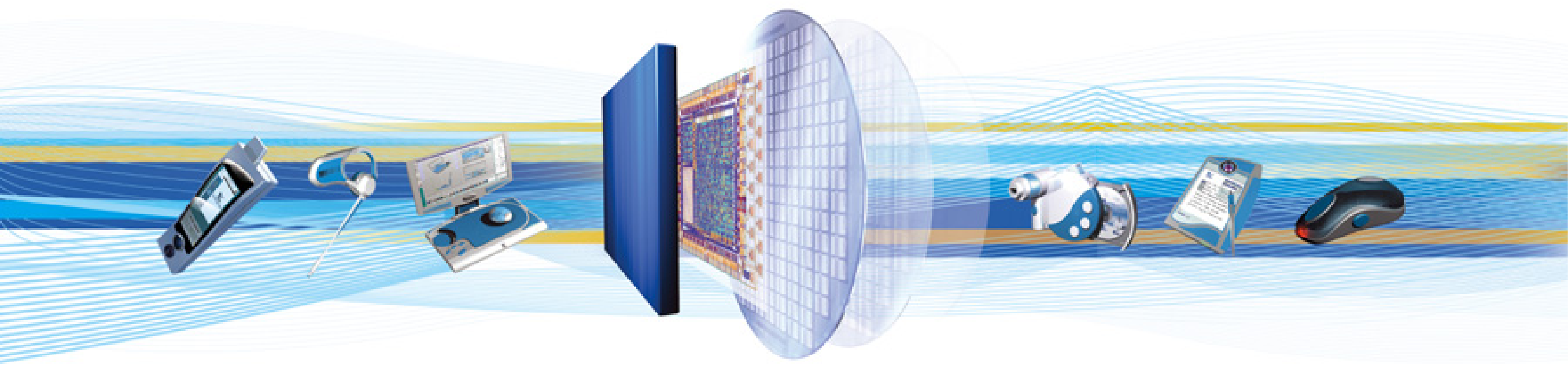
CSR Synergy Framework 3.1.0

Framework Extensions

API Description

August September 2010



Cambridge Silicon Radio Limited

Churchill House
Cambridge Business Park
Cowley Road
Cambridge CB4 0WZ
United Kingdom

Registered in England and Wales 3665875

Tel: +44 (0)1223 692000
Fax: +44 (0)1223 692001
www.csr.com

Contents

1 Introduction

This document describes the functionality and interface provided by the CSR Synergy Framework Extensions API.

The Framework Extensions (FW Ext) is an API that must be used (if the functionality in this document is needed) for porting third party software into a CSR Synergy Product. In addition, some of the driver implementations delivered as a part of the CSR Synergy Framework may depend on these extensions. Driver implementations that require the FW Ext will have this requirement explicitly stated in the driver documentation.

The FW Ext API provides the following services:

- Events
- Mutexes
- Threads
- Dynamic memory management

The following sections provide a detailed description of each of the above services.

2 Events

The event functions are designed to be directly portable to most kernels. The functionality is kept as simple as possible, for example, manual clear is not possible.

The event functions have the following limitations:

- only supports single waiting thread (two threads can not wait for the same event)
- a maximum of 31 event bits can be used
- events are automatically cleared when read
- events stay triggered until read (pulse based events are not supported)
- named events not supported

2.1 CsrEventCreate

Prototype

```
#include "csr_framework_ext.h"

CsrResult CsrEventCreate(CsrEventHandle *eventHandle);
```

Description

Creates an event and returns a handle to the created event.

Parameters

Type	Argument	Description
`CsrEventHandle*`	`eventHandle`	Pointer to store the eventHandle in.

Returns

`CsrResult` with possible values:

`CSR_RESULT_SUCCESS`	in case of success
`CSR_FE_RESULT_NO_MORE_EVENTS`	in case of out of event resources
`CSR_FE_RESULT_INVALID_POINTER`	in case the eventHandle pointer is invalid

2.2 CsrEventWait

Prototype

```
#include "csr_framework_ext.h"

CsrResult CsrEventWait(CsrEventHandle *eventHandle, CsrUint16 timeoutInMs,
CsrUint32 *eventBits);
```

Description

The CsrEventWait function waits for one or more unspecified event bits to be set in the event pointed to by the eventHandle argument. If some event bits were set prior to calling this function, the function returns without waiting for further events. The function returns, in the output parameter eventBits, the event bits set and clears the triggered bits in the event. The timeoutInMs argument specifies an upper limit, in milliseconds, on the time for which CsrEventWait will block. A timeoutinMs value of 0xFFFF means an infinite timeout.

Parameters

Type	Argument	Description
`CsrEventHandle*`	`eventHandle`	Handle to previously created event .
`CsrUint16`	`timeoutInMs`	Timeout value in milliseconds, zero for no wait (return immediately) and 0xFFFF for infinite (will block until at least one event bit is set)
`CsrUint32*`	`eventBits`	Pointer in which to store the returned event bits (zero in case of timeout). **Note:** The eventBits pointer is NOT used an input parameter. I.e. it can not be used for specifying a specific event mask to wait for. The function always waits for one or more unspecified bits.

Returns

`CsrResult` with possible values:

`CSR_RESULT_SUCCESS`	in case of success
`CSR_FE_RESULT_TIMEOUT`	in case of timeout
`CSR_FE_RESULT_INVALID_HANDLE`	in case the eventHandle is invalid
`CSR_FE_RESULT_INVALID_POINTER`	in case the eventBits pointer is invalid

NOTICE: If an invalid handle is passed to this function as argument, the behaviour may be unpredictable on some systems.

ALSO NOTICE: The use this function inside the Framework Scheduler (COAL) is not recommended.

2.3 CsrEventSet

Prototype

```
#include ”csr_framework_ext.h”

CsrResult CsrEventSet(CsrEventHandle *eventHandle, CsrUint32 eventBits);
```

Description

CsrEventSet sets one or more of the event bits for a particular event identified by the eventHandle. The event bits are or’ed together with the current event bits.

Parameters

Type	Argument	Description
`CsrEventHandle*`	`eventHandle`	Handle to previously created event .
`CsrUint16`	`timeoutInMs`	Timeout value in milliseconds, zero for no wait (return immediately) and 0xFFFF for infinite (will block until at least one event bit is set)
`CsrUint32`	`eventBits`	Mask of events to set. Most significant bit is reserved and will be ignored if set

Returns

`CsrResult`	with possible values:	
	`CSR_RESULT_SUCCESS`	in case of success
	`CSR_FE_RESULT_INVALID_HANDLE`	in case the eventHandle is invalid

NOTICE: If an invalid handle is passed to this function as argument, the behaviour may be unpredictable on some systems.

2.4 CsrEventDestroy

Prototype

```
#include ”csr_framework_ext.h”

void CsrEventDestroy(CsrEventHandle *eventHandle);
```

Description

This function destroys the event associated with the handle.

Parameters

Type	Argument	Description
`CsrEventHandle*`	`eventHandle`	Handle to previously created event .

Returns

None - invalid handle silently ignored

NOTICE: If an invalid handle is passed to this function as argument, the behaviour may be unpredictable on some systems.

3 Mutexes

The mutex functions are designed to be directly portable to most kernels. The functionality is kept as simple as possible.

3.1 CsrMutexCreate

Prototype

```
#include ”csr_framework_ext.h”

CsrResult CsrMutexCreate(CsrMutexHandle *mutexHandle);
```

Description

The CsrMutexCreate function creates a mutex and returns a handle to the created mutex.

Parameters

Type	Argument	Description
`CsrMutexHandle*`	`mutexHandle`	Pointer to store the mutexHandle in.

Returns

`CsrResult` with possible values:

`CSR_RESULT_SUCCESS`	in case of success
`CSR_FE_RESULT_NO_MORE_MUTEXES`	in case of out of mutex resources
`CSR_FE_RESULT_INVALID_POINTER`	in case the mutexHandle pointer is invalid

3.2 CsrMutexLock

Prototype

```
#include ”csr_framework_ext.h”

CsrResult CsrMutexLock(CsrMutexHandle *mutexHandle);
```

Description

This function is used for locking the mutex refered to by the provided handle.

Parameters

Type	Argument	Description
`CsrMutexHandle*`	`mutexHandle`	Handle to previously created mutex.

Returns

`CsrResult` with possible values:

`CSR_RESULT_SUCCESS`	in case of success
`CSR_FE_RESULT_INVALID_HANDLE`	in case the mutexHandle is invalid

NOTICE: If an invalid handle is passed to this function as argument, the behaviour may be unpredictable on some systems.

ALSO NOTICE: The use this function inside the Framework Scheduler (COAL) is not recommended, unless the mutex mechanism is only used for atomic variable access (predictable behaviour, very short access time, no blocking calls by other “lockers” while the mutex is locked).

3.3 CsrMutexUnlock

Prototype

```
#include ”csr_framework_ext.h”

CsrResult CsrMutexUnlock(CsrMutexHandle *mutexHandle);
```

Description

This function is used for unlocking a previously locked mutex.

Parameters

Type	Argument	Description
`CsrMutexHandle*`	`mutexHandle`	Handle to previously created mutex.

Returns

`CsrResult` with possible values:

`CSR_RESULT_SUCCESS`	in case of success
`CSR_ERROR_INVALID_HANDLE`	in case the mutexHandle is invalid

NOTICE: If an invalid handle is passed to this function as argument, the behaviour may be unpredictable on some systems.

3.4 CsrMutexDestroy

Prototype

```
#include ”csr_framework_ext.h”

void CsrMutexDestroy(CsrMutexHandle *mutexHandle);
```

Description

This function destroys a previously created mutex.

Parameters

Type	Argument	Description
`CsrMutexHandle*`	`mutexHandle`	Handle to previously created mutex.

Returns

None	- invalid handle silently ignored

NOTICE: If an invalid handle is passed to this function as argument, the behaviour may be unpredictable on some systems.

3.5 CsrGlobalMutexLock

Prototype

```
#include ”csr_framework_ext.h”

void CsrGlobalMutexLock(void);
```

Description

The framework extensions must provide a single pre-created and initialised global mutex that can be used without first doing a CsrMutexCreate() call. This function is used for locking the global mutex. The global mutex must only be held briefly and it is recommended only to use it for protecting the creation of a normal mutex that can then be used for subsequent synchronisation. Depending on the specific platform, the global mutex may be implemented as a spinning lock. This is usually the case if it is not possible to statically create and initialise a mutex or equivalent thread locking mechanism.

Parameters

None

Returns

None

3.6 CsrGlobalMutexUnlock

Prototype

```
#include ”csr_framework_ext.h”

void CsrGlobalMutexUnlock(void);
```

Description

This function is used for unlocking the global mutex.

Parameters

None

Returns

None

4 Threads

The thread functions are very limited in functionality to ensure that they can be implemented on (almost) any kernel. Some of the limitations are:

- No synchronisation functionality – use events if needed
- Only 5 priority levels defined to make mapping to native scheme easier
- No explicit « exit thread » function – just exit from the thread function

4.1 CsrThreadCreate

Prototype

```
#include ”csr_framework_ext.h”

CsrResult CsrThreadCreate(void (*threadFunction)(void *pointer), void *pointer,
CsrUint32 stackSize, CsrUint16 priority, CsrCharString *threadName, CsrThreadHandle
*threadHandle) ;
```

Description

The create thread function returns a handle to the created thread.

Parameters

Type	Argument	Description
`void (*)(void *)`	`threadFunction`	The thread function. The function will be passed the second parameter, pointer, on invocation. The thread is started automatically as soon as the underlying kernel allows it. The thread is terminated when the thread function exits.
`void*`	`pointer`	The parameter passed to the thread function on invocation.
`CsrUint32`	`stackSize`	A hint for the stack size required by the thread specified in bytes. The actual size may be rounded up or down by the implementation. If the stacks size specified is zero or the implementation is unable to provide the requested size, an implementation defined default value is used.
`CsrUint16`	`priority`	The priority that the thread should runs as. 5 priorities are defined : 0 – highest priority 1 – high priority 2 – normal priority 3 – low priority 4 - lowest priority Any values > 4 will be taken as a 4
`CsrCharString*`	`threadName`	A string representing the name of the thread.
`CsrThreadHandle*`	`threadHandle`	Pointer to store the handle to the new thread.

Returns

`CsrResult` with possible values:

`CSR_RESULT_SUCCESS`	in case of success
`CSR_FE_RESULT_NO_MORE_THREADS`	in case of out of thread resources
`CSR_FE_RESULT_INVALID_POINTER`	in case one of the supplied pointers is invalid

4.2 CsrThreadGetHandle

Prototype

```
#include "csr_framework_ext.h"

CsrResult CsrThreadGetHandle(CsrThreadHandle * threadHandle) ;
```

Description

This function returns the handle of the thread calling the function.

Parameters

Type	Argument	Description
`CsrThreadHandle*`	`threadHandle`	Pointer to store the handle to the thread or NULL if the function failed.

Returns

`CsrResult` with possible values:

`CSR_RESULT_SUCCESS`	in case of success
`CSR_FE_RESULT_INVALID_POINTER`	in case the threadHandle pointer is invalid

4.3 CsrThreadEqual

Prototype

```
#include "csr_framework_ext.h"

CsrResult CsrThreadEqual(CsrThreadHandle *threadHandle1, CsrThreadHandle
*threadHandle2) ;
```

Description

This function compares the thread handles.

Parameters

Type	Argument	Description
`CsrThreadHandle*`	`threadHandle1`	Thread handle.
`CsrThreadHandle*`	`threadHandle2`	Thread handle.

Returns

`CsrResult` with possible values:

`CSR_RESULT_SUCCESS`	in case of thread handles are identical
`CSR_FE_RESULT_INVALID_POINTER`	in case either threadHandle pointer is invalid
`CSR_RESULT_FAILURE`	`otherwise`

4.4 CsrThreadSleep

Prototype

```
#include ”csr_framework_ext.h”

void CsrThreadSleep(CsrUint32 sleepTimeInMs);
```

Description

This function will make the calling thread sleep for a given period.

Parameters

Type	Argument	Description
`CsrUint32`	`sleepTimeInMs`	The number of milliseconds the thread should at least sleep. The actual time may be longer.

Returns

None

NOTICE: Only callable in thread context.

5 Dynamic Memory Management

These functions are basically wrappers for the standard C library malloc and free functions.

5.1 CsrMemAlloc

Prototype

```
#include ”csr_framework_ext.h”

void* CsrMemAlloc(size_t size);
```

Description

Allocate dynamic memory of a given size.

Parameters

Type	Argument	Description
`size_t`	`size`	Number of bytes to allocate. A size of zero will return a valid pointer that can be CsrMemFree’d.

Returns

`void*`	Pointer to allocated memory, or NULL in case of failure. The allocated memory is not initialised.

5.2 CsrMemCalloc

Prototype

```
#include ”csr_framework_ext.h”

void* CsrMemCalloc(size_t numberOfElements, size_t elementSize);
```

Description

Allocate dynamic memory of a given size calculated as the numberOfElements times the elementSize. The returned memory is zero initialised.

Parameters

Type	Argument	Description
`size_t`	`numberOfElements`	Number of elements of elementSize to allocate.
`size_t`	`elementSize`	Number of bytes in element.

Returns

`void *`	Pointer to allocated memory, or NULL in case of failure. The allocated memory is not initialised.
NOTICE:	If either/both of the parameters (numberOfElements, elementSize) are zero a valid pointer that can be CsrMemFree’d is returned.

5.3 CsrMemFree

Prototype

```
#include "csr_framework_ext.h"

void CsrMemFree(void *memblock) ;
```

Description

Free dynamic allocated memory.

Parameters

Type	Argument	Description
`void*`	`memblock`	Pointer to previously allocated memory block. If the pointer is a null pointer, no action occurs. Otherwise, if the argument does not match a pointer earlier returned by the CsrMemAlloc, CsrMemAllocDma or CsrMemCalloc functions, or if the space has already been deallocated by a call to CsrMemFree, the behaviour is undefined.

Returns

None.

5.4 CsrMemAllocDma

Prototype

```
#include "csr_framework_ext.h"

void *CsrMemAllocDma(size_t size);
```

Description

Allocate dynamic memory is suitable for DMA transfers.

Parameters

Type	Argument	Description
`size_t`	`size`	Number of bytes to allocate. A size of zero will return a valid pointer that can be CsrMemFreeDma'ed.

Returns

`void *` Pointer to allocated memory, or NULL in case of failure. The allocated memory is not initialised.

5.5 CsrMemFreeDma

Prototype

```
#include "csr_framework_ext.h"

void *CsrMemFreeDma(void *memBlock);
```

Description

Free dynamic memory allocated by CsrMemAllocDma.

Parameters

Type	Argument	Description
`void*`	`memblock`	Pointer to previously allocated memory block. If the pointer is a null pointer, no action occurs. Otherwise, if the argument does not match a pointer earlier returned by the CsrMemAllocDma or if the space has already been deallocated, the behaviour is undefined.

Returns

`None.`

6 Document References

Ref	Title

Terms and Definitions

Abbreviation	Explanation
CSR	Cambridge Silicon Radio

Document History

Revision	Date	History
1	25 Nov 08	First draft version
2	26 Feb 09	Updated according to hj02 comments
3	6 April 09	Updated according to comments from Steering Group
4	14 April 09	Changed a couple of function names and updated a couple of parameters
5	15 April 09	Updated spin lock parameters
6	20 APR 10	Ready for release 2.1.0
7	DEC 10	Ready for release 3.0.0
8	Aug 11	Ready for release 3.1.0

TradeMarks, Patents and Licences

Life Support Policy and Use in Safety-critical Compliance

Performance and Conformance

Refer to www.csrsupport.com for compliance and conformance to standards information.