEMAPHORE_TYPE_INTERRUPT,
  STAPLER_SEMAPHORE_TYPE_PROTECTION }

  The Semaphore types.

- enum STAPLER_InterruptNumber_t {
  STAPLER_INTERRUPT_NUMBER_TANGO_SC_MBX0 = 1,
  STAPLER_INTERRUPT_NUMBER_TANGO_SC_MBX1 = 2 }

  Security driver fixed interrupt numbers.

4.2.1 Define Documentation

4.2.1.1 #define STAPLER_REVISION "STAPLER-REL_1.7.3"

The base revision string for this version of STAPLER

4.2.2 Enumeration Type Documentation

4.2.2.1 enum STAPLER_Build_Mode_t

The mode in which STAPLER was built.

Enumerator:

STAPLER_BUILD_MODE_STOS STAPLER built in stos mode
STAPLER_BUILD_MODE_OS20 STAPLER built in native OS20 mode
STAPLER_BUILD_MODE_OS21 STAPLER built in native OS21 mode
STAPLER_BUILD_MODE_BARE STAPLER built in bare mode

4.2.2.2 enum STAPLER_Error_t

STAPLER driver-specific return codes.

Enumerator:

STAPLER_NO_ERROR No error.
STAPLER_ERROR_BAD_PARAMETER An invalid function parameter was passed.
STAPLER_ERROR_NO_MEMORY Memory allocation failed through lack of memeory.
STAPLER_ERROR_UNKNOWN_DEVICE Unknown device name.
STAPLER_ERROR_ALREADY_INITIALIZED The driver has already been initialised.
STAPLER_ERROR_NO_FREE_HANDLES Cannot open device again.
STAPLER_ERROR_OPEN_HANDLE At least one open handle.
STAPLER_ERROR_INVALID_HANDLE Handle is not valid.
STAPLER_ERROR_FEATURE_NOT_SUPPORTED Feature not supported.
STAPLER_ERROR_INTERRUPT_INSTALL An error occurred while installing an interrupt.
STAPLER_ERROR_INTERRUPT_UNINSTALL An error occurred while uninstalling an interrupt.
STAPLER_ERROR_TIMEOUT Timeout occured.
STAPLER_ERROR_DEVICE_BUSY Device is currently busy.
STAPLER_ERROR_NOT_SUPPORTED The requested operation is not supported.
STAPLER_ERROR_DRIVER_NOT_INITIALIZED The driver has not been intialised.
STAPLER_ERROR_INTERRUPT_ENABLE An error occurred while enabling an interrupt.
STAPLER_ERROR_INTERRUPT_DISABLE An error occurred while disabling an interrupt.
STAPLER_ERROR_CAN_NOT_FIND_INTERRUPT The Intterrupt cold not be found.
STAPLER_ERROR_NO_FREE_SLOT there is no free slots in the STAPLER data store.
STAPLER_ERROR_SEMAPHORE_MAKE Error making a semaphore.
STAPLER_ERROR_MUTEX_MAKE Error making a mutex.
STAPLER_ERROR_MUTEX_LOCKED Failed to get mutex - it is already locked

4.2.2.3 enum STAPLER_Interrupt_Mode_t

The interrupt mode in which STAPLER is running.

Enumerator:

STAPLER_INTERRUPT_POLL Use polling interrupts.

STAPLER_INTERRUPT_REAL Use real ISR code.

4.2.2.4 enum STAPLER_Semaphore_Types_t

The Semaphore types.

Enumerator:

STAPLER_SEMAPHORE_TYPE_INTERRUPT The semaphore is used to sync with an ISR.

STAPLER_SEMAPHORE_TYPE_PROTECTION The semaphore is used to proctect a block of code.

4.3 Stapler management functions

Functions

- STAPLER_Error_t STAPLER_Init (STAPLER_Init_t param)
  Driver initialisation function.
- STAPLER_Error_t STAPLER_Term (void)
  Terminates driver and frees any resouces it holds.
- const char * STAPLER_GetRevision (void)
  Returns a text string indicating the driver version.

4.3.1 Function Documentation

4.3.1.1 const char* STAPLER_GetRevision (void)

Returns a text string indicating the driver version.

Returns:

A printable string containing the driver version of the form: STAPLER-REL_-<major>.<minor>.<patch>.

4.3.1.2 STAPLER_Error_t STAPLER_Init (STAPLER_Init_t *param*)

Driver initialisation function.

This function must be called before any other API function except STAPLER_GetRevision()

- param - Initialisation structure.

Return values:

STAPLER_NO_ERROR
STAPLER_ERROR_ALREADY_INITIALIZED

4.3.1.3 STAPLER_Error_t STAPLER_Term (void)

Terminates driver and frees any resouces it holds.

Return values:

STAPLER_NO_ERROR
STAPLER_ERROR_DRIVER_NOT_INITIALIZED

4.4 Driver registration and lookup functions

Functions

- STAPLER_DriverHandle_t STAPLER_RegDriver (char ∗driverName, U32 driverId)
  Registers a driver with STAPLER.
- STAPLER_Error_t STAPLER_RegAPI (STAPLER_DriverHandle_t handle, U32 apiHandle, U32 data)
  Registers an API to the driver with STAPLER.
- STAPLER_DriverHandle_t STAPLER_FindDriver (U32 driverID)
  Gets a driver handle from a driver id.
- U32 STAPLER_FindAPI (STAPLER_DriverHandle_t handle, U32 apiHandle)
  Gets an API function pointer for a given API function in a driver.
- STAPLER_Build_Mode_t STAPLER_BuildMode (void)
  Gets the mode that STAPLER was built in.
- STAPLER_Error_t STAPLER_HandleAlloc (U32 DriverId, void ∗DriverData, U32 ∗SysHandle_p)
  Allocates a unique system handle for a driver.
- STAPLER_Error_t STAPLER_HandleFree (U32 SysHandle)
  Frees a system handle allocated with STAPLER_HandleAlloc.

4.4.1 Detailed Description

The Driver Registration mechanism allows client drivers to access API's of server drivers. A server driver such as STLOAD, registers itself and one or more API's. Drivers and API's are registered via unique handles known to the drivers involved at compile time. A client driver wishing to use a server driver API, checks first that the corresponding driver has been registered and then requests the API entry point. This mechanism is used by STTKDMA when loading firmware via STLOAD.

Note: No API version checking is performed. Server and client compatibility must be confirmed by the user.

4.4.2 Function Documentation

4.4.2.1 STAPLER_Build_Mode_t STAPLER_BuildMode (void)

Gets the mode that STAPLER was built in.

Returns:

The build mode that STAPLER was built in

4.4.2.2 U32 STAPLER_FindAPI (STAPLER_DriverHandle_t *handle*, U32 *apiHandle*)

Gets an API function pointer for a given API function in a driver.

- handle - The handle of the driver whose API to search
- apiHandle - The handle of the API function within the driver's API

Returns:

A function pointer to the required API function

4.4.2.3 STAPLER_DriverHandle_t STAPLER_FindDriver (U32 *driverID*)

Gets a driver handle from a driver id.

- driverID - The id of the driver for which to obtain a handle

Returns:

A handle to the required driver or STAPLER_BAD_DRIVER_HANDLE value if the handle is undefined

4.4.2.4 STAPLER_Error_t STAPLER_HandleAlloc (U32 *DriverId*, void * *DriverData*, U32 * *SysHandle_p*)

Allocates a unique system handle for a driver.

- DriverId - Driver identifier number
- DriverData - Point to driver data to be stored with this handle
- SysHandle_p - Pointer to U32 to be set with the allocated handle

Returns:

STAPLER error code

4.4.2.5 STAPLER_Error_t STAPLER_HandleFree (U32 *SysHandle*)

Frees a system handle allocated with STAPLER_HandleAlloc.

- SysHandle - The system handle to free

Returns:

STAPLER error code

4.4.2.6 STAPLER_Error_t STAPLER_RegAPI (STAPLER_DriverHandle_t *handle*, U32 *apiHandle*, U32 *data*)

Registers an API to the driver with STAPLER.

- handle - The handle of the driver registering the API function
- apiHandle - The handle of the API function to register. This is a number used to uniquely identify the api function within the driver's API
- data - A function pointer to the API function being registered.

Return values:

STAPLER_NO_ERROR

STAPLER_ERROR_BAD_PARAMETER

4.4.2.7 STAPLER_DriverHandle_t STAPLER_RegDriver (char * *driverName*, U32 *driverId*)

Registers a driver with STAPLER.

- driverName - The name of the driver to register
- driverId - The unique id of the driver to register

Returns:

A STAPLER_DriverHandle_t which may be used to reference the registered driver within the context of STAPLER

4.5 Memory management functions

Functions

- void * STAPLER_MemoryMap (U32 baseAddressPhy, U32 size, char *name, STAPLER_DriverHandle_t handle)

  Maps a physical address range into virtual memory space.

- void STAPLER_MemoryUnmap (void *baseAddressVirt, U32 size, STAPLER_DriverHandle_t handle)

  Unmaps a virtual address range mapped with STAPLER_MemoryMap().

- U32 STAPLER_MemoryVirtToPhy (void *addressVirt)

  Obtains the corresponding physical address from a virtual address.

- STAPLER_PartitionHandle_t STAPLER_MemoryMakePartition (void *baseAddressVirt, U32 size, char *name, STAPLER_DriverHandle_t handle)

  Creates a new memory partition in virtual memory space.

- void STAPLER_MemoryDeletePartition (STAPLER_PartitionHandle_t pHandle, STAPLER_DriverHandle_t dHandle)

  Deletes a memory partition created with STAPLER_MemoryMakePartition().

- STAPLER_PartitionHandle_t STAPLER_MemoryFindPartition (char *name)

  Locates a partition by name.

- void * STAPLER_MemoryAllocate (STAPLER_PartitionHandle_t pHandle, U32 size, STAPLER_DriverHandle_t dHandle, U32 ref)

  Allocates a buffer in virtual memory.

- void STAPLER_MemoryFree (STAPLER_PartitionHandle_t pHandle, void *address, STAPLER_DriverHandle_t dHandle, U32 ref)

  Frees a buffer allocated with STAPLER_MemoryAllocate().

- U32 STAPLER_MemoryCopy (void *destVirt, void *srcVirt, U32 size)

  Copies data between two memory buffers.

- STAPLER_Error_t STAPLER_MemoryCacheFlush (void *addressVirt, U32 size)

  Flushes a cached memory range.

- STAPLER_Error_t STAPLER_MemoryCacheInvalidate (void *addressVirt, U32 size)

  Invalidates a cached memory range.

- void * STAPLER_MemoryCachedToUncached (void *addressVirt, U32 size)

  Creates an uncached virtual address mapping for a range of cached virtual addresses.

- void * STAPLER_MemoryUncachedToCached (void *addressVirt, U32 size)

  Creates an cached virtual address mapping for a range of uncahced virtual addresses.

- STAPLER_Error_t STAPLER_MemoryUnmapCached (void *addressVirt, U32 size)

  Unmap a cached virtual address.

- STAPLER_Error_t STAPLER_MemoryUnmapUncached (void *addressVirt, U32 size)

  Unmap an uncached virtual address.

4.5.1 Function Documentation

4.5.1.1 void* STAPLER_MemoryAllocate (STAPLER_PartitionHandle_t *pHandle*, U32 *size*, STAPLER_DriverHandle_t *dHandle*, U32 *ref*)

Allocates a buffer in virtual memory.

- pHandle - A handle for the memory partition from which the buffer should be allocated
- size - The required size of the buffer
- dHandle - The handle of the driver requesting the buffer
- ref - A driver-specific reference number for the buffer

Returns:

A pointer to the allocated memory buffer, or NULL if the operation failed

See also:

also STAPLER_MemoryFree()

4.5.1.2 void* STAPLER_MemoryCachedToUncached (void * *addressVirt*, U32 *size*)

Creates an uncached virtual address mapping for a range of cached virtual addresses.

- addressVirt - Base address of the cached memory range to map
- size - Size of the address range

Returns:

A pointer to the base of the uncached memory range

See also:

also STAPLER_MemoryUncachedToCached()

4.5.1.3 STAPLER_Error_t STAPLER_MemoryCacheFlush (void * *addressVirt*, U32 *size*)

Flushes a cached memory range.

- addressVirt - Base address of the cached memory range to flush
- size - Size of the address range to flush

Return values:

STAPLER_NO_ERROR
STAPLER_ERROR_BAD_PARAMETER

4.5.1.4 STAPLER_Error_t STAPLER_MemoryCacheInvalidate (void * *addressVirt*, U32 *size*)

Invalidates a cached memory range.

Remarks:

Use with care. Usage should be reviewed.

- addressVirt - Base address of the cached memory range to invalidate
- size - Size of the address range to invalidate

Return values:

STAPLER_NO_ERROR
STAPLER_ERROR_BAD_PARAMETER

4.5.1.5 U32 STAPLER_MemoryCopy (void * *destVirt*, void * *srcVirt*, U32 *size*)

Copies data between two memory buffers.

- destVirt - Destination virtual address
- srcVirt - Source virtual address
- size - The size of the data to copy (in bytes)

Returns:

The number of bytes copied

4.5.1.6 void STAPLER_MemoryDeletePartition (STAPLER_PartitionHandle_t *pHandle*, STAPLER_DriverHandle_t *dHandle*)

Deletes a memory partition created with STAPLER_MemoryMakePartition().

- pHandle - The handle of the partition to delete
- dHandle - The handle of the driver creating the partition

See also:

also STAPLER_MemoryMakePartition()

4.5.1.7 STAPLER_PartitionHandle_t STAPLER_MemoryFindPartition (char * *name*)

Locates a partition by name.

Gets a handle for a partition using the name assigned to the partition with STAPLER_MemoryMakePartition()

- name - The name of the partition

Returns:

A handle to the required partition NULL - could not find partition

See also:

STAPLER_MemoryMakePartition()

4.5.1.8 void STAPLER_MemoryFree (STAPLER_PartitionHandle_t *pHandle*, void * *address*, STAPLER_DriverHandle_t *dHandle*, U32 *ref*)

Frees a buffer allocated with STAPLER_MemoryAllocate().

- pHandle - A handle for the memory partition containing the buffer
- address - The address of the buffer to frre
- dHandle - The handle of the driver to which the buffer belongs
- ref - A driver-specific reference number for the buffer

See also:

also STAPLER_MemoryAllocate()

4.5.1.9 STAPLER_PartitionHandle_t STAPLER_MemoryMakePartition (void * *baseAddressVirt*, U32 *size*, char * *name*, STAPLER_DriverHandle_t *handle*)

Creates a new memory partition in virtual memory space.

- baseAddressVirt - The base virtual address of the partition
- size - The size of the partition
- name - The name of the partition
- handle - The handle of the driver creating the partition

Returns:

A handle for the created partition

See also:

also STAPLER_MemoryDeletePartition()
also STAPLER_MemoryFindPartition()

4.5.1.10 void* STAPLER_MemoryMap (U32 *baseAddressPhy*, U32 *size*, char * *name*, STAPLER_DriverHandle_t *handle*)

Maps a physical address range into virtual memory space.

- baseAddressPhy - The base physical address to map
- size - The size of memory to map
- name - An optional name to associate with the mapped range
- handle - The handle of the driver which is requesting the mapping

Returns:

A virtual address space pointer to the mapped buffer, or NULL if the operation failed.

See also:

also STAPLER_MemoryUnmap()

4.5.1.11 void* STAPLER_MemoryUncachedToCached (void * *addressVirt*, U32 *size*)

Creates an cached virtual address mapping for a range of uncahced virtual addresses.

- addressVirt - Base address of the uncached memory range to map
- size - Size of the address range

Returns:

A pointer to the base of the cached memory range

See also:

also STAPLER_MemoryCachedToUncached()

4.5.1.12 void STAPLER_MemoryUnmap (void * *baseAddressVirt*, U32 *size*, STAPLER_DriverHandle_t *handle*)

Unmaps a virtual address range mapped with STAPLER_MemoryMap().

See also:

also STAPLER_MemoryMap()

- baseAddressVirt - The base virtual address to unmap
- size - The size of the virtual address range
- handle - The handle of the driver which is requesting the mapping

4.5.1.13 STAPLER_Error_t STAPLER_MemoryUnmapCached (void * *addressVirt*, U32 *size*)

Unmap a cached virtual address.

- addressVirt - Base address of the cached memory range to map
- size - Size of the address range

Returns:

STAPLER_NO_ERROR

See also:

also STAPLER_MemoryCachedToUncached()
also STAPLER_MemoryUncachedToCached()

4.5.1.14 STAPLER_Error_t STAPLER_MemoryUnmapUncached (void * *addressVirt*, U32 *size*)

Unmap an uncached virtual address.

- addressVirt - Base address of the cached memory range to map
- size - Size of the address range

Returns:

STAPLER_NO_ERROR

See also:

also STAPLER_MemoryCachedToUncached()
also STAPLER_MemoryUncachedToCached()

4.5.1.15 U32 STAPLER_MemoryVirtToPhy (void * *addressVirt*)

Obtains the corresponding physical address from a virtual address.

- addressVirt - The virtual address for which to get the physical address

Returns:

The corresponding physical address

4.6 Interrupt functions

Defines

- #define STAPLER_INTERRUPT_DECLARE(func, context) static STAPLER_Error_t func(void *context)

  Declares an interrupt service routine.

- #define STAPLER_POLL_DECLARE(func, context, timeout) static STAPLER_Error_t func(void *context, STAPLER_Time_t timeout)

  Declares an interrupt polling function.

- #define STAPLER_INTERRUPT_CAST(func) (U32)func

  Cast an interrupt handler function.

- #define STAPLER_POLL_CAST(func) (U32)func

  Cast a polling handler function.

Functions

- STAPLER_Interrupt_Mode_t STAPLER_InterruptMode (void)

  Returns the interrupt mode STAPLER is running in.

- STAPLER_Error_t STAPLER_InterruptMaskSet (U32 *addressVirt, U32 value)

  Sets the interrupt mask
  Remarks:

  If STAPLER is running in polling mode then the mask will not be set.

- STAPLER_Error_t STAPLER_InterruptMake (U32 interruptVector, U32 level, U32 funcISR, U32 funcPoll, void *context, char *name, STAPLER_DriverHandle_t handle)

  Make in interrupt handler.

- STAPLER_Error_t STAPLER_InterruptDelete (U32 interruptVector, U32 level, STAPLER_DriverHandle_t handle)

  Deletes an interrupt handler.

- STAPLER_Error_t STAPLER_InterruptLock (STAPLER_DriverHandle_t handle, U32 ref)

  Disables interrupts.

- STAPLER_Error_t STAPLER_InterruptUnlock (STAPLER_DriverHandle_t handle, U32 ref)

  Enables interrupts.

- STAPLER_Error_t STAPLER_TaskLock (STAPLER_DriverHandle_t handle, U32 ref)

  Disables task switching.

- STAPLER_Error_t STAPLER_TaskUnlock (STAPLER_DriverHandle_t handle, U32 ref)

  Enables task switching.

- STAPLER_Error_t STAPLER_InterruptMap (STAPLER_InterruptNumber_t SecurityInterruptNumber, U32 *pStapiInterruptNumber)

  Maps a pre-defined security driver interrrupt number to a STAPI interrupt number.

4.6.1 Define Documentation

4.6.1.1 #define STAPLER_INTERRUPT_CAST(func) (U32)func

Cast an interrupt handler function.

- func - The interrupt service routine

4.6.1.2 #define STAPLER_INTERRUPT_DECLARE(func, context) static STAPLER_Error_t func(void *context)

Declares an interrupt service routine.

- func - The interrupt service routing
- context - The data the will be passed to the ISR on intterrupt.

4.6.1.3 #define STAPLER_POLL_CAST(func) (U32)func

Cast a polling handler function.

- func - The polling routine

4.6.1.4 #define STAPLER_POLL_DECLARE(func, context, timeout) static STAPLER_Error_t func(void *context, STAPLER_Time_t timeout)

Declares an interrupt polling function.

- func - The interrupt service routing
- context - The data the will be passed ro the Poll code on semaphore wait
- timeout - Maximum number of clock ticks to poll for

4.6.2 Function Documentation

4.6.2.1 STAPLER_Error_t STAPLER_InterruptDelete (U32 *interruptVector*, U32 *level*, STAPLER_DriverHandle_t *handle*)

Deletes an interrupt handler.

- interruptVector - The interrupt vector
- level - The interrupt level
- handle - The driver handle of the driver which created the interrupt handle

Return values:

STAPLER_NO_ERROR
STAPLER_ERROR_INTERRUPT_DISABLE
STAPLER_ERROR_INTERRUPT_UNINSTALL
STAPLER_ERROR_CAN_NOT_FIND_INTERRUPT

See also:

also STAPLER_InterruptMake()

4.6.2.2 STAPLER_Error_t STAPLER_InterruptLock (STAPLER_DriverHandle_t *handle*, U32 *ref*)

Disables interrupts.

- handle - The handle of the driver disabling interrupts
- ref - A driver-specific reference

Return values:

STAPLER_NO_ERROR

See also:

STAPLER_InterruptUnlock()

4.6.2.3 STAPLER_Error_t STAPLER_InterruptMake (U32 *interruptVector*, U32 *level*, U32 *funcISR*, U32 *funcPoll*, void * *context*, char * *name*, STAPLER_DriverHandle_t *handle*)

Make in interrupt handler.

- interruptVector - The interrupt vector
- level - The interrupt level
- funcISR - The ISR function pointer. Cast the isr function using STAPLER_INTERRUPT_CAST
- funcPoll - The polling func pointer. Cast the polling function using STAPLER_POLL_CAST
- context - The context that will be passed to the ISR or polling function on being called
- name - The name of the interrupt (Note: must have a interrupt semaphore with the same name registered)
- handle - The handle of the driver making the interrupt handler

Return values:

STAPLER_NO_ERROR
STAPLER_ERROR_BAD_PARAMETER
STAPLER_ERROR_INTERRUPT_INSTALL
STAPLER_ERROR_NO_FREE_SLOT

See also:

also STAPLER_InterruptDelete

4.6.2.4 STAPLER_Error_t STAPLER_InterruptMap (STAPLER_InterruptNumber_t *SecurityInterruptNumber*, U32 * *pStapiInterruptNumber*)

Maps a pre-defined security driver interrrupt number to a STAPI interrupt number.

The STAPI interrupt number can be used with STAPLER_InterruptMake. This function works-around the issues with STAPI interrupt numbers changing whereas pre-compiled security drivers use fixed interrupt numbers.

Parameters:

← - Security Drivers fixed interrupt number

→ - Pointer set with STAPI interrupt number

Return values:

STAPLER_NO_ERROR

ST_ERROR_BAD_PARAMETER

4.6.2.5 STAPLER_Error_t STAPLER_InterruptMaskSet (U32 * *addressVirt*, U32 *value*)

Sets the interrupt mask

Remarks:

If STAPLER is running in polling mode then the mask will not be set.

- addressVirt - Pointer to the mask register
- value - The mask value

Return values:

STAPLER_NO_ERROR

STAPLER_ERROR_BAD_PARAMETER

4.6.2.6 STAPLER_Interrupt_Mode_t STAPLER_InterruptMode (void)

Returns the interrupt mode STAPLER is running in.

Return values:

STAPLER_INTERRUPT_POLL STAPLER is running in polling mode

STAPLER_INTERRUPT_REAL STAPLER is running in interrupt mode

See also:

also STAPLER_Init()

4.6.2.7 STAPLER_Error_t STAPLER_InterruptUnlock (STAPLER_DriverHandle_t *handle*, U32 *ref*)

Enables interrupts.

- handle - The handle of the driver enabling interrupts
- ref - A driver-specific reference

Return values:

STAPLER_NO_ERROR

See also:

STAPLER_InterruptLock()

4.6.2.8 STAPLER_Error_t STAPLER_TaskLock (STAPLER_DriverHandle_t *handle*, U32 *ref*)

Disables task switching.

- handle - The handle of the driver disabling interrupts
- ref - A driver-specific reference

Return values:

STAPLER_NO_ERROR

See also:

STAPLER_InterruptUnlock()

4.6.2.9 STAPLER_Error_t STAPLER_TaskUnlock (STAPLER_DriverHandle_t *handle*, U32 *ref*)

Enables task switching.

- handle - The handle of the driver enabling interrupts
- ref - A driver-specific reference

Return values:

STAPLER_NO_ERROR

See also:

STAPLER_InterruptLock()

4.7 Semaphore functions

Functions

- STAPLER_Error_t STAPLER_SemaphoreMake (char *name, STAPLER_SemaphoreHandle_t *sHandle, STAPLER_Semaphore_Types_t type, char *interruptName, STAPLER_DriverHandle_t handle)

  Creates a new semaphore.

- STAPLER_Error_t STAPLER_SemaphoreDelete (STAPLER_SemaphoreHandle_t sHandle, STAPLER_DriverHandle_t handle)

  Deletes a semaphore.

- STAPLER_Error_t STAPLER_SemaphoreWait (STAPLER_SemaphoreHandle_t sHandle, U32 timeoutMsec, STAPLER_DriverHandle_t handle, U32 ref)

  Waits for a semaphore to be signalled.

- STAPLER_Error_t STAPLER_SemaphoreSignal (STAPLER_SemaphoreHandle_t sHandle, STAPLER_DriverHandle_t handle, U32 ref)

  Signals a semaphore.

4.7.1 Function Documentation

4.7.1.1 STAPLER_Error_t STAPLER_SemaphoreDelete (STAPLER_SemaphoreHandle_t *sHandle*, STAPLER_DriverHandle_t *handle*)

Deletes a semaphore.

- sHandle - The handle of the semaphore to delete
- handle - The handle of the driver deleting the semaphore

Return values:

STAPLER_NO_ERROR

STAPLER_ERROR_BAD_PARAMETER

STAPLER_ERROR_NO_FREE_SLOT

STAPLER_ERROR_SEMAPHORE_MAKE

See also:

STAPLER_SemaphoreMake()

4.7.1.2 STAPLER_Error_t STAPLER_SemaphoreMake (char * *name*, STAPLER_SemaphoreHandle_t * *sHandle*, STAPLER_Semaphore_Types_t *type*, char * *interruptName*, STAPLER_DriverHandle_t *handle*)

Creates a new semaphore.

- name - A name for the semaphore

- sHandle - Pointer to a semaphore handle to be initialised.
- type - The type of semaphore to create
- handle - The handle of the driver creating the semaphore

Return values:

STAPLER_NO_ERROR STAPLER_ERROR_BAD_PARAMETER STAPLER_ERROR_NO_FREE_SLOT

See also:

STAPLER_SemaphoreDelete()

4.7.1.3 STAPLER_Error_t STAPLER_SemaphoreSignal (STAPLER_SemaphoreHandle_t *sHandle*, STAPLER_DriverHandle_t *handle*, U32 *ref*)

Signals a semaphore.

- sHandle - The handle of the semaphore to signal
- handle - The handle of the driver signalling the semaphore
- ref - A driver-specific reference

Return values:

STAPLER_NO_ERROR

STAPLER_ERROR_BAD_PARAMETER

See also:

also STAPLER_SemaphoreWait()

4.7.1.4 STAPLER_Error_t STAPLER_SemaphoreWait (STAPLER_SemaphoreHandle_t *sHandle*, U32 *timeoutMsec*, STAPLER_DriverHandle_t *handle*, U32 *ref*)

Waits for a semaphore to be signalled.

- sHandle - The handle of the semaphore to wait upon
- timeout - Timeout in milliseconds to wait for the semaphore to be signalled
- handle - The handle of the driver waiting for the semaphore
- ref - A driver-specific reference

Return values:

STAPLER_NO_ERROR

STAPLER_ERROR_NO_INTERRUPT

See also:

also STAPLER_SemaphoreSignal()

4.8 Mutex functions

Functions

- STAPLER_Error_t STAPLER_MutexMake (char *name, STAPLER_MutexHandle_t *mHandle, STAPLER_DriverHandle_t handle)

  Creates a new mutex.

- STAPLER_Error_t STAPLER_MutexDelete (STAPLER_MutexHandle_t mHandle, STAPLER_DriverHandle_t handle)

  Deletes a mutex.

- STAPLER_Error_t STAPLER_MutexLock (STAPLER_MutexHandle_t mHandle, STAPLER_DriverHandle_t handle, U32 ref)

  Waits for a mutex to be locked.

- STAPLER_Error_t STAPLER_MutexTryLock (STAPLER_MutexHandle_t mHandle, STAPLER_DriverHandle_t handle, U32 ref)

  Locks a mutex if it is immediately available, but does not block if it is not available.

- STAPLER_Error_t STAPLER_MutexRelease (STAPLER_MutexHandle_t mHandle, STAPLER_DriverHandle_t handle, U32 ref)

  Releases a mutex.

4.8.1 Function Documentation

4.8.1.1 STAPLER_Error_t STAPLER_MutexDelete (STAPLER_MutexHandle_t *mHandle*, STAPLER_DriverHandle_t *handle*)

Deletes a mutex.

- mHandle - The handle of the mutex to delete
- handle - The handle of the driver deleting the semaphore

Return values:

STAPLER_NO_ERROR

STAPLER_ERROR_BAD_PARAMETER

STAPLER_ERROR_NO_FREE_SLOT

STAPLER_ERROR_MUTEX_MAKE

See also:

STAPLER_MutexMake()

4.8.1.2 STAPLER_Error_t STAPLER_MutexLock (STAPLER_MutexHandle_t *mHandle*, STAPLER_DriverHandle_t *handle*, U32 *ref*)

Waits for a mutex to be locked.

- mHandle - The handle of the mutex to be lock
- handle - The handle of the driver waiting for the semaphore
- ref - A driver-specific reference

Return values:

STAPLER_NO_ERROR
STAPLER_ERROR_NO_INTERRUPT

See also:

also STAPLER_MutexRelease()

4.8.1.3 STAPLER_Error_t STAPLER_MutexMake (char * *name*, STAPLER_MutexHandle_t * *mHandle*, STAPLER_DriverHandle_t *handle*)

Creates a new mutex.

- name - A name for the mutex
- mHandle - Pointer to a mutex handle to be initialised.
- handle - The handle of the driver creating the mutex

Return values:

STAPLER_NO_ERROR
STAPLER_ERROR_BAD_PARAMETER
STAPLER_ERROR_NO_FREE_SLOT

See also:

STAPLER_MutexDelete()

4.8.1.4 STAPLER_Error_t STAPLER_MutexRelease (STAPLER_MutexHandle_t *mHandle*, STAPLER_DriverHandle_t *handle*, U32 *ref*)

Releases a mutex.

- mHandle - The handle of the mutex to release
- handle - The handle of the driver signalling the mutex
- ref - A driver-specific reference

Return values:

STAPLER_NO_ERROR
STAPLER_ERROR_BAD_PARAMETER

See also:

also STAPLER_MutexLock()

4.8.1.5 STAPLER_Error_t STAPLER_MutexTryLock (STAPLER_MutexHandle_t *mHandle*, STAPLER_DriverHandle_t *handle*, U32 *ref*)

Locks a mutex if it is immediately available, but does not block if it is not available.

- mHandle - The handle of the mutex to be lock
- handle - The handle of the driver waiting for the semaphore
- ref - A driver-specific reference

Return values:

STAPLER_NO_ERROR

STAPLER_ERROR_NO_INTERRUPT

See also:

also STAPLER_MutexRelease()

4.9 Timer functions

Functions

- STAPLER_Error_t STAPLER_TimerWait (U32 msecs)
  Waits a number of milliseconds.
- STAPLER_Error_t STAPLER_TimerWaitUs (U32 usWait)
  Waits a number of microseconds.
- STAPLER_Time_t STAPLER_TimerGetTime (void)
  Returns the current time.
- STAPLER_Time_t STAPLER_TimerMinusTime (STAPLER_Time_t a, STAPLER_Time_t b)
  Subtracts two times.
- STAPLER_Time_t STAPLER_TimerPlusTime (STAPLER_Time_t a, STAPLER_Time_t b)
  Adds two times.
- STAPLER_Time_t STAPLER_TimerGetTicksPerSecond (void)
  Returns the number of timer ticks per second.

4.9.1 Function Documentation

4.9.1.1 STAPLER_Time_t STAPLER_TimerGetTicksPerSecond (void)

Returns the number of timer ticks per second.

Returns:

The number of timer ticks per second

4.9.1.2 STAPLER_Time_t STAPLER_TimerGetTime (void)

Returns the current time.

Returns:

The current time

4.9.1.3 STAPLER_Time_t STAPLER_TimerMinusTime (STAPLER_Time_t *a*, STAPLER_Time_t *b*)

Subtracts two times.

Returns a - b, where a and b are time values

- a - Time a (to be subtracted from time b)
- b - Time b

Returns:

(time a) - (time b)

4.9.1.4 STAPLER_Time_t STAPLER_TimerPlusTime (STAPLER_Time_t *a*, STAPLER_Time_t *b*)

Adds two times.

Returns a + b, where a and b are time values

- a - Time a (to be added to time b)
- b - Time b

Returns:

(time a) + (time b)

4.9.1.5 STAPLER_Error_t STAPLER_TimerWait (U32 *msecs*)

Waits a number of milliseconds.

- msec - The number of milliseconds to wait

Return values:

STAPLER_NO_ERROR

4.9.1.6 STAPLER_Error_t STAPLER_TimerWaitUs (U32 *usWait*)

Waits a number of microseconds.

- usWait - The number of microseconds to wait

Return values:

STAPLER_NO_ERROR

4.10 Chip functions

Functions

- U32 STAPLER_ChipGetCutRevision (void)

  Returns the chip Cut.

4.10.1 Function Documentation

4.10.1.1 U32 STAPLER_ChipGetCutRevision (void)

Returns the chip Cut.

Returns:

The cut of chip

4.11 Task Functions

Functions

- STAPLER_Error_t STAPLER_TaskCreate (void(∗Function)(void ∗), void ∗Param, STAPLER_PartitionHandle_t ∗StackPartition, STAPLER_Size_t StackSize, void ∗∗Stack, STAPLER_PartitionHandle_t ∗TaskPartition, STAPLER_Task_t ∗∗Task, STAPLER_TDesc_t ∗Tdesc, int Priority, const char ∗Name, STAPLER_Task_Flags_t Flags, STAPLER_DriverHandle_t Handle)

  STAPLER_TaskCreate() creates a new task.

- STAPLER_Error_t STAPLER_TaskDelete (STAPLER_Task_t ∗Task, STAPLER_PartitionHandle_t ∗TaskPartition, void ∗Stack, STAPLER_PartitionHandle_t ∗StackPartition, STAPLER_DriverHandle_t Handle)

  STAPLER_TaskDelete() removes a previously created task.

- STAPLER_Error_t STAPLER_TaskWait (STAPLER_Task_t ∗∗Task, STAPLER_Time_t ∗TimeOutValue_p, STAPLER_DriverHandle_t Handle)

  STAPLER_TaskWait() causes a context switch and puts the task on the wait queue for the timeout to occur.

4.11.1 Function Documentation

4.11.1.1 STAPLER_Error_t STAPLER_TaskCreate (void(∗)(void ∗) *Function*, void ∗ *Param*, STAPLER_PartitionHandle_t ∗ *StackPartition*, STAPLER_Size_t *StackSize*, void ∗∗ *Stack*, STAPLER_PartitionHandle_t ∗ *TaskPartition*, STAPLER_Task_t ∗∗ *Task*, STAPLER_TDesc_t ∗ *Tdesc*, int *Priority*, const char ∗ *Name*, STAPLER_Task_Flags_t *Flags*, STAPLER_DriverHandle_t *Handle*)

STAPLER_TaskCreate() creates a new task.

Returns:

ST_NO_ERROR if successful else the standard STAPI Errorcode.

4.11.1.2 STAPLER_Error_t STAPLER_TaskDelete (STAPLER_Task_t * *Task*, STAPLER_PartitionHandle_t * *TaskPartition*, void * *Stack*, STAPLER_PartitionHandle_t * *StackPartition*, STAPLER_DriverHandle_t *Handle*)

STAPLER_TaskDelete() removes a previously created task.

Returns:

ST_NO_ERROR if successful else the standard STAPI Errorcode.

4.11.1.3 STAPLER_Error_t STAPLER_TaskWait (STAPLER_Task_t ** *Task*, STAPLER_Time_t * *TimeOutValue_p*, STAPLER_DriverHandle_t *Handle*)

STAPLER_TaskWait() causes a context switch and puts the task on the wait queue for the timeout to occur.

Returns:

ST_NO_ERROR if successful else the standard STAPI Errorcode.

4.12 Stapler queue functions

Functions

- STAPLER_Queue_t STAPLER_QueueCreate (unsigned int ElementSize, unsigned int NoElements)

  STAPLER_QueueCreate creates a message queue.

- STAPLER_Error_t STAPLER_QueueDelete (STAPLER_Queue_t MessageQueue)

  STAPLER_QueueDelete deletes a message queue.

- void * STAPLER_QueueClaimTimeout (STAPLER_Queue_t MessageQueue, U32 ticks)

  STAPLER_QueueClaimTimeout get a free buffer from the message queue.

- void * STAPLER_QueueClaim (STAPLER_Queue_t MessageQueue)

  STAPLER_QueueClaim get a free buffer from the message queue.

- STAPLER_Error_t STAPLER_QueueRelease (STAPLER_Queue_t MessageQueue, void *Message)

  STAPLER_QueueRelease return a buffer to the queue (for a Claim func).

- STAPLER_Error_t STAPLER_QueueSend (STAPLER_Queue_t MessageQueue, void *Message)

  STAPLER_QueueSendTimeout pushs a message on a queue.

- void * STAPLER_QueueReceive (STAPLER_Queue_t MessageQueue)

  STAPLER_QueueReceive pops a message off a queue.

- void * STAPLER_QueueReceiveTimeout (STAPLER_Queue_t MessageQueue, U32 ticks)

  STAPLER_QueueReceiveTimeout pops a message off a queue.

4.12.1 Function Documentation

4.12.1.1 void* STAPLER_QueueClaim (STAPLER_Queue_t *MessageQueue*)

STAPLER_QueueClaim get a free buffer from the message queue.

- MessageQueue - Queue to get message buffer from

Return values:

- a point to the message

NULL - Error

4.12.1.2 void* STAPLER_QueueClaimTimeout (STAPLER_Queue_t *MessageQueue*, U32 *ticks*)

STAPLER_QueueClaimTimeout get a free buffer from the message queue.

- MessageQueue - Queue to get message buffer from
- ticks - Time out

Return values:

- a point to the message

NULL - Error

4.12.1.3 STAPLER_Queue_t STAPLER_QueueCreate (unsigned int *ElementSize*, unsigned int *NoElements*)

STAPLER_QueueCreate creates a message queue.

- ElementSize - The size of each element in the queue
- NoElements - Number of in a queue

Return values:

the message handle

4.12.1.4 STAPLER_Error_t STAPLER_QueueDelete (STAPLER_Queue_t *MessageQueue*)

STAPLER_QueueDelete deletes a message queue.

- MessageQueue - the message queue to delete

Returns:

ST_NO_ERROR if successful else the standard Errorcode.
???????

4.12.1.5 void* STAPLER_QueueReceive (STAPLER_Queue_t *MessageQueue*)

STAPLER_QueueReceive pops a message off a queue.

- MessageQueue - Queue to pop message off

Return values:

- a point to the message

NULL - Error

4.12.1.6 void* STAPLER_QueueReceiveTimeout (STAPLER_Queue_t *MessageQueue*, U32 *ticks*)

STAPLER_QueueReceiveTimeout pops a message off a queue.

- MessageQueue - Queue to pop message off
- ticks - Time out

Return values:

- a point to the message

NULL - Error

4.12.1.7 STAPLER_Error_t STAPLER_QueueRelease (STAPLER_Queue_t *MessageQueue*, void * *Message*)

STAPLER_QueueRelease return a buffer to the queue (for a Claim func).

- MessageQueue - Queue to return the buffer to
- Message - a Pointer to a buffer (that has come from Receive)

Returns:

ST_NO_ERROR if successful else the standard Errorcode.
???????

4.12.1.8 STAPLER_Error_t STAPLER_QueueSend (STAPLER_Queue_t *MessageQueue*, void * *Message*)

STAPLER_QueueSendTimeout pushs a message on a queue.

- MessageQueue - Queue to push message off
- Message - a Pointer to a message to send (must come from a claim call)

Returns:

ST_NO_ERROR if successful else the standard Errorcode.
???????

5 STAPLER Class Documentation

5.1 STAPLER_Init_t Struct Reference

Initialsation paramameters - currently only one!

`#include <stapler.h>`

Public Attributes

- STAPLER_Interrupt_Mode_t interruptMode

5.1.1 Detailed Description

Initialsation paramameters - currently only one!

5.1.2 Member Data Documentation

5.1.2.1 STAPLER_Interrupt_Mode_t STAPLER_Init_t::interruptMode

The interrupt mode that stapler is to run in

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

- stapler.h

Index