

eSolid - Real-Time Kernel
1.0BetaR01

Generated by Doxygen 1.8.3.1

Thu Jul 4 2013 04:54:09

Contents

1	eSolid Real-Time Kernel	1
1.1	eSolid RT Kernel Features	1
1.1.1	Source code	1
1.1.2	Consistent Application Programming Interface	1
1.1.3	Preemptive multi-threading	2
1.1.4	Round-Robin scheduling	2
1.1.5	Deterministic	2
1.1.6	Configurable	2
1.1.7	Portable	2
1.1.8	Static object allocation	2
1.1.9	Unlimited number of threads	2
1.1.10	Up to 256 thread priority levels	2
1.1.11	Tickless idle	2
1.1.12	Error checking	3
1.1.13	Profiling	3
2	Directory and file organization	3
2.1	Intro	3
2.1.1	What is a port?	3
2.2	Code Sections	4
2.2.1	Port independent code	4
2.2.2	Port dependent code	4
2.2.3	Template and example code	4
3	Kernel states	5
3.1	Intro	5
3.2	eSolid RT Kernel states	5
4	Thread Management	6
4.1	Intro	6
4.1.1	eSolid RT Kernel thread	6
4.1.2	Thread states	7
4.2	Initializing Threads	8
4.2.1	esThdInit() API function	8
5	Critical sections	8
5.1	Intro	8
5.1.1	eSolid RT Kernel internal critical sections	8
5.2	Implementation	8
5.2.1	Disabling interrupts	9

5.2.2	Disabling Kernel scheduler	9
5.2.3	Using semaphores	10
6	Time complexity	10
6.1	Intro	10
6.1.1	Big O notation	10
6.2	Constant time	10
6.2.1	eSolid RT Kernel time complexity	10
7	Scheduler	10
7.1	Quantum	11
7.2	Threads List	11
7.3	Threads Queue	12
7.3.1	Simple Ready Threads Queue	12
7.3.2	Complex Ready Threads Queue	14
7.4	Ready Threads Queue	14
8	Error checking	14
8.1	Intro	14
9	Module Documentation	14
9.1	Kernel	15
9.1.1	Detailed Description	15
9.2	Interface	16
9.2.1	Detailed Description	20
9.2.2	Macro Definition Documentation	20
9.2.3	Typedef Documentation	21
9.2.4	Enumeration Type Documentation	21
9.2.5	Function Documentation	22
9.2.6	Variable Documentation	36
9.3	Internals	37
9.3.1	Detailed Description	41
9.3.2	Macro Definition Documentation	41
9.3.3	Typedef Documentation	42
9.3.4	Function Documentation	43
9.3.5	Variable Documentation	58
9.4	Configuration	60
9.4.1	Detailed Description	61
9.4.2	Macro Definition Documentation	61
9.5	Port template	63
9.5.1	Detailed Description	63

9.6	Compiler port	64
9.6.1	Detailed Description	65
9.6.2	Macro Definition Documentation	65
9.6.3	Typedef Documentation	66
9.6.4	Enumeration Type Documentation	67
9.7	CPU port interface	68
9.7.1	Detailed Description	71
9.7.2	Macro Definition Documentation	71
9.7.3	Typedef Documentation	75
9.7.4	Function Documentation	75
9.7.5	Variable Documentation	78
9.8	CPU port internals	79
9.8.1	Detailed Description	79
9.8.2	Function Documentation	79
9.8.3	Variable Documentation	80
9.9	CPU port configuration	81
9.9.1	Detailed Description	81
9.9.2	Macro Definition Documentation	81
9.10	Kernel configuration	82
9.10.1	Detailed Description	83
9.10.2	Macro Definition Documentation	83
9.11	Debug	86
9.11.1	Detailed Description	86
9.12	Interface	87
9.12.1	Detailed Description	88
9.12.2	Macro Definition Documentation	88
9.12.3	Enumeration Type Documentation	89
9.12.4	Function Documentation	89
9.13	Internals	91
9.13.1	Detailed Description	91
9.13.2	Function Documentation	91
9.14	Configuration	92
9.14.1	Detailed Description	92
9.14.2	Macro Definition Documentation	92
10	Data Structure Documentation	94
10.1	esKernCtrl Struct Reference	94
10.1.1	Detailed Description	95
10.1.2	Field Documentation	95
10.2	esThd Struct Reference	95

10.2.1 Detailed Description	97
10.2.2 Field Documentation	97
10.3 esThdQ Struct Reference	98
10.3.1 Detailed Description	100
10.3.2 Field Documentation	100
10.4 esVTmr Struct Reference	100
10.4.1 Detailed Description	101
10.4.2 Field Documentation	102
10.5 portCtx Struct Reference	102
10.5.1 Detailed Description	102
10.5.2 Field Documentation	103
10.6 portStck Struct Reference	103
10.6.1 Detailed Description	103
10.6.2 Field Documentation	103
10.7 esThdQ::prioBM Struct Reference	103
10.7.1 Detailed Description	104
10.7.2 Field Documentation	104
10.8 sysTmr Struct Reference	104
10.8.1 Detailed Description	105
10.8.2 Field Documentation	105
10.9 esThd::thdL Struct Reference	106
10.9.1 Detailed Description	107
10.9.2 Field Documentation	107
10.10esThdQ::thdLSentinel Struct Reference	107
10.10.1 Detailed Description	109
10.10.2 Field Documentation	109
10.11esVTmr::tmrL Struct Reference	109
10.11.1 Detailed Description	110
10.11.2 Field Documentation	110
11 File Documentation	110
11.1 compiler.h File Reference	110
11.1.1 Detailed Description	113
11.2 cpu.c File Reference	113
11.2.1 Detailed Description	114
11.3 cpu.h File Reference	114
11.3.1 Detailed Description	117
11.4 cpu_cfg.h File Reference	118
11.4.1 Detailed Description	118
11.5 dbg.c File Reference	118

11.5.1 Detailed Description	119
11.6 dbg.dox File Reference	119
11.7 dbg.h File Reference	119
11.7.1 Detailed Description	121
11.8 dbg_cfg.h File Reference	121
11.8.1 Detailed Description	122
11.9 kernel-example.dox File Reference	122
11.10 kernel.c File Reference	122
11.10.1 Detailed Description	127
11.11 kernel.dox File Reference	127
11.12 kernel.h File Reference	127
11.12.1 Detailed Description	132
11.13 kernel_cfg.h File Reference	132
11.13.1 Detailed Description	133
11.14 kernel_cfg.h File Reference	133
11.14.1 Detailed Description	134

Index

134

1 eSolid Real-Time Kernel

1.1 eSolid RT Kernel Features

1.1.1 Source code

The source code of the kernel and all of its ports are published under **free software license**, which guarantees end users (individuals, organizations, companies) the freedoms to use, study, share (copy), and modify the software.

The GPL grants the recipients of a computer software the rights of the Free Software Definition (written by Richard Stallman) and uses copyleft to ensure the freedoms are preserved whenever the work is distributed, even when the work is changed or added to. The GPL is a copyleft license, which means that derived works can only be distributed under the same license terms.

For more details visit: <https://gnu.org/licenses/gpl.html>

1.1.2 Consistent Application Programming Interface

All objects declared in Application Programming Interface are following these naming rules:

- All objects except macros are using `CamelCase` style names
- All functions, structures and unions are prefixed with: `es`
- All typedef-ed types are prefixed with: `es` and postfixed with: `_T`
- All macro names are in `UPPERCASE` style, words are delimited by underscore sign
- All macro names are prefixed with: `ES_`
- All Global variables are prefixed with: `g`

All API objects are named following this convention: `es<group><action><suffix>()`

- Group:
 - `Kern` - General Kernel services
 - `Thd` - Thread management
 - `ThdQ` - Thread Queue management
 - `Sched` - Scheduler invocation
 - `SchedRdy` - Scheduler Ready Thread Queue management
- Suffix:
 - `none` - normal API object
 - `I` - I class - Regular Interrupts are locked

All Port Interface objects are named using the rules stated above with certain differences:

- All functions, structures and unions are prefixed with: `port`
- All macro names are prefixed with: `PORT_`

1.1.3 Preemptive multi-threading

eSolid RT Kernel uses a **preemptive scheduler**, which has the power to preempt, or interrupt, and later resume, other threads in the system. The scheduler always runs ready thread with the highest priority.

1.1.4 Round-Robin scheduling

Round-Robin scheduling is very **simple algorithm** to implement and it is **starvation free**. It employs time-sharing, giving to each thread a time slice or `quantum`. Processor's time is shared between a number of threads, giving the illusion that it is dealing with these threads **concurrently**. This scheduling is only used when there are two or more threads of the same priority ready for execution.

1.1.5 Deterministic

Majority of algorithms used in eSolid RT Kernel implementation are belonging to **Constant Time Complexity** category. Constant Time $O(1)$ functions needs fixed amount of time to execute an algorithm. In other words the execution time of Constant Time Complexity functions does not depend on number of inputs. For more information see [Time complexity](#).

1.1.6 Configurable

The kernel provides two configuration files `kernel_cfg.h` and [cpu_cfg.h](#) which can be used to tailor the kernel to application needs.

In addition, the kernel implements a number of hooks which can alter or augment the behavior of the kernel or applications, by intercepting function calls between software components.

1.1.7 Portable

During the design stage of the kernel a special attention was given to achieve high portability of the kernel. Some data structures and algorithms are tailored to exploit new hardware features.

1.1.8 Static object allocation

All objects used in eSolid RT Kernel can be statically allocated. There is no need to use any memory management functionality which makes it very easy to verify the application.

1.1.9 Unlimited number of threads

eSolid RT Kernel allows applications to have any number of threads. The only limiting factors for the maximum number of threads are the amount of RAM and ROM memory capacity and required processing time.

1.1.10 Up to 256 thread priority levels

Each thread has a defined priority. Lowest priority level is 0, while the highest available level is configurable. If Round-Robin scheduling is used then multiple threads can be in the same priority level. If Round-Robin scheduling is disabled then each thread must have unique priority level. The priority sorting algorithm has constant time complexity which means it always executes in the same time period regardless of the levels of priority used.

1.1.11 Tickless idle

Classic kernel architectures periodically interrupted CPU at a predetermined frequency — 100 Hz, 250 Hz, or 1000 Hz, depending on the application needs. Known as the system timer tick, the kernel performed this interrupt regardless of the power state of the CPU. Therefore, even an idle CPU was responding to up to 1000 of these requests every second. On systems that implemented power saving measures for idle CPUs, the timer tick prevented the CPU from remaining idle long enough for the system to benefit from these power savings.

The eSolid RT kernel runs in tickless idle: that is, it replaces the old periodic timer interrupts with on-demand interrupts. Therefore, idle CPUs are allowed to remain idle until a new task is queued for processing, and CPUs that have entered lower power states can remain in these states longer.

1.1.12 Error checking

All eSolid software is using design methods very similar to approaches of **contract programming** paradigm for software design. The contract programming prescribes that Application Programming Interface should have formal, precise and verifiable specifications, which extend the ordinary definition of abstract data types with preconditions and postconditions. These specifications are referred to as "contracts". The contract for each method will normally contain the following pieces of information:

- Acceptable and unacceptable input values
- Return values and their meanings
- Error and exception condition values that can occur during the execution
- Side effects
- Preconditions
- Postconditions
- Invariants

The contract validations are done by **assert** macros. They have the responsibility of informing the programmer when a contract can not be validated.

1.1.13 Profiling

Note

This feature is not implemented

2 Directory and file organization

Details about directory and file organization

2.1 Intro

The directory structure of eSolid RT Kernel is fairly easy to understand. Once the organization of directories and files is understood it is fairly easy to integrate eSolid RT Kernel into application.

2.1.1 What is a port?

Porting is a process of adapting software to an architecture that is different from the one for which it was originally designed. The term is also used when software is changed to make it usable in different environments. Software is portable when the cost of porting it to a new platform is less than the cost of writing it from beginning.

2.2 Code Sections

The kernel is divided into three sections. One section is port independent code, the second one is port dependent code and the third sections is code templates.

2.2.1 Port independent code

Port independent code is the code which does not change from port to port, e.g. when the CPU is changed this code is not changed at all and it is still correctly executed. Code can be developed and tested on another machine, which greatly reduces design efforts. It provides API and some common data structures. Port independent code lives under `/inc` and `/src` directories:

- [inc/kernel.h](#)
- [inc/kernel_cfg.h](#)
- [src/kernel.c](#)
- [inc/dbg.h](#)
- [inc/dbg_cfg.h](#)
- [src/dbg.c](#)

Click on file name for further description of the file.

2.2.2 Port dependent code

Second section is the port dependent code. This code provides low-level functions which are needed to interact with interrupt controllers, manipulate CPU settings and do the context switching. They are highly CPU/compiler bounded and are often written in assembly language.

Each port has its name which is also the name of directory which holds all the port files. Usually each port has some kind of variant. In that case each variant is a subdirectory of the containing port. Common code for all variants will be in common subdirectory. Each eSolid RT Kernel port will have at least the following files:

- `port/[port_name]/common/compiler.h`
- `port/[port_name]/[variant_name]/cpu_cfg.h`
- `port/[port_name]/[variant_name]/cpu.h`
- `port/[port_name]/[variant_name]/cpu.c`

Note

Port dependent code is separately described in documentation for relevant port.

2.2.3 Template and example code

Templates are some predefined configuration settings for various scenarios where eSolid RT Kernel can be used. Templates also contain some example code for how to write new ports.

In the example below is given `Generic` template which holds files with default configuration settings and some example code for new ports. New port files are in `template/generic/port` directory. When porting to a new architecture/compiler use provided template files for starters. This will greatly reduce the time needed to become familiar with the kernel port requirements. Generic template files are the following:

- `template/generic/port/compiler.h`
- `template/generic/port/cpu_cfg.h`
- `template/generic/port/cpu.h`
- `template/generic/port/cpu.c`
- `template/generic/kernel_cfg.h`
- `template/generic/dbg_cfg.h`

3 Kernel states

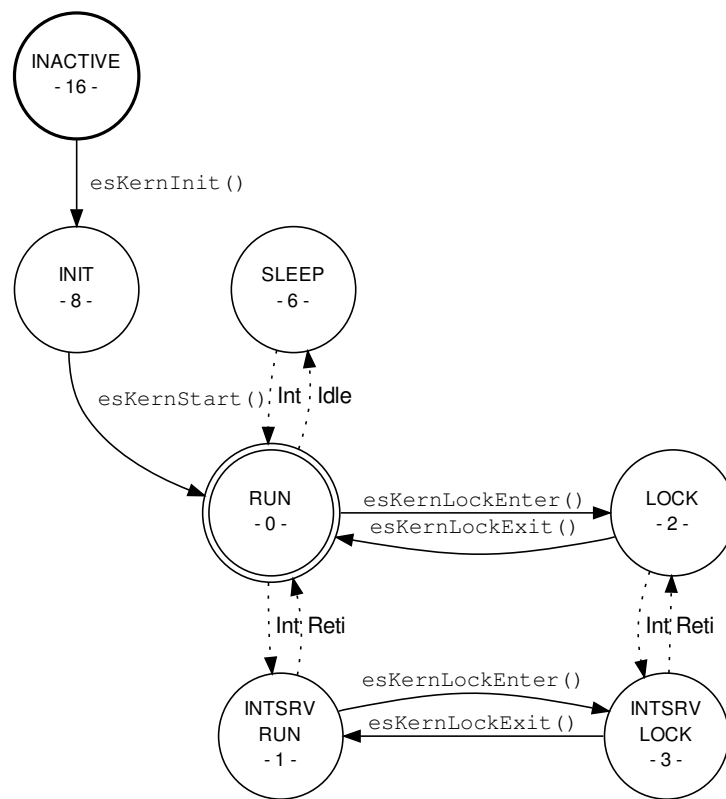
Details about kernel states

3.1 Intro

A Kernel state machine is a behavior model of the kernel core. Each state defines what methods are allowed.

3.2 eSolid RT Kernel states

The kernel can be in one of the following states:



INACTIVE

Inactive state of the kernel (Level 5). This state is entered after a physical reset. When the system is in this state all the maskable interrupt sources are disabled. In this state none of kernel internal data structures are initialized. In this state it is not possible to use any Kernel API except [esKernInit\(\)](#).

INIT

Initialization state of the kernel (Level 4). In this state all internal data structures are initialized but the kernel is still not running. In this stage new threads can be created by calling [esThdlInit\(\)](#) function. Also, the application is allowed to use API which is used to create kernel structures like Thread Queues [esThdQ](#). All the maskable interrupt sources are DISABLED.

RUN

Normal, running state of the kernel (Level 0). To start multi-threading just call the [esKernStart\(\)](#) function. This function will switch the kernel into RUN state and multi-threading of created threads will commence. During the RUN state you are allowed to create other task as well. All the interrupt sources are enabled and the system APIs are accessible, threads are running. All the maskable interrupt sources are ENABLED.

LOCK

Scheduler locked state, no context switching (Level 2). The running state of the kernel can be switched to LOCK state where the scheduler is locked and no context switching is allowed. LOCK state is one way of preventing the access to a shared resource. One more reason to lock the scheduler would be during the accessing of special hardware (e.g. programming the FLASH memory) which does not allow interruption of the running operation. Usage of scheduler locks should be kept at minimum. All the maskable interrupt sources are ENABLED.

INTSRV_RUN or INTSRV_LOCK

Interrupt Service state, no context switching (Levels 1 and 3). During the both states RUN and LOCK, an interrupt event can occur. When Interrupt Service Routine is executing the kernel is in INTSRV_RUN or I-

NTSRV_LOCK state. Each state corresponds to the state where the execution was interrupted from and the kernel will return to it's original state.

SLEEP

When idle condition occurs the kernel will switch to SLEEP state (if power saving is enabled). In order to return to RUN state an interrupt must occur whether from system timer or any other interrupt source which must request a context switch upon exit from ISR.

Note

The level of state INACTIVE is the highest. As the kernel boots up the level is decremented. The running state is level 0.

4 Thread Management

Introduction to threads and how to use them

4.1 Intro

A thread, also called a thread of execution is the smallest sequence of program instructions that can be managed by an operating system scheduler. Multi-threading is implemented by time-division multiplexing where the processor switches between threads. Context switching occurs fast enough that the user perceives the threads as running at the same time. By using threads a programmer can split the work into the threads, each responsible for a smaller portion of the problem. From a threads view he thinks it has the processor all to itself.

4.1.1 eSolid RT Kernel thread

eSolid RT Kernel supports multi-threading and allows applications to have any number of threads. The only limiting factors for the maximum number of threads are the amount of RAM and ROM memory and processing time.

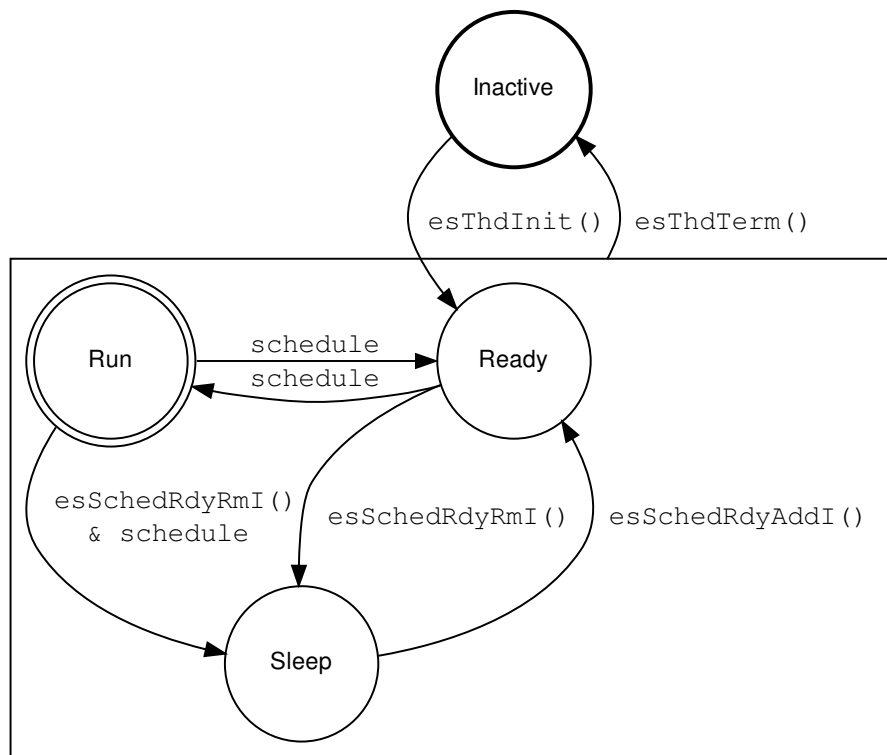
Threads are implemented as normal C functions. Thread functions must have the following prototype:

```
void fn (void *);
```

Which in plain english means: *fn is a function (pointer to void) returning void.*

4.1.2 Thread states

A thread can be in one of the following states:



Inactive

This is thread initial state. Threads in this state are still not activated (**inactive**) by `esThdInit()` function or they were deleted by `esThdTerm()` function. The scheduler does not recognize these threads and they will never execute.

Ready

Threads waiting to execute. There are the threads that are **ready** to execute but are not currently executing because a different thread (equal or higher priority) is already executing.

Run

Thread is currently executing. When the thread is in this state then the code is actually being **run** on the processor.

Sleep

Thread is sleeping. These threads are **sleeping** while waiting for an event to occur.

4.2 Initializing Threads

4.2.1 esThdInit() API function

Threads are initialized by using `esThdInit()` API function.

Stack size

There is no easy way to determine the stack size required by a thread. It is possible to calculate approximate stack size for simple threads, but for more complex ones (e.g. which calls library API function) this can be a daunting task. In this case stack size will be set to a size more than adequate for the thread and then use the profiling features provided by the kernel to ensure both that the space allocated is adequate, and that RAM space is not being unnecessarily wasted.

5 Critical sections

How to deal with critical sections in an application

5.1 Intro

In concurrent programming, a critical section is a piece of code that accesses a shared resource (data structure or device) that must not be concurrently accessed by more than one thread of execution. A critical section will usually terminate in fixed time, and a thread will have to wait for a fixed time to enter it (aka bounded waiting). Some synchronization mechanism is required at the entry and exit of the critical section to ensure exclusive use, for example a semaphore.

5.1.1 eSolid RT Kernel internal critical sections

In contrast to application code in kernel code there is no other mechanism to protect critical code except disabling interrupts. Fortunately, some ports have ability to mask certain interrupts with low priority and allow interrupts with higher priority. By masking low priority interrupts the kernel can protect its critical sections. However for this scheme to work it's forbidden to call any OS service function from a high priority interrupt. If this rule is not followed then the high priority interrupt with an OS service function call can preempt the kernel low priority interrupt which will in that case corrupt the kernel internal data structures.

Note

- 1) It is forbidden to call any OS service function from an interrupt with the priority higher than the kernel interrupt priority.
- 2) On some ports the kernel never completely disables interrupts.

5.2 Implementation

There are multiple ways how are critical sections implemented:

- The simplest method is to prevent interrupts on entry into the critical section, and restoring interrupts to their previous state on exit from critical section. Any thread of execution entering any critical section anywhere in the system, with this implementation, will prevent any other thread, including an interrupt, from being executed on the CPU.
- This approach can be improved upon by using semaphores. To enter a critical section, a thread must obtain a semaphore, which it releases on leaving the section. Other threads are prevented from entering the critical section at the same time as the original thread, but are free to gain control of the CPU and execute other code, including other critical sections that are protected by different semaphores.

5.2.1 Disabling interrupts

In order to properly disable interrupts the application must follow these steps:

- declare an `auto` variable which will hold interrupt state
- save interrupt status into `auto` variable and disable interrupts

- access the shared resource
- restore previously saved interrupt state

For `auto` variable declaration macro `ES_CRITICAL_DECL()` is used. This macro will declare a temporary interrupt status variable. Then by using the macro `ES_CRITICAL_ENTER()` the state of enabled interrupts will be saved in `auto` variable declared earlier. Immediately after saving the interrupt state the macro will lock interrupts. Now the code can safely access and use the shared resource. When code finishes using the resource it will call `ES_CRITICAL_EXIT()` macro. This macro will restore interrupts from the previously saved interrupt state.

```
ES_CRITICAL_DECL();           /* Declare an interrupt status variable */
:
:
:
ES_CRITICAL_ENTER();          /* Save state and lock interrupts */
/* Access the shared resource
*/
ES_CRITICAL_EXIT();           /* Restore previous state unlocking the interrupts */
```

When to use this scheme

- If interrupt service routine *changes* the shared resource state.
- If the processing time of critical section is very small.

When not to use this scheme

- If interrupt service routine takes a lot of CPU time to process critical section. If a critical section is long, then the system clock will drift every time a critical section is executed because the system timer interrupt is no longer serviced, so tracking time is impossible during the critical section. Also, if a program execution halts during its critical section, control will never be returned to another thread, effectively halting the entire system.

5.2.2 Disabling Kernel scheduler

Another way to implement a critical section and protect your data is by locking the kernel scheduler. The kernel locking can be used only if you know that protected data will be modified only by other threads. This protection scheme can not be used when data is modified by interrupt service routines.

```
esKernLockEnter();           /* Temporarily disable kernel scheduler */
/* Access the shared resource
*/
esKernLockExit();            /* Enable kernel scheduler */
```

When to use this scheme

- If interrupt service routine *never changes* the shared resource state.
- If the processing time of critical section is very small.

When not to use this scheme

- If interrupt service routine takes a lot of CPU time to process critical section. If a critical section is long, then the system will be partially responsive to other events since interrupt service routines can be invoked, but note that any further processing by other threads is still disabled.

5.2.3 Using semaphores

6 Time complexity

About time categories of algorithms

6.1 Intro

In computer science, the time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input. The time complexity of an algorithm is commonly expressed using **big O** notation, which excludes coefficients and lower order terms. When expressed this way, the time complexity is said to be described asymptotically, i.e., as the input size goes to infinity. For example, if the time required by an algorithm on all inputs of size n is at most $5n^3 + 3n$, the asymptotic time complexity is $O(n^3)$.

Time complexity is commonly estimated by counting the number of elementary operations performed by the algorithm, where an elementary operation takes a fixed amount of time to perform. Thus the amount of time taken and the number of elementary operations performed by the algorithm differ by at most a constant factor.

Since an algorithm's performance time may vary with different inputs of the same size, one commonly uses the worst-case time complexity of an algorithm, denoted as $T(n)$, which is defined as the maximum amount of time taken on any input of size n . Time complexities are classified by the nature of the function $T(n)$. For instance, an algorithm with $T(n) = O(n)$ is called a linear time algorithm, and an algorithm with $T(n) = O(2^n)$ is said to be an exponential time algorithm.

Note

Worst-case time-complexity $T(n)$ indicates the longest running time performed by an algorithm given any input of size n , and thus this guarantees that the algorithm finishes on time.

6.1.1 Big O notation

Big O notation describes the limiting behavior of a function when the argument tends towards a particular value or infinity, usually in terms of simpler functions and it is used to classify algorithms by how they respond (e.g., in their processing time or working space requirements) to changes in input size.

6.2 Constant time

An algorithm is said to be constant time (also written as $O(1)$ time) if the value of $T(n)$ is bounded by a value that does not depend on the size of the input.

Despite the name *constant time*, the running time does not have to be independent of the problem size, but an upper bound for the running time has to be bounded independently of the problem size.

Note

Constant time effectively means that there is a constant upper bound to how long the function will take to run which isn't affected by any of the input argument.

6.2.1 eSolid RT Kernel time complexity

All eSolid RT Kernel functions are using `constant time` $O(1)$ algorithms. This is especially important for Real Time applications.

7 Scheduler

About the scheduler and Ready Threads Queue

7.1 Quantum

The period of time for which a thread is allowed to execute in a preemptive multi-threading system is generally called the time slice, or `quantum`. The scheduler is run once every quantum to choose the next thread for execution. If the quantum is too short then the scheduler overhead may become high.

An interrupt is used to allow the kernel to switch between threads when their quantum expires, effectively allowing the processor's time to be shared between a number of threads, giving the illusion that it is dealing with these threads concurrently.

7.2 Threads List

Each thread structure `esThd` contains Thread List structure `esThd::thdL`. All threads of the same priority are linked together via `next` and `prev` members in `esThd::thdL` structure. The first member of the structure is pointer `q` which points back to the Threads Queue structure (`esThdQ`) which contains the threads.

The list is organized as **circular doubly linked list**, which means that `tail` and `head` nodes are linked together just like every other node in the list. This provides easy and efficient traversal of the list.

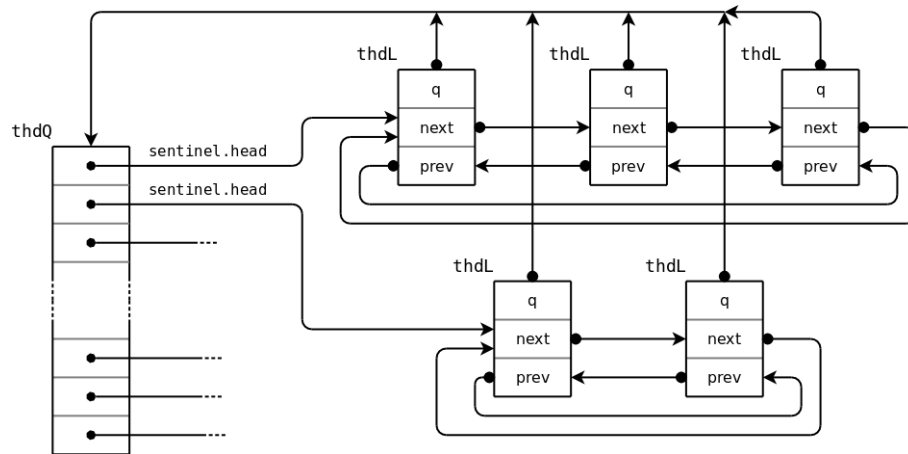


Figure 1: Detailed view of Threads List (sentinel.next pointers not shown)

Each sentinel of a list has two pointers, `head` and `next`. Pointer `sentinel.head` always points to the first entry of the list which is called `head`. Every new thread is added at the `tail` of the list which is essentially just after the node `head`. When a first thread is added to the list the pointer `sentinel.next` points to the thread, too. When the list is rotated using function `esThdQFetchRotatel()` the pointer `sentinel.next` is advanced forward and points to the next thread in list.

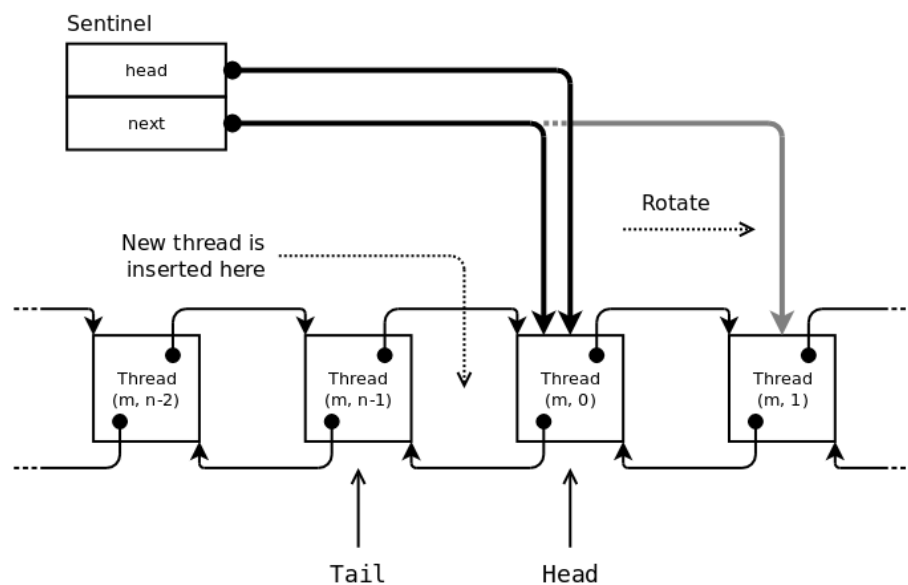


Figure 2: Detailed view of the sentinel and linked list

7.3 Threads Queue

Based on the number of configured priority levels (see [CFG_SCHED_PRIO_LVL](#)) and on the number of data register bits (see [PORT_DATA_WIDTH_VAL](#)) of the used CPU, two configurations are possible:

- Simple Ready Threads Queue
- Complex Ready Threads Queue

Simple Ready Threads Queue configuration is used when the number of configured priority levels is lower or equal to the number of bits in general purpose data register. For example if application is using 9 priority levels on 32-bit CPU than simple Ready Threads Queue configuration is used. In contrast, when using 9 priority levels on an 8-bit CPU than the kernel is forced to use the Complex Ready Threads Queue configuration since 8-bit register cannot carry 9 bits of data.

7.3.1 Simple Ready Threads Queue

Each bit in `bit[0]` variable represents one priority level. The number of bits used in this variable depends on [CFG_SCHED_PRIO_LVL](#) value. If a bit at N^{th} position is set then there is a thread inserted in Thread List at N^{th} priority level.

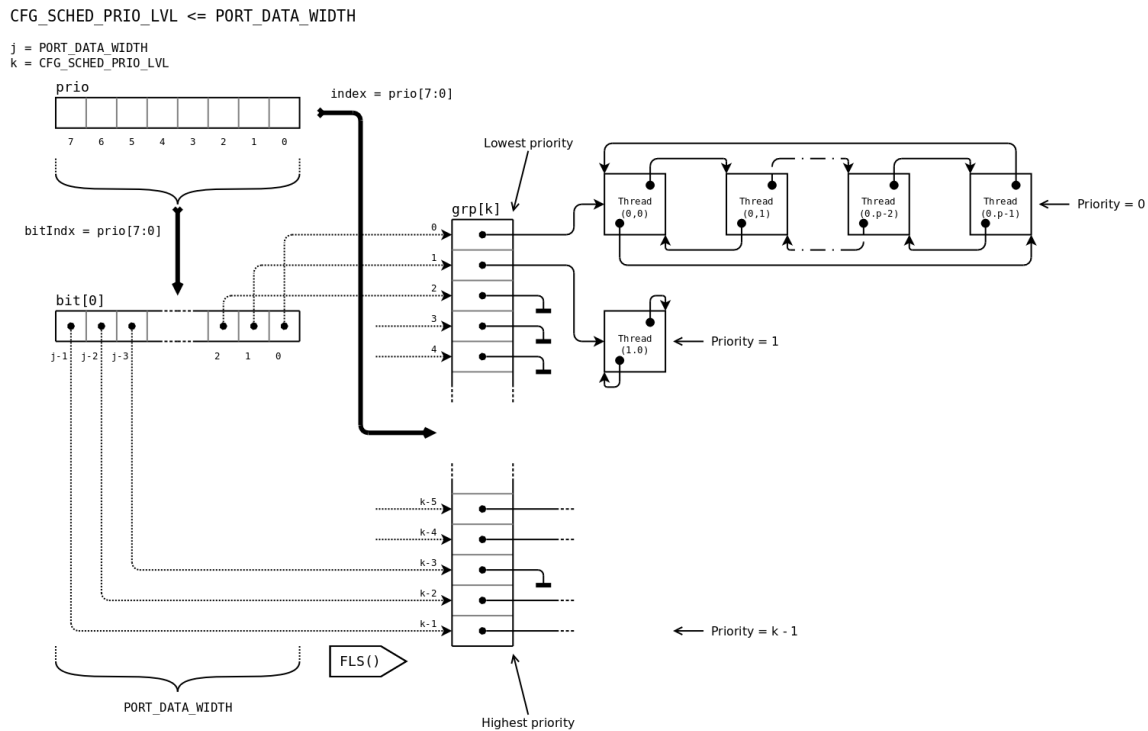


Figure 3: Ready Threads Queue - low number of priority levels

Inserting a thread

The process of a thread insertion into a thread queue can be described using the following pseudo-code:

```
function insert(thread)
    priority := thread.priority

    if (grp[priority].head == NULL)
        grp[priority].head := thread
        grp[priority].next := thread
        bitIndx := 2^priority
        priority
        bit[0] := bit[0] or bitIndx
    else
        # Get the priority of the thread
        # If this priority level has a list
        # Create a list with this thread as head
        # bitIndx equals to 2 raised to the power of
        # Set the calculated bit in Bit Map
```

```

        listInsertAtTail(grp[priority].head, thread)      # Add thread at tail of existing list
    end if
end function

```

Removing a thread

The process of a thread removal can be described with the following pseudo-code:

```

function remove(thread)
    priority := thread.priority                          # Get the priority of the thread

    if (listIsEntryLast(thread))                        # In case we are removing the last entry
        grp[priority].head := NULL                    # List is deleted
        bitIndx := 2^priority                          # bitIndx equals to 2 raised to the power of
        priority                                     # Clear the calculated bit in Bit Map
        bit[0] := bit[0] and not bitIndx
    else
        listRemove(thread)                             # Remove the thread from list
    end if
end function

```

Fetching the highest priority thread

The process of fetching the highest priority thread is inverse function of 2^{priority} which was used in `insert()` function:

```

function fetch()
    priority := log2(bit[0])                            # Find Last Set bit position in bit[0]

    return grp[priority]
end function

```

Rotating the threads queue

The process can be described with the following algorithm:

```

function rotate()
    priority := log2(bit[0])                            # Find Last Set bit position in bit[0]

    grp[priority].next := grp[priority].next.next

    return grp[priority].next
end function

```

7.3.2 Complex Ready Threads Queue

CFG_SCHED_PRIO_LVL > PORT_DATA_WIDTH

```
i = PRIO_BM_GRP_INDX = round_up(CFG_SCHED_PRIO_LVL / PORT_DATA_WIDTH)
j = PORT_DATA_WIDTH
k = CFG_SCHED_PRIO_LVL
l = PRIO_BM_DATA_WIDTH_LOG2 = log2(PORT_DATA_WIDTH)
```

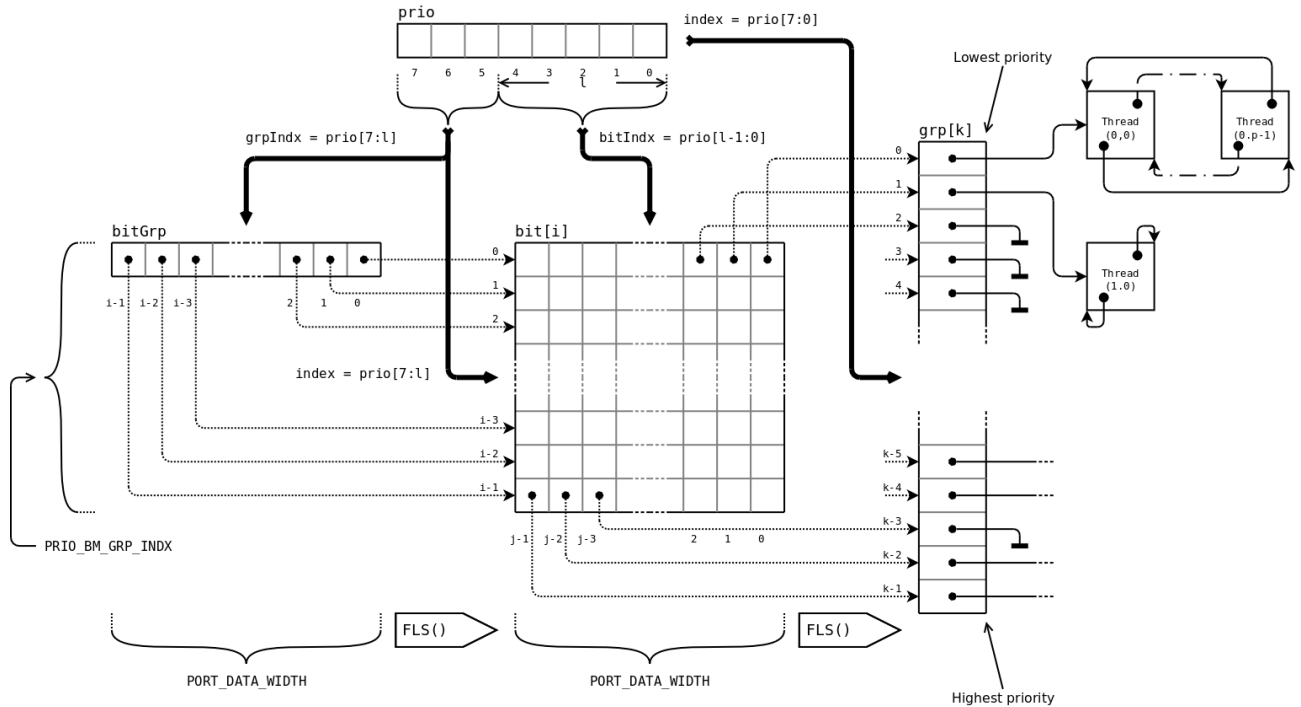


Figure 4: Ready Threads Queue - high number of priority levels

7.4 Ready Threads Queue

Ready Threads Queue holds threads that are ready for execution.

8 Error checking

How errors are detected

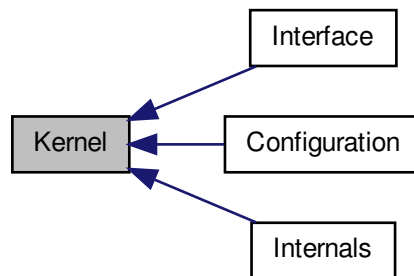
8.1 Intro

9 Module Documentation

9.1 Kernel

Overview.

Collaboration diagram for Kernel:



Modules

- [Configuration](#)
Configuration settings.
- [Interface](#)
Application programming interface.
- [Internals](#)
Kernel inner work.

9.1.1 Detailed Description

Overview.

9.2 Interface

Application programming interface.

Collaboration diagram for Interface:



Data Structures

- struct [esThd](#)
Thread structure.
- struct [esVTmr](#)
Virtual Timer structure.
- struct [esThdQ](#)
Thread Queue structure.
- struct [esKernCtrl](#)
Kernel control block structure.

Kernel identification and version number

- #define [ES_KERN_VER](#) 0x10000UL
Identifies the underlying kernel version number.
- #define [ES_KERN_ID](#) "eSolid Kernel v1.0"
Kernel identification string.

Critical section management

These macros are used to prevent interrupts on entry into the critical section, and restoring interrupts to their previous state on exit from critical section.

For more details see [Critical sections](#).

- #define [ES_CRITICAL_DECL\(\)](#) [PORT_CRITICAL_DECL\(\)](#)
Critical section status variable declaration.
- #define [ES_CRITICAL_ENTER\(\)](#) [PORT_CRITICAL_ENTER\(\)](#)
Enter a critical section.
- #define [ES_CRITICAL_EXIT\(\)](#) [PORT_CRITICAL_EXIT\(\)](#)
Exit from critical section.
- #define [ES_CRITICAL_ENTER_LOCK_EXIT\(\)](#)
Enter critical section and exit scheduler lock.
- #define [ES_CRITICAL_EXIT_LOCK_ENTER\(\)](#)
Exit critical section and enter scheduler lock.

Thread management

Basic thread management services

For more details see [Thread Management](#).

- typedef struct [esThd](#) [esThd_T](#)
Thread type.
- typedef [portStck_T](#) [esStck_T](#)
Stack type.
- void [esThdInit](#) ([esThd_T](#) *thd, void(*fn)(void *), void *arg, [portStck_T](#) *stck, size_t stckSize, uint8_t prio)
Initialize the specified thread.
- void [esThdTerm](#) ([esThd_T](#) *thd)
Terminate the specified thread.
- static [PORT_C_INLINE](#) [esThd_T](#) * [esThdGetId](#) (void)
Get the current thread ID.
- static [PORT_C_INLINE](#) uint8_t [esThdGetPrio](#) ([esThd_T](#) *thd)
Get the priority of a thread.
- void [esThdSetPrio](#) ([esThd_T](#) *thd, uint8_t prio)
Set the priority of a thread.
- void [esThdPostl](#) ([esThd_T](#) *thd)
Post to thread semaphore.
- void [esThdPost](#) ([esThd_T](#) *thd)
Post to thread semaphore.
- void [esThdWaitl](#) (void)
Wait for thread semaphore.
- void [esThdWait](#) (void)
Wait for thread semaphore.
- #define [ES_STCK_SIZE](#)(elem) [PORT_STCK_SIZE](#)(elem)
Converts the required stack elements into the stack array index.

Virtual Timer management

- typedef uint_fast32_t [esTick_T](#)
Timer tick type.
- typedef struct [esVTmr](#) [esVTmr_T](#)
Virtual Timer type.
- void [esVTmrInitl](#) ([esVTmr_T](#) *vTmr, [esTick_T](#) tick, void(*fn)(void *), void *arg)
Add and start a new virtual timer.
- void [esVTmrlnit](#) ([esVTmr_T](#) *vTmr, [esTick_T](#) tick, void(*fn)(void *), void *arg)
Add and start a new virtual timer.
- void [esVTmrTerml](#) ([esVTmr_T](#) *vTmr)
Cancel and remove a virtual timer.
- void [esVTmrTerm](#) ([esVTmr_T](#) *vTmr)
Cancel and remove a virtual timer.
- void [esVTmrDelay](#) ([esTick_T](#) tick)
Delay for specified amount of ticks.

Thread Queue management

- typedef struct `esThdQ` `esThdQ_T`
Thread queue type.
- void `esThdQInit` (`esThdQ_T` *thdQ)
Initialize Thread Queue.
- void `esThdQTerm` (`esThdQ_T` *thdQ)
Terminate Thread Queue.
- void `esThdQAddI` (`esThdQ_T` *thdQ, `esThd_T` *thd)
Add a thread to the Thread Queue.
- void `esThdQRml` (`esThdQ_T` *thdQ, `esThd_T` *thd)
Removes the thread from the Thread Queue.
- `esThd_T` * `esThdQFetchI` (const `esThdQ_T` *thdQ)
Fetch the first high priority thread from the Thread Queue.
- `esThd_T` * `esThdQFetchRotatel` (`esThdQ_T` *thdQ, `uint_fast8_t` prio)
Fetch the next thread and rotate thread linked list.
- `bool_T` `esThdQIsEmpty` (const `esThdQ_T` *thdQ)
Is thread queue empty.
- #define `PRIO_BM_GRP_INDX` ((`CFG_SCHED_PRIO_LVL` + `PORT_DATA_WIDTH_VAL` - 1U) / `PORT_DATA_WIDTH_VAL`)
Priority Bit Map Group Index.

Kernel control block

- enum `esKernState` {
 `ES_KERN_RUN` = 0x00U,
 `ES_KERN_INTSRV_RUN` = 0x01U,
 `ES_KERN_LOCK` = 0x02U,
 `ES_KERN_INTSRV_LOCK` = 0x03U,
 `ES_KERN_SLEEP` = 0x06U,
 `ES_KERN_INIT` = 0x08U,
 `ES_KERN_INACTIVE` = 0x10U }
Kernel state enumeration.
- typedef enum `esKernState` `esKernState_T`
Kernel state type.
- typedef struct `esKernCtrl` `esKernCtrl_T`
Kernel control block type.
- const volatile `esKernCtrl_T` `gKernCtrl`
Kernel control block.

General kernel functions

There are several groups of functions:

- kernel initialization and start
- ISR prologue and epilogue
- void `esKernInit` (void)
Initialize kernel internal data structures.
- `PORT_C_NORETURN` void `esKernStart` (void)
Start the multi-threading.

- void `esKernSysTmr` (void)
Process the system timer event.
- void `esKernIsrPrologueI` (void)
Enter Interrupt Service Routine.
- void `esKernIsrEpilogueI` (void)
Exit Interrupt Service Routine.

Scheduler notification and invocation

- void `esSchedRdyAddI` (`esThd_T` *thd)
Add thread `thd` to the ready thread list and notify the scheduler.
- void `esSchedRdyRml` (`esThd_T` *thd)
Remove thread `thd` from the ready thread list and notify the scheduler.
- void `esSchedYieldI` (void)
Force the scheduler invocation which will evaluate all ready threads and switch to ready thread with the highest priority.
- void `esSchedYieldIsrI` (void)
Force the scheduler invocation which will evaluate all ready threads and switch to ready thread with the highest priority.
- void `esSchedLockEnterI` (void)
Lock the scheduler.
- void `esSchedLockExitI` (void)
Unlock the scheduler.
- void `esSchedLockEnter` (void)
Lock the scheduler.
- void `esSchedLockExit` (void)
Unlock the scheduler.

Kernel hook functions

Note

- 1) The definition of this functions must be written by the user.

- void `userPreSysTmr` (void)
System timer hook function, called from system system timer ISR function before the kernel functions.
- void `userPreKernInit` (void)
Kernel initialization hook function, called from `esKernInit()` function before kernel initialization.
- void `userPostKernInit` (void)
Kernel initialization hook function, called from `esKernInit()` function after kernel initialization.
- void `userPreKernStart` (void)
Kernel start hook function, called from `esKernStart()` function.
- void `userPostThdInit` (`esThd_T` *thd)
Thread initialization end hook function, called from `esThdInit()` function.
- void `userPreThdTerm` (void)
Thread terminate hook function, called from `esThdTerm()` or when a thread terminates itself.
- void `userPreIdle` (void)
Pre Idle hook function, called from idle thread, just before entering idle period.
- void `userPostIdle` (void)
Post idle hook function, called from idle thread, just after exiting idle period.
- void `userPreCtxSw` (`esThd_T` *oldThd, `esThd_T` *newThd)
Kernel context switch hook function, called from `esSchedYieldI()` and `esSchedYieldIsrI()` functions just before context switch.

9.2.1 Detailed Description

Application programming interface.

9.2.2 Macro Definition Documentation

9.2.2.1 `#define ES_KERN_VER 0x10000UL`

Identifies the underlying kernel version number.

Kernel identification and version (main [31:16] .sub [15:0])

9.2.2.2 `#define ES_KERN_ID "eSolid Kernel v1.0"`

Kernel identification string.

9.2.2.3 `#define ES_CRITICAL_DECL() PORT_CRITICAL_DECL()`

Critical section status variable declaration.

9.2.2.4 `#define ES_CRITICAL_ENTER() PORT_CRITICAL_ENTER()`

Enter a critical section.

9.2.2.5 `#define ES_CRITICAL_EXIT() PORT_CRITICAL_EXIT()`

Exit from critical section.

9.2.2.6 `#define ES_CRITICAL_ENTER_LOCK_EXIT()`

Value:

```
do {
    PORT_CRITICAL_ENTER();
    esSchedLockExitI();
} while (0U)
```

Enter critical section and exit scheduler lock.

9.2.2.7 `#define ES_CRITICAL_EXIT_LOCK_ENTER()`

Value:

```
do {
    esSchedLockEnterI();
    PORT_CRITICAL_EXIT();
} while (0U)
```

Exit critical section and enter scheduler lock.

9.2.2.8 `#define ES_STCK_SIZE(elem) PORT_STCK_SIZE(elem)`

Converts the required stack elements into the stack array index.

Parameters

<i>elem</i>	Number of stack elements: the stack size is expressed in number of elements regardless of the size of port general purpose registers.
-------------	---

9.2.2.9 `#define PRIO_BM_GRP_IDX ((CFG_SCHED_PRIO_LVL + PORT_DATA_WIDTH_VAL - 1U) / PORT_DATA_WIDTH_VAL)`

Priority Bit Map Group Index.

Object class:

Not API object, this object is not part of the application programming interface and it is intended for internal use only.

9.2.3 Typedef Documentation

9.2.3.1 `typedef struct esThd esThd_T`

Thread type.

9.2.3.2 `typedef portStck_T esStck_T`

Stack type.

9.2.3.3 `typedef uint_fast32_t esTick_T`

Timer tick type.

9.2.3.4 `typedef struct esVTmr esVTmr_T`

Virtual Timer type.

9.2.3.5 `typedef struct esThdQ esThdQ_T`

Thread queue type.

9.2.3.6 `typedef enum esKernState esKernState_T`

Kernel state type.

9.2.3.7 `typedef struct esKernCtrl esKernCtrl_T`

Kernel control block type.

9.2.4 Enumeration Type Documentation

9.2.4.1 `enum esKernState`

Kernel state enumeration.

For more details see: [Kernel states](#)

Object class:

Regular **API** object, this object is part of the application programming interface.

Enumerator

ES_KERN_RUN Kernel is active

ES_KERN_INTSRV_RUN Servicing an interrupt return to ES_KERN_RUN state

ES_KERN_LOCK Kernel is locked

ES_KERN_INTSRV_LOCK Servicing an interrupt, return to ES_KERN_LOCK state

ES_KERN_SLEEP Kernel is sleeping

ES_KERN_INIT Kernel is in initialization state

ES_KERN_INACTIVE Kernel data structures are not initialized

9.2.5 Function Documentation

9.2.5.1 void esKernInit (void)

Initialize kernel internal data structures.

Precondition

- 1) The kernel state == ES_KERN_INACTIVE, see [Kernel states](#).

Postcondition

- 1) The kernel state == ES_KERN_INIT.

Note

- 1) This function may be invoked only once.

This function must be called first before any other kernel API. It initializes internal data structures that are used by other API functions.

Object class:

Regular **API** object, this object is part of the application programming interface.

9.2.5.2 PORT_C_NORETURN void esKernStart (void)

Start the multi-threading.

Precondition

- 1) The kernel state == ES_KERN_INIT, see [Kernel states](#).

Postcondition

- 1) The kernel state == ES_KERN_RUN
- 2) The multi-threading execution will commence.

Note

- 1) Once this function is called the execution of threads will start and this function will never return.

This function will start multi-threading. Once the multi-threading has started the execution will never return to this function again (this function never returns).

Object class:

Regular **API** object, this object is part of the application programming interface.

9.2.5.3 void esKernSysTmr (void)

Process the system timer event.

Precondition

- 1) The kernel state < ES_KERN_INIT, see [Kernel states](#).

This function will be called only by port system timer interrupt.

Object class:

Not API object, this object is not part of the application programming interface and it is intended for internal use only.

9.2.5.4 void esKernIsrPrologueI(void)

Enter Interrupt Service Routine.

Precondition

- 1) The kernel state < ES_KERN_INIT, see [Kernel states](#).

Note

- 1) You must call [esKernIsrEpilogueI\(\)](#) at the exit of ISR.
- 2) You must invoke [esKernIsrPrologueI\(\)](#) and [esKernIsrEpilogueI\(\)](#) in pair. In other words, for every call to [esKernIsrPrologueI\(\)](#) at the beginning of the ISR you must have a call to [esKernIsrEpilogueI\(\)](#) at the end of the ISR.

Function will notify kernel that you are about to enter interrupt service routine (ISR). This allows kernel to keep track of interrupt nesting and then only perform rescheduling at the last nested ISR.

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.2.5.5 void esKernIsrEpilogueI(void)

Exit Interrupt Service Routine.

Precondition

- 1) The kernel state < ES_KERN_INIT, see [Kernel states](#).

Note

- 1) You must invoke [esKernIsrPrologueI\(\)](#) and [esKernIsrEpilogueI\(\)](#) in pair. In other words, for every call to [esKernIsrPrologueI\(\)](#) at the beginning of the ISR you must have a call to [esKernIsrEpilogueI\(\)](#) at the end of the ISR.
- 2) Rescheduling is prevented when the scheduler is locked (see [esSchedLockEnterI\(\)](#))

This function is used to notify kernel that you have completed servicing an interrupt. When the last nested ISR has completed, the function will call the scheduler to determine whether a new, high-priority task, is ready to run.

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.2.5.6 void esThdInit(esThd_T * thd, void(*)(void *) fn, void * arg, portStck_T * stck, size_t stckSize, uint8_t prio)

Initialize the specified thread.

Parameters

<i>thd</i>	Thread: is a pointer to the thread structure, esThd . The structure will be used as information container for the thread. It is assumed that storage for the esThd structure is allocated by the user code.
<i>fn</i>	Function: is a pointer to thread function. Thread function must have the following signature: <code>void thread (void * arg).</code>
<i>arg</i>	Argument: is a void pointer to an optional data area. It's usage is application defined and it is intended to pass arguments to thread when it is started for the first time.

<i>stck</i>	Stack: is a pointer to a allocated memory for thread stack. The pointer always points to the first element in the array, regardless of what type of stack the CPU is using. The thread's stack is used to store local variables, function parameters, return addresses. Each thread has its own stack and different sized stack. The stack type must be an array of portStck .
<i>stckSize</i>	Stack Size: specifies the size of allocated stack memory. Size is expressed in bytes. Please see port documentation about minimal stack size. Usage of C unary operator <code>sizeof</code> is the recommended way of specifying stack size.
<i>prio</i>	Priority: is the priority of the thread. The higher the number, the higher the priority (the importance) of the thread. Several threads can have the same priority. Note that lowest (0) and highest (<code>CFG_SCHED_PRIO_LVL - 1</code>) levels are reserved for kernel threads only.

Precondition

- 1) The kernel state `ES_KERN_INACTIVE`, see [Kernel states](#).
- 2) `thd != NULL`
- 3) `thd->signature != THD_CONTRACT_SIGNATURE`, the thread structure can't be initialized more than once.
- 4) `fn != NULL`
- 5) `stckSize >= PORT_STCK_MINSIZE_VAL`, see [PORT_STCK_MINSIZE_VAL](#).
- 6) `0 < prio < CFG_SCHED_PRIO_LVL - 1`, see [CFG_SCHED_PRIO_LVL](#).

Postcondition

- 1) `thd->signature == THD_CONTRACT_SIGNATURE`, each [esThd](#) structure will have valid signature after initialization.

Threads must be created in order for kernel to recognize them as threads. Initialize a thread by calling [esThd-Init\(\)](#) and provide arguments specifying to kernel how the thread will be managed. Threads are always created in the ready-to-run state. Threads can be created either prior to the start of multi-threading (before calling [esKernStart\(\)](#)), or by a running thread.

Object class:

Regular **API** object, this object is part of the application programming interface.

9.2.5.7 void esThdTerm (esThd_T * thd)

Terminate the specified thread.

Parameters

<i>thd</i>	Thread: is a pointer to the thread structure, esThd .
------------	---

Precondition

- 1) The kernel state `ES_KERN_INACTIVE`, see [Kernel states](#).
- 2) `thd != NULL`
- 3) `thd->signature == THD_CONTRACT_SIGNATURE`, the pointer must point to a valid [esThd](#) structure.
- 4) `(thd->thdL.q == NULL) OR (thd->thdL.q == gRdyQueue)`, thread must be either in Ready Threads Queue or not be in any queue (e.g. not waiting for a synchronization mechanism).

Postcondition

- 1) `thd->signature == ~THD_CONTRACT_SIGNATURE`, each [esThd](#) structure will have invalid signature after termination.

Object class:

Regular **API** object, this object is part of the application programming interface.

9.2.5.8 static PORT_C_INLINE esThd_T* esThdGetId (void) [static]

Get the current thread ID.

Returns

Pointer to current thread ID structure [esThd](#).

Note

This is `inline` function.

Object class:

Regular **API** object, this object is part of the application programming interface.

9.2.5.9 static PORT_C_INLINE uint8_t esThdGetPrio (esThd_T * thd) [static]

Get the priority of a thread.

Parameters

<i>thd</i>	Thread: is pointer to the thread structure, esThd .
------------	---

Returns

The priority of the thread pointed by `thd`.

Note

This is `inline` function.

Object class:

Regular **API** object, this object is part of the application programming interface.

9.2.5.10 void esThdSetPrio (esThd_T * thd, uint8_t prio)

Set the priority of a thread.

Parameters

<i>thd</i>	Thread: is pointer to the thread structure, esThd .
<i>prio</i>	Priority: is new priority of the thread pointed by <code>thd</code> .

Precondition

- 1) The kernel state < ES_KERN_INACTIVE, see [Kernel states](#).
- 2) thd != NULL
- 3) thd->signature == THD_CONTRACT_SIGNATURE, the pointer must point to a valid [esThd](#) structure.
- 4) 0 < prio < CFG_SCHED_PRIO_LVL - 1, see [CFG_SCHED_PRIO_LVL](#).

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.2.5.11 void esThdPostl (esThd_T * thd)

Post to thread semaphore.

Parameters

<i>thd</i>	Pointer to the thread ID structure
------------	------------------------------------

Precondition

- 1) The kernel state < ES_KERN_INACTIVE, see [Kernel states](#).
- 2) thd != NULL
- 3) thd->signature == THD_CONTRACT_SIGNATURE, the pointer must point to a valid [esThd](#) structure.

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.2.5.12 void esThdPost (esThd_T * thd)

Post to thread semaphore.

Parameters

<i>thd</i>	Pointer to the thread ID structure
------------	------------------------------------

Precondition

- 1) The kernel state < ES_KERN_INACTIVE, see [Kernel states](#).
- 2) thd != NULL
- 3) thd->signature == THD_CONTRACT_SIGNATURE, the pointer must point to a valid [esThd](#) structure.

Object class:

Regular **API** object, this object is part of the application programming interface.

9.2.5.13 void esThdWaitl (void)

Wait for thread semaphore.

Precondition

- 1) The `kernel state == ES_KERN_RUN`, see [Kernel states](#).

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.2.5.14 void esThdWait (void)

Wait for thread semaphore.

Precondition

- 1) The `kernel state == ES_KERN_RUN`, see [Kernel states](#).

Object class:

Regular **API** object, this object is part of the application programming interface.

9.2.5.15 void esThdQInit (esThdQ_T * thdQ)

Initialize Thread Queue.

Parameters

<i>thdQ</i>	Thread Queue: is a pointer to thread queue structure, esThdQ .
-------------	--

Precondition

- 1) `thdQ != NULL`
- 2) `thdQ->signature != THDQ_CONTRACT_SIGNATURE`, the thread queue structure can't be initialized more than once.

Postcondition

- 1) `thdQ->signature == THDQ_CONTRACT_SIGNATURE`, each [esThdQ](#) structure will have valid signature after initialization.

Object class:

Regular **API** object, this object is part of the application programming interface.

9.2.5.16 void esThdQTerm (esThdQ_T * thdQ)

Terminate Thread Queue.

Parameters

<i>thdQ</i>	Thread Queue: is a pointer to thread queue structure, esThdQ .
-------------	--

Precondition

- 1) `thdQ != NULL`
- 2) `thdQ->signature == THDQ_CONTRACT_SIGNATURE`, the thread queue structure must be already initialized.

Postcondition

1) `thdQ->signature == ~THDQ_CONTRACT_SIGNATURE`, each `esThdQ` structure will have invalid signature after termination.

Object class:

Regular **API** object, this object is part of the application programming interface.

9.2.5.17 void esThdQAddl (esThdQ_T * thdQ, esThd_T * thd)

Add a thread to the Thread Queue.

Parameters

<i>thdQ</i>	Thread Queue: is a pointer to thread queue structure, esThdQ .
<i>thd</i>	Thread: is a pointer to the thread ID structure, esThd .

Precondition

- 1) `thdQ != NULL`
- 2) `thdQ->signature == THDQ_CONTRACT_SIGNATURE`, the pointer must point to a valid [esThdQ](#) structure.
- 3) `thd != NULL`
- 4) `thd->signature == THD_CONTRACT_SIGNATURE`, the pointer must point to a valid [esThd](#) structure.
- 5) `thd->thdL.q == NULL`, thread must not be in any queue.

This function adds a thread at the specified Thread Queue.

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.2.5.18 void esThdQRml (esThdQ_T * thdQ, esThd_T * thd)

Removes the thread from the Thread Queue.

Parameters

<i>thdQ</i>	Thread Queue: is a pointer to thread queue structure, esThdQ .
<i>thd</i>	Thread: is a pointer to the thread ID structure, esThd .

Precondition

- 1) `thd != NULL`
- 2) `thd->signature == THD_CONTRACT_SIGNATURE`, the pointer must point to a valid [esThd](#) structure.
- 3) `thdQ != NULL`
- 4) `thdQ->signature == THDQ_CONTRACT_SIGNATURE`, the pointer must point to a valid [esThdQ](#) structure.
- 5) `thd->thdL.q == thdQ`, thread must be in the `thdQ` queue.

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.2.5.19 `esThd_T* esThdQFetchl (const esThdQ_T * thdQ)`

Fetch the first high priority thread from the Thread Queue.

Parameters

<i>thdQ</i>	Thread Queue: is a pointer to thread queue structure, esThdQ .
-------------	--

Returns

A pointer to the thread ID structure with the highest priority.

Precondition

- 1) `thdQ != NULL`
- 2) `thdQ->signature == THDQ_CONTRACT_SIGNATURE`, the pointer must point to a valid [esThdQ](#) structure.
- 3) `prioBM != 0`, priority bit map must not be empty

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.2.5.20 `esThd_T* esThdQFetchRotatel (esThdQ_T * thdQ, uint_fast8_t prio)`

Fetch the next thread and rotate thread linked list.

Parameters

<i>thdQ</i>	Thread Queue: is a pointer to thread queue structure, esThdQ . This is the thread queue to fetch from.
<i>prio</i>	Priority: is the priority level to fetch and rotate.

Returns

Pointer to the next thread in queue.

Precondition

- 1) `thdQ != NULL`
- 2) `thdQ->signature == THDQ_CONTRACT_SIGNATURE`, the pointer must point to a valid [esThdQ](#) structure.
- 3) `0 <= prio <= CFG_SCHED_PRIO_LVL`, see [CFG_SCHED_PRIO_LVL](#).
- 4) `sentinel != NULL`, at least one thread must be in the selected priority level

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.2.5.21 `bool_T esThdQIsEmpty (const esThdQ_T * thdQ)`

Is thread queue empty.

Parameters

<i>thdQ</i>	Thread Queue: is a pointer to thread queue structure, esThdQ .
-------------	--

Returns

The state of thread queue

Return values

<i>TRUE</i>	- thread queue is empty
<i>FALSE</i>	- thread queue is not empty

Precondition

- 1) `thdQ != NULL`
- 2) `thdQ->signature == THDQ_CONTRACT_SIGNATURE`, the pointer must point to a valid [esThdQ](#) structure.

Object class:

Regular **API** object, this object is part of the application programming interface.

9.2.5.22 void esSchedRdyAddl (esThd_T * thd)

Add thread `thd` to the ready thread list and notify the scheduler.

Parameters

<i>thd</i>	Pointer to the initialized thread ID structure, esThd .
------------	---

Precondition

- 1) The kernel state < ES_KERN_INACTIVE, see [Kernel states](#).
- 2) `thd != NULL`
- 3) `thd->signature == THD_CONTRACT_SIGNATURE`, the pointer must point to a valid [esThd](#) structure.
- 4) `thd->thdL.q == NULL`, thread must not be in a queue.

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.2.5.23 void esSchedRdyRml (esThd_T * thd)

Remove thread `thd` from the ready thread list and notify the scheduler.

Parameters

<i>thd</i>	Pointer to the initialized thread ID structure, esThd .
------------	---

Precondition

- 1) The kernel state < ES_KERN_INACTIVE, see [Kernel states](#).
- 2) `thd != NULL`
- 3) `thd->signature == THD_CONTRACT_SIGNATURE`, the pointer must point to a valid [esThd](#) structure.
- 4) `thd->thdL.q == &gRdyQueue`, thread must be in Ready Threads queue.

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.2.5.24 void esSchedYieldl (void)

Force the scheduler invocation which will evaluate all ready threads and switch to ready thread with the highest priority.

Precondition

- 1) The `kernel state < ES_KERN_INACTIVE`, see [Kernel states](#).

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.2.5.25 void esSchedYieldlsl (void)

Force the scheduler invocation which will evaluate all ready threads and switch to ready thread with the highest priority.

Precondition

- 1) The `kernel state < ES_KERN_INACTIVE`, see [Kernel states](#).

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.2.5.26 void esSchedLockEnterl (void)

Lock the scheduler.

Precondition

- 1) The `kernel state < ES_KERN_INIT`, see [Kernel states](#).

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.2.5.27 void esSchedLockExitl (void)

Unlock the scheduler.

Precondition

- 1) The `kernel state < ES_KERN_INIT`, see [Kernel states](#).
- 2) `gKernLockCnt > 0U`, current number of locks must be greater than zero, in other words: each call to kernel lock function must have its matching call to kernel unlock function.

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.2.5.28 void esSchedLockEnter (void)

Lock the scheduler.

Precondition

- 1) The `kernel state < ES_KERN_INIT`, see [Kernel states](#).

Object class:

Regular **API** object, this object is part of the application programming interface.

9.2.5.29 void esSchedLockExit (void)

Unlock the scheduler.

Precondition

- 1) The `kernel state < ES_KERN_INIT`, see [Kernel states](#).
- 2) `gKernLockCnt > 0U`, current number of locks must be greater than zero, in other words: each call to kernel lock function must have its matching call to kernel unlock function.

Object class:

Regular **API** object, this object is part of the application programming interface.

9.2.5.30 void esVTmrInitl (esVTmr_T * vTmr, esTick_T tick, void(*)(void *) fn, void * arg)

Add and start a new virtual timer.

Parameters

<i>vTmr</i>	Virtual Timer: is pointer to the timer ID structure, esVTmr .
<i>tick</i>	Tick: the timer delay expressed in system ticks
<i>fn</i>	Function: is pointer to the callback function
<i>arg</i>	Argument: is pointer to the arguments of callback function

Precondition

- 1) The `kernel state < ES_KERN_INACTIVE`, see [Kernel states](#).
- 2) `vTmr != NULL`
- 3) `vTmr->signature != VTMR_CONTRACT_SIGNATURE`, the timer structure can't be initialized more than once.
- 4) `tick > 1U`
- 5) `fn != NULL`

Postcondition

- 1) `vTmr->signature == VTMR_CONTRACT_SIGNATURE`, each [esVTmr](#) structure will have valid signature after initialization.

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.2.5.31 void esVTmrInit (esVTmr_T * vTmr, esTick_T tick, void(*)(void *) fn, void * arg)

Add and start a new virtual timer.

Parameters

<i>vTmr</i>	Virtual Timer: is pointer to the timer ID structure, esVTmr .
<i>tick</i>	Tick: the timer delay expressed in system ticks
<i>fn</i>	Function: is pointer to the callback function
<i>arg</i>	Argument: is pointer to the arguments of callback function

Precondition

- 1) The kernel state < ES_KERN_INACTIVE, see [Kernel states](#).
- 2) vTmr != NULL
- 3) vTmr->signature != VTMR_CONTRACT_SIGNATURE, the timer structure can't be initialized more than once.
- 4) tick > 1U
- 5) fn != NULL

Postcondition

- 1) vTmr->signature == VTMR_CONTRACT_SIGNATURE, each [esVTmr](#) structure will have valid signature after initialization.

Object class:

Regular **API** object, this object is part of the application programming interface.

9.2.5.32 void esVTmrTerm (esVTmr_T * vTmr)

Cancel and remove a virtual timer.

Parameters

<i>vTmr</i>	Timer: is pointer to the timer ID structure, esVTmr .
-------------	---

Precondition

- 1) The kernel state < ES_KERN_INACTIVE, see [Kernel states](#).
- 2) vTmr != NULL
- 3) vTmr->signature == VTMR_CONTRACT_SIGNATURE, the pointer must point to a valid [esVTmr](#) structure.

Postcondition

- 1) vTmr->signature = ~VTMR_CONTRACT_SIGNATURE, each [esVTmr](#) structure will have invalid signature after termination.

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.2.5.33 void esVTmrTerm (esVTmr_T * vTmr)

Cancel and remove a virtual timer.

Parameters

<i>vTmr</i>	Timer: is pointer to the timer ID structure, esVTmr .
-------------	---

Precondition

- 1) The kernel state < ES_KERN_INACTIVE, see [Kernel states](#).
- 2) vTmr != NULL
- 3) vTmr->signature == VTMR_CONTRACT_SIGNATURE, the pointer must point to a valid [esVTmr](#) structure.

Postcondition

1) `vTmr->signature = ~VTMR_CONTRACT_SIGNATURE`, each `esVTmr` structure will have invalid signature after termination.

Object class:

Regular **API** object, this object is part of the application programming interface.

9.2.5.34 void esVTmrDelay (esTick_T tick)

Delay for specified amount of ticks.

Parameters

<i>tick</i>	Tick: number of system ticks to delay.
-------------	--

This function will create a virtual timer with count down time specified in argument `tick` and put the calling thread into `sleep` state. When timeout expires the thread will be placed back into `ready` state.

Precondition

1) `tick > 1U`

Object class:

Regular **API** object, this object is part of the application programming interface.

9.2.5.35 void userPreSysTmr (void)

System timer hook function, called from system system timer ISR function before the kernel functions.

Note

1) This function is called only if `CFG_HOOK_PRE_SYSTMTR_EVENT` is active.

This function is called whenever a system event is generated.

9.2.5.36 void userPreKernInit (void)

Kernel initialization hook function, called from `esKernInit()` function before kernel initialization.

Note

1) This function is called only if `CFG_HOOK_PRE_KERN_INIT` is active.

This function is called before the kernel initialization.

9.2.5.37 void userPostKernInit (void)

Kernel initialization hook function, called from `esKernInit()` function after kernel initialization.

Note

1) This function is called only if `CFG_HOOK_POST_KERN_INIT` is active.

This function is called after the kernel initialization.

9.2.5.38 void userPreKernStart (void)

Kernel start hook function, called from `esKernStart()` function.

Note

- 1) This function is called only if [CFG_HOOK_PRE_KERN_START](#) is active.

This function is called before kernel start.

9.2.5.39 void userPostThdInit (esThd_T * *thd*)

Thread initialization end hook function, called from [esThdInit\(\)](#) function.

Parameters

<i>thd</i>	Thread: pointer to thread Id structure that has just been initialized.
------------	--

Note

- 1) This function is called only if [CFG_HOOK_POST_THD_INIT](#) is active.

This function is called after the thread initialization.

9.2.5.40 void userPreThdTerm (void)

Thread terminate hook function, called from [esThdTerm\(\)](#) or when a thread terminates itself.

Note

- 1) This function is called only if [CFG_HOOK_PRE_THD_TERM](#) is active.

9.2.5.41 void userPreIdle (void)

Pre Idle hook function, called from idle thread, just before entering idle period.

Note

- 1) This function is called only if [CFG_HOOK_PRE_IDLE](#) and [CFG_SCHED_POWER_SAVE](#) are active.
2) This function is called with interrupts and scheduler locked.

9.2.5.42 void userPostIdle (void)

Post idle hook function, called from idle thread, just after exiting idle period.

Note

- 1) This function is called only if [CFG_HOOK_POST_IDLE](#) and [CFG_SCHED_POWER_SAVE](#) are active.
2) This function is called with scheduler locked.

9.2.5.43 void userPreCtxSw (esThd_T * *oldThd*, esThd_T * *newThd*)

Kernel context switch hook function, called from [esSchedYield\(\)](#) and [esSchedYieldIsrl\(\)](#) functions just before context switch.

Parameters

<i>oldThd</i>	Pointer to the thread being switched out.
<i>newThd</i>	Pointer to the thread being switched in.

Note

- 1) This function is called only if [CFG_HOOK_PRE_CTX_SW](#) is active.

This function is called at each context switch.

9.2.6 Variable Documentation**9.2.6.1 `const volatile esKernCtrl_T gKernCtrl`**

Kernel control block.

Note

This variable has Read-Only access rights for application.

9.3 Internals

Kernel inner work.

Collaboration diagram for Internals:



Data Structures

- struct `sysTmr`
Main System Timer structure.

Macros

- #define `PRIO_BM_DATA_WIDTH_LOG2`
Priority Bit Map log base 2: $\log_2(\text{PORT_DATA_WIDTH_VAL})$
- #define `SCHED_STATE_INTSRV_MSK` (1U << 0)
Kernel state variable bit position which defines if the kernel is in interrupt servicing state.
- #define `SCHED_STATE_LOCK_MSK` (1U << 1)
Kernel state variable bit position which defines if the kernel is locked or not.
- #define `THD_CONTRACT_SIGNATURE` ((portReg_T)0xFEEDBEEFUL)
Thread structure signature.
- #define `THDQ_CONTRACT_SIGNATURE` ((portReg_T)0xFEEDBEEFUL)
Thread Queue structure signature.
- #define `VTMR_CONTRACT_SIGNATURE` ((portReg_T)0xFEEDBCCCUL)
Timer structure signature.
- #define `DLIST_IS_ENTRY_FIRST`(list, entry) ((entry) == (entry)->list.next)
DList macro: is the thread the first one in the list.
- #define `DLIST_IS_ENTRY_LAST`(list, entry) `DLIST_IS_ENTRY_FIRST`(list, entry)
DList macro: is the thread the last one in the list.
- #define `DLIST_IS_ENTRY_SINGLE`(list, entry) `DLIST_IS_ENTRY_FIRST`(list, entry)
DList macro: is the thread single in the list.
- #define `DLIST_ENTRY_NEXT`(list, entry) (entry)->list.next
DList macro: get the next entry.
- #define `DLIST_ENTRY_INIT`(list, entry)
DList macro: initialize entry.
- #define `DLIST_ENTRY_ADD_AFTER`(list, current, entry)
DList macro: add new entry after current entry.
- #define `DLIST_ENTRY_RM`(list, entry)
DList macro: remove the entry from a list.

Functions

- void `esKernInit` (void)
Initialize kernel internal data structures.
- PORT_C_NORETURN void `esKernStart` (void)
Start the multi-threading.
- void `esKernSysTmr` (void)
Process the system timer event.
- void `esKernIsrPrologueI` (void)
Enter Interrupt Service Routine.
- void `esKernIsrEpilogueI` (void)
Exit Interrupt Service Routine.
- void `esThdInit` (`esThd_T` *thd, void(*fn)(void *), void *arg, `portStck_T` *stck, `size_t` stckSize, `uint8_t` prio)
Initialize the specified thread.
- void `esThdTerm` (`esThd_T` *thd)
Terminate the specified thread.
- void `esThdSetPrioI` (`esThd_T` *thd, `uint8_t` prio)
Set the priority of a thread.
- void `esThdPostI` (`esThd_T` *thd)
Post to thread semaphore.
- void `esThdPost` (`esThd_T` *thd)
Post to thread semaphore.
- void `esThdWaitI` (void)
Wait for thread semaphore.
- void `esThdWait` (void)
Wait for thread semaphore.
- void `esThdQInit` (`esThdQ_T` *thdQ)
Initialize Thread Queue.
- void `esThdQTerm` (`esThdQ_T` *thdQ)
Terminate Thread Queue.
- void `esThdQAddI` (`esThdQ_T` *thdQ, `esThd_T` *thd)
Add a thread to the Thread Queue.
- void `esThdQRml` (`esThdQ_T` *thdQ, `esThd_T` *thd)
Removes the thread from the Thread Queue.
- `esThd_T` * `esThdQFetchI` (const `esThdQ_T` *thdQ)
Fetch the first high priority thread from the Thread Queue.
- `esThd_T` * `esThdQFetchRotatel` (`esThdQ_T` *thdQ, `uint_fast8_t` prio)
Fetch the next thread and rotate thread linked list.
- `bool_T` `esThdQIsEmpty` (const `esThdQ_T` *thdQ)
Is thread queue empty.
- void `esSchedRdyAddI` (`esThd_T` *thd)
Add thread `thd` to the ready thread list and notify the scheduler.
- void `esSchedRdyRml` (`esThd_T` *thd)
Remove thread `thd` from the ready thread list and notify the scheduler.
- void `esSchedYieldI` (void)
Force the scheduler invocation which will evaluate all ready threads and switch to ready thread with the highest priority.
- void `esSchedYieldIsrcI` (void)
Force the scheduler invocation which will evaluate all ready threads and switch to ready thread with the highest priority.
- void `esSchedLockEnterI` (void)

- *Lock the scheduler.*
- void `esSchedLockExitl` (void)
- *Unlock the scheduler.*
- void `esSchedLockEnter` (void)
- *Lock the scheduler.*
- void `esSchedLockExit` (void)
- *Unlock the scheduler.*
- void `esVTmrInitl` (`esVTmr_T` *vTmr, `esTick_T` tick, void(*fn)(void *), void *arg)
- *Add and start a new virtual timer.*
- void `esVTmrInit` (`esVTmr_T` *vTmr, `esTick_T` tick, void(*fn)(void *), void *arg)
- *Add and start a new virtual timer.*
- void `esVTmrTermI` (`esVTmr_T` *vTmr)
- *Cancel and remove a virtual timer.*
- void `esVTmrTerm` (`esVTmr_T` *vTmr)
- *Cancel and remove a virtual timer.*
- void `esVTmrDelay` (`esTick_T` tick)
- *Delay for specified amount of ticks.*

Variables

- static uint_fast8_t `gKernLockCnt`
- *Kernel Lock Counter.*
- const volatile `esKernCtrl_T` `gKernCtrl`
- *Kernel control initialization.*

System timer

- typedef struct `sysTmr` `sysTmr_T`
- *System Timer type.*
- static `sysTmr_T` `gSysTmr`
- *Main System Timer structure.*
- static `esVTmr_T` `gVTmrArmed`
- *List of virtual timers to armed expire.*
- static `esVTmr_T` `gVTmrPend`
- *Virtual timers pending to be inserted into waiting list.*
- static `esThd_T` `gKVTmr`
- *Virtual timer thread ID.*
- static `PORT_C_INLINE` void `sysTmrInit` (void)
- *Initialize system timer hardware.*
- static `PORT_C_INLINE` void `sysTmrActivate` (void)
- *Try to activate system timer.*
- static `PORT_C_INLINE` void `sysTmrDeactivatel` (void)
- *Try to deactivate system timer.*

Priority Bit Map

- typedef struct prioBM [prioBM_T](#)
Priority Bit Map type.
- static [PORT_C_INLINE](#) void [prioBMInit](#) ([prioBM_T](#) *bm)
Initialize bitmap.
- static [PORT_C_INLINE](#) void [prioBMSet](#) ([prioBM_T](#) *bm, uint_fast8_t prio)
Set the bit corresponding to the prio argument.
- static [PORT_C_INLINE](#) void [prioBMClear](#) ([prioBM_T](#) *bm, uint_fast8_t prio)
Clear the bit corresponding to the prio argument.
- static [PORT_C_INLINE](#) uint_fast8_t [prioBMGet](#) (const [prioBM_T](#) *bm)
Get the highest priority set.
- static [PORT_C_INLINE](#) bool_T [prioBMIsEmpty](#) (const [prioBM_T](#) *bm)
Is bit map empty?

Threads Queue

- typedef struct thdLSentinel [thdLSentinel_T](#)
Thread list sentinel type.

Scheduler

- static [esThdQ_T](#) [gRdyQueue](#)
Ready Thread queue.
- static [PORT_C_INLINE](#) void [schedInit](#) (void)
Initialize Ready Thread Queue structure [gRdyQueue](#) and Kernel control structure [esKernCtrl](#).
- static [PORT_C_INLINE](#) void [schedStart](#) (void)
Set the scheduler data structures for multi-threading.
- static [PORT_C_INLINE](#) void [schedSleep](#) (void)
Set the scheduler to sleep.
- static [PORT_C_INLINE](#) void [schedWakeUpl](#) (void)
Wake up the scheduler.
- static [PORT_C_INLINE](#) void [schedRdyAddInitl](#) ([esThd_T](#) *thd)
Initialize scheduler ready structure during the thread add operation.
- static [PORT_C_INLINE](#) void [schedQmNextl](#) (void)
Fetch and try to schedule the next thread of the same priority as the current thread.
- static [PORT_C_INLINE](#) void [schedQml](#) (void)
Do the Quantum (Round-Robin) scheduling.

Virtual Timer and Virtual Timer kernel thread

- static [PORT_C_INLINE](#) void [vTmrSleep](#) ([esTick_T](#) ticks)
Set up system timer for different tick period during sleeping.
- static [PORT_C_INLINE](#) void [vTmrEvaluatel](#) (void)
Evaluate armed virtual timers.
- static void [vTmrAddArmedS](#) ([esVTmr_T](#) *vTmr)
Add a virtual timer into sorted list.
- static [PORT_C_INLINE](#) void [vTmrImportPendl](#) (void)
Import timers from pending list to armed list.
- static void [vTmrImportPend](#) (void)
Import timers from pending list to armed list.

- static void `kVTmrInit` (void)
Initialization of Virtual Timer kernel thread.
- static void `kVTmr` (void *arg)
Virtual Timer thread code.

Idle kernel thread

- static `esThd_T gKIdle`
Idle thread ID.
- static void `kIdleInit` (void)
Initialization of Idle thread.
- static void `kIdle` (void *arg)
Idle thread code.

9.3.1 Detailed Description

Kernel inner work.

9.3.2 Macro Definition Documentation

9.3.2.1 #define PRIO_BM_DATA_WIDTH_LOG2

Value:

```
(PORT_DATA_WIDTH_VAL < 2 ? 0 :
 (PORT_DATA_WIDTH_VAL < 4 ? 1 :
  (PORT_DATA_WIDTH_VAL < 8 ? 2 :
   (PORT_DATA_WIDTH_VAL < 16 ? 3 :
    (PORT_DATA_WIDTH_VAL < 32 ? 4 :
     (PORT_DATA_WIDTH_VAL < 64 ? 5 :
      (PORT_DATA_WIDTH_VAL < 128 ? 6 : 7)))))))
```

Priority Bit Map log base 2: `log2 (PORT_DATA_WIDTH_VAL)`

9.3.2.2 #define SCHED_STATE_INTSRV_MSK (1U << 0)

Kernel state variable bit position which defines if the kernel is in interrupt servicing state.

9.3.2.3 #define SCHED_STATE_LOCK_MSK (1U << 1)

Kernel state variable bit position which defines if the kernel is locked or not.

9.3.2.4 #define THD_CONTRACT_SIGNATURE ((portReg_T)0xFEEDBEEFUL)

Thread structure signature.

The signature is used to confirm that a structure passed to a kernel function is indeed a `esThd_T` thread structure.

9.3.2.5 #define THDQ_CONTRACT_SIGNATURE ((portReg_T)0xFEEDBEEFUL)

Thread Queue structure signature.

The signature is used to confirm that a structure passed to a kernel function is indeed a `esThdQ_T` thread queue structure.

9.3.2.6 #define VTMR_CONTRACT_SIGNATURE ((portReg_T)0xFEEDBEEFUL)

Timer structure signature.

The signature is used to confirm that a structure passed to a timer function is indeed a `esVTmr_T` timer structure.

9.3.2.7 **#define DLIST_IS_ENTRY_FIRST(list, entry)** ((entry) == (entry)->list.next)

DList macro: is the thread the first one in the list.

9.3.2.8 **#define DLIST_IS_ENTRY_LAST(list, entry)** DLIST_IS_ENTRY_FIRST(list, entry)

DList macro: is the thread the last one in the list.

9.3.2.9 **#define DLIST_IS_ENTRY_SINGLE(list, entry)** DLIST_IS_ENTRY_FIRST(list, entry)

DList macro: is the thread single in the list.

9.3.2.10 **#define DLIST_ENTRY_NEXT(list, entry)** (entry)->list.next

DList macro: get the next entry.

9.3.2.11 **#define DLIST_ENTRY_INIT(list, entry)**

Value:

```
do {
    (entry)->list.next = (entry);
    (entry)->list.prev = (entry);
} while (0U)
```

DList macro: initialize entry.

9.3.2.12 **#define DLIST_ENTRY_ADD_AFTER(list, current, entry)**

Value:

```
do {
    (entry)->list.next = (current);
    (entry)->list.prev = (entry)->list.next->list.prev;
    (entry)->list.next->list.prev = (entry);
    (entry)->list.prev->list.next = (entry);
} while (0U)
```

DList macro: add new entry after current entry.

9.3.2.13 **#define DLIST_ENTRY_RM(list, entry)**

Value:

```
do {
    (entry)->list.next->list.prev = (entry)->list.prev;
    (entry)->list.prev->list.next = (entry)->list.next;
} while (0U)
```

DList macro: remove the entry from a list.

9.3.3 Typedef Documentation

9.3.3.1 **typedef struct sysTmr sysTmr_T**

System Timer type.

9.3.3.2 **typedef struct prioBM prioBM_T**

Priority Bit Map type.

9.3.3.3 **typedef struct thdLSentinel thdLSentinel_T**

Thread list sentinel type.

9.3.4 Function Documentation

9.3.4.1 `static PORT_C_INLINE void prioBMInit (prioBM_T * bm) [static]`

Initialize bitmap.

Parameters

<i>bm</i>	Pointer to the bit map structure
-----------	----------------------------------

9.3.4.2 `static PORT_C_INLINE void prioBMSet (prioBM_T * bm, uint_fast8_t prio) [static]`

Set the bit corresponding to the prio argument.

Parameters

<i>bm</i>	Pointer to the bit map structure
<i>prio</i>	Priority which will be marked as used

9.3.4.3 `static PORT_C_INLINE void prioBMClear (prioBM_T * bm, uint_fast8_t prio) [static]`

Clear the bit corresponding to the prio argument.

Parameters

<i>bm</i>	Pointer to the bit map structure
<i>prio</i>	Priority which will be marked as unused

9.3.4.4 `static PORT_C_INLINE uint_fast8_t prioBMGet (const prioBM_T * bm) [static]`

Get the highest priority set.

Parameters

<i>bm</i>	Pointer to the bit map structure
-----------	----------------------------------

Returns

The number of the highest priority marked as used

9.3.4.5 `static PORT_C_INLINE bool_T prioBMIsEmpty (const prioBM_T * bm) [static]`

Is bit map empty?

Parameters

<i>bm</i>	Pointer to the bit map structure
-----------	----------------------------------

Returns

The status of the bit map

Return values

<i>TRUE</i>	- bit map is empty
<i>FALSE</i>	- there is at least one bit set

9.3.4.6 `static PORT_C_INLINE void schedInit(void) [static]`

Initialize Ready Thread Queue structure [gRdyQueue](#) and Kernel control structure [esKernCtrl](#).

9.3.4.7 `static PORT_C_INLINE void schedStart(void) [static]`

Set the scheduler data structures for multi-threading.

This function is called just before multi-threading will start.

9.3.4.8 `static PORT_C_INLINE void schedSleep(void) [static]`

Set the scheduler to sleep.

Note

This function is used only when [CFG_SCHED_POWER_SAVE](#) option is active.

9.3.4.9 `static PORT_C_INLINE void schedWakeUpI(void) [static]`

Wake up the scheduler.

Note

This function is used only when [CFG_SCHED_POWER_SAVE](#) option is active.

9.3.4.10 `static PORT_C_INLINE void schedRdyAddInitI(esThd_T * thd) [static]`

Initialize scheduler ready structure during the thread add operation.

Parameters

<i>thd</i>	Pointer to the thread currently being initialized.
------------	--

Function will initialize scheduler structures during the init phase of the kernel.

9.3.4.11 `static PORT_C_INLINE void schedQmNextI(void) [static]`

Fetch and try to schedule the next thread of the same priority as the current thread.

9.3.4.12 `static PORT_C_INLINE void schedQmI(void) [static]`

Do the Quantum (Round-Robin) scheduling.

9.3.4.13 `static PORT_C_INLINE void sysTmrInit(void) [static]`

Initialize system timer hardware.

9.3.4.14 `static PORT_C_INLINE void sysTmrActivate(void) [static]`

Try to activate system timer.

Note

This function is used only when [CFG_SYSTMTR_ADAPTIVE_MODE](#) option is active.

9.3.4.15 `static PORT_C_INLINE void sysTmrDeactivateI(void) [static]`

Try to deactivate system timer.

Note

This function is used only when [CFG_SYSTMTR_ADAPTIVE_MODE](#) option is active.

9.3.4.16 `static PORT_C_INLINE void vTmrSleep (esTick_T ticks) [static]`

Set up system timer for different tick period during sleeping.

Parameters

<i>ticks</i>	Number of ticks to sleep
--------------	--------------------------

Note

This function is used only when [CFG_SYSTMTR_ADAPTIVE_MODE](#) option is active.

9.3.4.17 `static PORT_C_INLINE void vTmrEvaluate1 (void) [static]`

Evaluate armed virtual timers.

9.3.4.18 `static void vTmrAddArmedS (esVTmr_T * vTmr) [static]`

Add a virtual timer into sorted list.

Parameters

<i>vTmr</i>	Virtual timer: pointer to virtual timer to add
-------------	--

9.3.4.19 `static PORT_C_INLINE void vTmrImportPend1 (void) [static]`

Import timers from pending list to armed list.

Note

This function is used only when [CFG_SYSTMTR_ADAPTIVE_MODE](#) option is active.

9.3.4.20 `static void vTmrImportPend (void) [static]`

Import timers from pending list to armed list.

9.3.4.21 `static void kVTmrInit (void) [static]`

Initialization of Virtual Timer kernel thread.

9.3.4.22 `static void kVTmr (void * arg) [static]`

Virtual Timer thread code.

Parameters

<i>arg</i>	Argument: thread does not use argument
------------	--

This thread is responsible for virtual timer callback invocation and to import pending timers into armed linked list.

9.3.4.23 `static void kIdleInit (void) [static]`

Initialization of Idle thread.

9.3.4.24 static void kldle (void * *arg*) [static]

Idle thread code.

Parameters

<i>arg</i>	Argument: thread does not use argument
------------	--

9.3.4.25 void esKernInit (void)

Initialize kernel internal data structures.

Precondition

- 1) The `kernel state == ES_KERN_INACTIVE`, see [Kernel states](#).

Postcondition

- 1) The `kernel state == ES_KERN_INIT`.

Note

- 1) This function may be invoked only once.

This function must be called first before any other kernel API. It initializes internal data structures that are used by other API functions.

Object class:

Regular **API** object, this object is part of the application programming interface.

9.3.4.26 PORT_C_NORETURN void esKernStart (void)

Start the multi-threading.

Precondition

- 1) The `kernel state == ES_KERN_INIT`, see [Kernel states](#).

Postcondition

- 1) The `kernel state == ES_KERN_RUN`
- 2) The multi-threading execution will commence.

Note

- 1) Once this function is called the execution of threads will start and this function will never return.

This function will start multi-threading. Once the multi-threading has started the execution will never return to this function again (this function never returns).

Object class:

Regular **API** object, this object is part of the application programming interface.

9.3.4.27 void esKernSysTmr (void)

Process the system timer event.

Precondition

- 1) The kernel state < ES_KERN_INIT, see [Kernel states](#).

This function will be called only by port system timer interrupt.

Object class:

Not API object, this object is not part of the application programming interface and it is intended for internal use only.

9.3.4.28 void esKernIsrPrologue() (void)

Enter Interrupt Service Routine.

Precondition

- 1) The kernel state < ES_KERN_INIT, see [Kernel states](#).

Note

- 1) You must call [esKernIsrEpilogue\(\)](#) at the exit of ISR.
- 2) You must invoke [esKernIsrPrologue\(\)](#) and [esKernIsrEpilogue\(\)](#) in pair. In other words, for every call to [esKernIsrPrologue\(\)](#) at the beginning of the ISR you must have a call to [esKernIsrEpilogue\(\)](#) at the end of the ISR.

Function will notify kernel that you are about to enter interrupt service routine (ISR). This allows kernel to keep track of interrupt nesting and then only perform rescheduling at the last nested ISR.

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.3.4.29 void esKernIsrEpilogue() (void)

Exit Interrupt Service Routine.

Precondition

- 1) The kernel state < ES_KERN_INIT, see [Kernel states](#).

Note

- 1) You must invoke [esKernIsrPrologue\(\)](#) and [esKernIsrEpilogue\(\)](#) in pair. In other words, for every call to [esKernIsrPrologue\(\)](#) at the beginning of the ISR you must have a call to [esKernIsrEpilogue\(\)](#) at the end of the ISR.
- 2) Rescheduling is prevented when the scheduler is locked (see [esSchedLockEnter\(\)](#))

This function is used to notify kernel that you have completed servicing an interrupt. When the last nested ISR has completed, the function will call the scheduler to determine whether a new, high-priority task, is ready to run.

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.3.4.30 `void esThdInit (esThd_T * thd, void(*)(void *) fn, void * arg, portStck_T * stck, size_t stckSize, uint8_t prio)`

Initialize the specified thread.

Parameters

<i>thd</i>	Thread: is a pointer to the thread structure, esThd . The structure will be used as information container for the thread. It is assumed that storage for the esThd structure is allocated by the user code.
<i>fn</i>	Function: is a pointer to thread function. Thread function must have the following signature: <code>void thread (void * arg)</code> .
<i>arg</i>	Argument: is a void pointer to an optional data area. It's usage is application defined and it is intended to pass arguments to thread when it is started for the first time.
<i>stck</i>	Stack: is a pointer to a allocated memory for thread stack. The pointer always points to the first element in the array, regardless of what type of stack the CPU is using. The thread's stack is used to store local variables, function parameters, return addresses. Each thread has its own stack and different sized stack. The stack type must be an array of portStck .
<i>stckSize</i>	Stack Size: specifies the size of allocated stack memory. Size is expressed in bytes. Please see port documentation about minimal stack size. Usage of C unary operator <code>sizeof</code> is the recommended way of specifying stack size.
<i>prio</i>	Priority: is the priority of the thread. The higher the number, the higher the priority (the importance) of the thread. Several threads can have the same priority. Note that lowest (0) and highest (<code>CFG_SCHED_PRIO_LVL - 1</code>) levels are reserved for kernel threads only.

Precondition

- 1) The kernel state `ES_KERN_INACTIVE`, see [Kernel states](#).
- 2) `thd != NULL`
- 3) `thd->signature != THD_CONTRACT_SIGNATURE`, the thread structure can't be initialized more than once.
- 4) `fn != NULL`
- 5) `stckSize >= PORT_STCK_MINSIZE_VAL`, see [PORT_STCK_MINSIZE_VAL](#).
- 6) `0 < prio < CFG_SCHED_PRIO_LVL - 1`, see [CFG_SCHED_PRIO_LVL](#).

Postcondition

- 1) `thd->signature == THD_CONTRACT_SIGNATURE`, each [esThd](#) structure will have valid signature after initialization.

Threads must be created in order for kernel to recognize them as threads. Initialize a thread by calling [esThd-Init\(\)](#) and provide arguments specifying to kernel how the thread will be managed. Threads are always created in the `ready-to-run` state. Threads can be created either prior to the start of multi-threading (before calling [esKernStart\(\)](#)), or by a running thread.

Object class:

Regular **API** object, this object is part of the application programming interface.

9.3.4.31 `void esThdTerm (esThd_T * thd)`

Terminate the specified thread.

Parameters

<i>thd</i>	Thread: is a pointer to the thread structure, esThd .
------------	---

Precondition

- 1) The kernel state `ES_KERN_INACTIVE`, see [Kernel states](#).
- 2) `thd != NULL`
- 3) `thd->signature == THD_CONTRACT_SIGNATURE`, the pointer must point to a valid [esThd](#) structure.
- 4) `(thd->thdL.q == NULL) OR (thd->thdL.q == gRdyQueue)`, thread must be either in Ready Threads Queue or not be in any queue (e.g. not waiting for a synchronization mechanism).

Postcondition

- 1) `thd->signature == ~THD_CONTRACT_SIGNATURE`, each [esThd](#) structure will have invalid signature after termination.

Object class:

Regular **API** object, this object is part of the application programming interface.

9.3.4.32 void esThdSetPriol (esThd_T * thd, uint8_t prio)

Set the priority of a thread.

Parameters

<i>thd</i>	Thread: is pointer to the thread structure, esThd .
<i>prio</i>	Priority: is new priority of the thread pointed by <i>thd</i> .

Precondition

- 1) The kernel state `< ES_KERN_INACTIVE`, see [Kernel states](#).
- 2) `thd != NULL`
- 3) `thd->signature == THD_CONTRACT_SIGNATURE`, the pointer must point to a valid [esThd](#) structure.
- 4) `0 < prio < CFG_SCHED_PRIO_LVL - 1`, see [CFG_SCHED_PRIO_LVL](#).

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.3.4.33 void esThdPostl (esThd_T * thd)

Post to thread semaphore.

Parameters

<i>thd</i>	Pointer to the thread ID structure
------------	------------------------------------

Precondition

- 1) The kernel state `< ES_KERN_INACTIVE`, see [Kernel states](#).
- 2) `thd != NULL`
- 3) `thd->signature == THD_CONTRACT_SIGNATURE`, the pointer must point to a valid [esThd](#) structure.

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.3.4.34 void esThdPost (esThd_T * thd)

Post to thread semaphore.

Parameters

<i>thd</i>	Pointer to the thread ID structure
------------	------------------------------------

Precondition

- 1) The kernel state < ES_KERN_INACTIVE, see [Kernel states](#).
- 2) thd != NULL
- 3) thd->signature == THD_CONTRACT_SIGNATURE, the pointer must point to a valid [esThd](#) structure.

Object class:

Regular **API** object, this object is part of the application programming interface.

9.3.4.35 void esThdWaitl (void)

Wait for thread semaphore.

Precondition

- 1) The kernel state == ES_KERN_RUN, see [Kernel states](#).

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.3.4.36 void esThdWait (void)

Wait for thread semaphore.

Precondition

- 1) The kernel state == ES_KERN_RUN, see [Kernel states](#).

Object class:

Regular **API** object, this object is part of the application programming interface.

9.3.4.37 void esThdQInit (esThdQ_T * thdQ)

Initialize Thread Queue.

Parameters

<i>thdQ</i>	Thread Queue: is a pointer to thread queue structure, esThdQ .
-------------	--

Precondition

- 1) thdQ != NULL
- 2) thdQ->signature != THDQ_CONTRACT_SIGNATURE, the thread queue structure can't be initialized more than once.

Postcondition

1) `thdQ->signature == THDQ_CONTRACT_SIGNATURE`, each `esThdQ` structure will have valid signature after initialization.

Object class:

Regular **API** object, this object is part of the application programming interface.

9.3.4.38 void esThdQTerm (esThdQ_T * thdQ)

Terminate Thread Queue.

Parameters

<i>thdQ</i>	Thread Queue: is a pointer to thread queue structure, <code>esThdQ</code> .
-------------	---

Precondition

1) `thdQ != NULL`
 2) `thdQ->signature == THDQ_CONTRACT_SIGNATURE`, the thread queue structure must be already initialized.

Postcondition

1) `thdQ->signature == ~THDQ_CONTRACT_SIGNATURE`, each `esThdQ` structure will have invalid signature after termination.

Object class:

Regular **API** object, this object is part of the application programming interface.

9.3.4.39 void esThdQAddl (esThdQ_T * thdQ, esThd_T * thd)

Add a thread to the Thread Queue.

Parameters

<i>thdQ</i>	Thread Queue: is a pointer to thread queue structure, <code>esThdQ</code> .
<i>thd</i>	Thread: is a pointer to the thread ID structure, <code>esThd</code> .

Precondition

1) `thdQ != NULL`
 2) `thdQ->signature == THDQ_CONTRACT_SIGNATURE`, the pointer must point to a valid `esThdQ` structure.
 3) `thd != NULL`
 4) `thd->signature == THD_CONTRACT_SIGNATURE`, the pointer must point to a valid `esThd` structure.
 5) `thd->thdL.q == NULL`, thread must not be in any queue.

This function adds a thread at the specified Thread Queue.

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.3.4.40 void esThdQRml (esThdQ_T * thdQ, esThd_T * thd)

Removes the thread from the Thread Queue.

Parameters

<i>thdQ</i>	Thread Queue: is a pointer to thread queue structure, esThdQ .
<i>thd</i>	Thread: is a pointer to the thread ID structure, esThd .

Precondition

- 1) *thd* != NULL
- 2) *thd*->signature == THD_CONTRACT_SIGNATURE, the pointer must point to a valid [esThd](#) structure.
- 3) *thdQ* != NULL
- 4) *thdQ*->signature == THDQ_CONTRACT_SIGNATURE, the pointer must point to a valid [esThdQ](#) structure.
- 5) *thd*->thdL.q == *thdQ*, thread must be in the *thdQ* queue.

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.3.4.41 esThd_T* esThdQFetchl (const esThdQ_T * thdQ)

Fetch the first high priority thread from the Thread Queue.

Parameters

<i>thdQ</i>	Thread Queue: is a pointer to thread queue structure, esThdQ .
-------------	--

Returns

A pointer to the thread ID structure with the highest priority.

Precondition

- 1) *thdQ* != NULL
- 2) *thdQ*->signature == THDQ_CONTRACT_SIGNATURE, the pointer must point to a valid [esThdQ](#) structure.
- 3) *prioBM* != 0, priority bit map must not be empty

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.3.4.42 esThd_T* esThdQFetchRotatel (esThdQ_T * thdQ, uint_fast8_t prio)

Fetch the next thread and rotate thread linked list.

Parameters

<i>thdQ</i>	Thread Queue: is a pointer to thread queue structure, esThdQ . This is the thread queue to fetch from.
<i>prio</i>	Priority: is the priority level to fetch and rotate.

Returns

Pointer to the next thread in queue.

Precondition

- 1) `thdQ != NULL`
- 2) `thdQ->signature == THDQ_CONTRACT_SIGNATURE`, the pointer must point to a valid [esThdQ](#) structure.
- 3) `0 <= prio <= CFG_SCHED_PRIO_LVL`, see [CFG_SCHED_PRIO_LVL](#).
- 4) `sentinel != NULL`, at least one thread must be in the selected priority level

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.3.4.43 `bool_T esThdQIsEmpty (const esThdQ_T * thdQ)`

Is thread queue empty.

Parameters

<i>thdQ</i>	Thread Queue: is a pointer to thread queue structure, esThdQ .
-------------	--

Returns

The state of thread queue

Return values

<i>TRUE</i>	- thread queue is empty
<i>FALSE</i>	- thread queue is not empty

Precondition

- 1) `thdQ != NULL`
- 2) `thdQ->signature == THDQ_CONTRACT_SIGNATURE`, the pointer must point to a valid [esThdQ](#) structure.

Object class:

Regular **API** object, this object is part of the application programming interface.

9.3.4.44 `void esSchedRdyAddl (esThd_T * thd)`

Add thread `thd` to the ready thread list and notify the scheduler.

Parameters

<i>thd</i>	Pointer to the initialized thread ID structure, esThd .
------------	---

Precondition

- 1) The kernel state `< ES_KERN_INACTIVE`, see [Kernel states](#).
- 2) `thd != NULL`
- 3) `thd->signature == THD_CONTRACT_SIGNATURE`, the pointer must point to a valid [esThd](#) structure.
- 4) `thd->thdL.q == NULL`, thread must not be in a queue.

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.3.4.45 void esSchedRdyRml (esThd_T * thd)

Remove thread `thd` from the ready thread list and notify the scheduler.

Parameters

<code>thd</code>	Pointer to the initialized thread ID structure, esThd .
------------------	---

Precondition

- 1) The kernel state < ES_KERN_INACTIVE, see [Kernel states](#).
- 2) `thd != NULL`
- 3) `thd->signature == THD_CONTRACT_SIGNATURE`, the pointer must point to a valid [esThd](#) structure.
- 4) `thd->thdL.q == &gRdyQueue`, thread must be in Ready Threads queue.

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.3.4.46 void esSchedYieldl (void)

Force the scheduler invocation which will evaluate all ready threads and switch to ready thread with the highest priority.

Precondition

- 1) The kernel state < ES_KERN_INACTIVE, see [Kernel states](#).

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.3.4.47 void esSchedYieldlsl (void)

Force the scheduler invocation which will evaluate all ready threads and switch to ready thread with the highest priority.

Precondition

- 1) The kernel state < ES_KERN_INACTIVE, see [Kernel states](#).

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.3.4.48 void esSchedLockEnterl (void)

Lock the scheduler.

Precondition

- 1) The `kernel state < ES_KERN_INIT`, see [Kernel states](#).

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.3.4.49 void esSchedLockExitl (void)

Unlock the scheduler.

Precondition

- 1) The `kernel state < ES_KERN_INIT`, see [Kernel states](#).
- 2) `gKernLockCnt > 0U`, current number of locks must be greater than zero, in other words: each call to kernel lock function must have its matching call to kernel unlock function.

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.3.4.50 void esSchedLockEnter (void)

Lock the scheduler.

Precondition

- 1) The `kernel state < ES_KERN_INIT`, see [Kernel states](#).

Object class:

Regular **API** object, this object is part of the application programming interface.

9.3.4.51 void esSchedLockExit (void)

Unlock the scheduler.

Precondition

- 1) The `kernel state < ES_KERN_INIT`, see [Kernel states](#).
- 2) `gKernLockCnt > 0U`, current number of locks must be greater than zero, in other words: each call to kernel lock function must have its matching call to kernel unlock function.

Object class:

Regular **API** object, this object is part of the application programming interface.

9.3.4.52 void esVTmrInitl (esVTmr_T * vTmr, esTick_T tick, void(*)(void *) fn, void * arg)

Add and start a new virtual timer.

Parameters

<i>vTmr</i>	Virtual Timer: is pointer to the timer ID structure, esVTmr .
<i>tick</i>	Tick: the timer delay expressed in system ticks
<i>fn</i>	Function: is pointer to the callback function
<i>arg</i>	Argument: is pointer to the arguments of callback function

Precondition

- 1) The kernel state < ES_KERN_INACTIVE, see [Kernel states](#).
- 2) vTmr != NULL
- 3) vTmr->signature != VTMR_CONTRACT_SIGNATURE, the timer structure can't be initialized more than once.
- 4) tick > 1U
- 5) fn != NULL

Postcondition

- 1) vTmr->signature == VTMR_CONTRACT_SIGNATURE, each [esVTmr](#) structure will have valid signature after initialization.

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.3.4.53 void esVTmrInit (esVTmr_T * vTmr, esTick_T tick, void(*) (void *) fn, void * arg)

Add and start a new virtual timer.

Parameters

<i>vTmr</i>	Virtual Timer: is pointer to the timer ID structure, esVTmr .
<i>tick</i>	Tick: the timer delay expressed in system ticks
<i>fn</i>	Function: is pointer to the callback function
<i>arg</i>	Argument: is pointer to the arguments of callback function

Precondition

- 1) The kernel state < ES_KERN_INACTIVE, see [Kernel states](#).
- 2) vTmr != NULL
- 3) vTmr->signature != VTMR_CONTRACT_SIGNATURE, the timer structure can't be initialized more than once.
- 4) tick > 1U
- 5) fn != NULL

Postcondition

- 1) vTmr->signature == VTMR_CONTRACT_SIGNATURE, each [esVTmr](#) structure will have valid signature after initialization.

Object class:

Regular **API** object, this object is part of the application programming interface.

9.3.4.54 void esVTmrTermI (esVTmr_T * vTmr)

Cancel and remove a virtual timer.

Parameters

<i>vTmr</i>	Timer: is pointer to the timer ID structure, esVTmr .
-------------	---

Precondition

- 1) The kernel state < ES_KERN_INACTIVE, see [Kernel states](#).
- 2) vTmr != NULL
- 3) vTmr->signature == VTMR_CONTRACT_SIGNATURE, the pointer must point to a valid [esVTmr](#) structure.

Postcondition

- 1) vTmr->signature = ~VTMR_CONTRACT_SIGNATURE, each [esVTmr](#) structure will have invalid signature after termination.

Function class:

I class API function, this function can be called from application and interrupt service routine only with interrupts locked.

9.3.4.55 void esVTmrTerm (esVTmr_T * vTmr)

Cancel and remove a virtual timer.

Parameters

<i>vTmr</i>	Timer: is pointer to the timer ID structure, esVTmr .
-------------	---

Precondition

- 1) The kernel state < ES_KERN_INACTIVE, see [Kernel states](#).
- 2) vTmr != NULL
- 3) vTmr->signature == VTMR_CONTRACT_SIGNATURE, the pointer must point to a valid [esVTmr](#) structure.

Postcondition

- 1) vTmr->signature = ~VTMR_CONTRACT_SIGNATURE, each [esVTmr](#) structure will have invalid signature after termination.

Object class:

Regular **API** object, this object is part of the application programming interface.

9.3.4.56 void esVTmrDelay (esTick_T tick)

Delay for specified amount of ticks.

Parameters

<i>tick</i>	Tick: number of system ticks to delay.
-------------	--

This function will create a virtual timer with count down time specified in argument *tick* and put the calling thread into *sleep* state. When timeout expires the thread will be placed back into *ready* state.

Precondition

- 1) *tick* > 1U

Object class:

Regular **API** object, this object is part of the application programming interface.

9.3.5 Variable Documentation

9.3.5.1 `esThdQ_T gRdyQueue` [static]

Ready Thread queue.

9.3.5.2 `sysTmr_T gSysTmr` [static]

Initial value:

```
= {
    OU,
    OU,

}
```

Main System Timer structure.

9.3.5.3 `esVTmr_T gVTmrArmed` [static]

Initial value:

```
= {
    {
        &gVTmrArmed,
        &gVTmrArmed,
        &gVTmrArmed
    },

    UINT_FAST8_MAX,

    NULL,
    NULL,

}
```

List of virtual timers to armed expire.

9.3.5.4 `esVTmr_T gVTmrPend` [static]

Initial value:

```
= {
    {
        &gVTmrPend,
        &gVTmrPend,
        &gVTmrPend
    },
    OU,
    NULL,
    NULL,

}
```

Virtual timers pending to be inserted into waiting list.

9.3.5.5 `esThd_T gKVtmr` [static]

Virtual timer thread ID.

9.3.5.6 `esThd_T gKIdle` [static]

Idle thread ID.

9.3.5.7 `uint_fast8_t gKernLockCnt` `[static]`

Kernel Lock Counter.

9.3.5.8 `const volatile esKernCtrl_T gKernCtrl`

Initial value:

```
= {  
    NULL,  
    NULL,  
    ES_KERN_INACTIVE  
}
```

Kernel control initialization.

Kernel control block.

9.4 Configuration

Configuration settings.

Collaboration diagram for Configuration:



Kernel configuration options and settings

- `#define CFG_SCHED_PRIO_LVL 8U`
Scheduler priority levels.
- `#define CFG_SCHED_TIME_QUANTUM 10U`
Scheduler Round-Robin time quantum.
- `#define CFG_SCHED_POWER_SAVE 0U`
Enable/disable scheduler power savings mode.
- `#define CFG_SYSTMTR_ADAPTIVE_MODE 0U`
System timer adaptive mode.
- `#define CFG_SYSTMTR_EVENT_FREQUENCY 100UL`
The frequency of system timer tick event.
- `#define CFG_SYSTMTR_TICK_TYPE 2U`
The size of the system timer tick event counter.

Kernel pre hooks

- `#define CFG_HOOK_PRE_SYSTMTR_EVENT 0U`
System timer event hook function.
- `#define CFG_HOOK_PRE_KERN_INIT 0U`
Pre kernel initialization hook function.
- `#define CFG_HOOK_POST_KERN_INIT 0U`
Post kernel initialization hook function.
- `#define CFG_HOOK_PRE_KERN_START 0U`
Pre kernel start hook function.
- `#define CFG_HOOK_POST_THD_INIT 0U`
Post thread initialization hook function.
- `#define CFG_HOOK_PRE_THD_TERM 0U`
Pre thread termination hook function.
- `#define CFG_HOOK_PRE_IDLE 0U`
Pre idle hook function.
- `#define CFG_HOOK_POST_IDLE 0U`
Post idle hook function.
- `#define CFG_HOOK_PRE_CTX_SW 0U`
Pre context switch hook function.

9.4.1 Detailed Description

Configuration settings.

9.4.2 Macro Definition Documentation

9.4.2.1 `#define CFG_SCHED_PRIO_LVL 8U`

Scheduler priority levels.

Possible values:

- Min: 3U (three priority levels)
- Max: 256U

9.4.2.2 `#define CFG_SCHED_TIME_QUANTUM 10U`

Scheduler Round-Robin time quantum.

9.4.2.3 `#define CFG_SCHED_POWER_SAVE 0U`

Enable/disable scheduler power savings mode.

Possible values are:

- 0U - power saving is disabled
- 1U - power saving is enabled

9.4.2.4 `#define CFG_SYSTMTR_ADAPTIVE_MODE 0U`

System timer adaptive mode.

Possible values are:

- 0U - adaptive mode is disabled
- 1U - adaptive mode is enabled

9.4.2.5 `#define CFG_SYSTMTR_EVENT_FREQUENCY 100UL`

The frequency of system timer tick event.

Note

This setting is valid only if configuration option [CFG_SYSTMTR_CLOCK_FREQUENCY](#) is properly set in port configuration file [cpu_cfg.h](#)

9.4.2.6 `#define CFG_SYSTMTR_TICK_TYPE 2U`

The size of the system timer tick event counter.

Possible values are:

- 0U - 8 bit counter
- 1U - 16 bit counter
- 2U - 32 bit counter

9.4.2.7 `#define CFG_HOOK_PRE_SYSTMV_EVENT 0U`

System timer event hook function.

9.4.2.8 `#define CFG_HOOK_PRE_KERN_INIT 0U`

Pre kernel initialization hook function.

9.4.2.9 `#define CFG_HOOK_POST_KERN_INIT 0U`

Post kernel initialization hook function.

9.4.2.10 `#define CFG_HOOK_PRE_KERN_START 0U`

Pre kernel start hook function.

9.4.2.11 `#define CFG_HOOK_POST_THD_INIT 0U`

Post thread initialization hook function.

9.4.2.12 `#define CFG_HOOK_PRE_THD_TERM 0U`

Pre thread termination hook function.

9.4.2.13 `#define CFG_HOOK_PRE_IDLE 0U`

Pre idle hook function.

9.4.2.14 `#define CFG_HOOK_POST_IDLE 0U`

Post idle hook function.

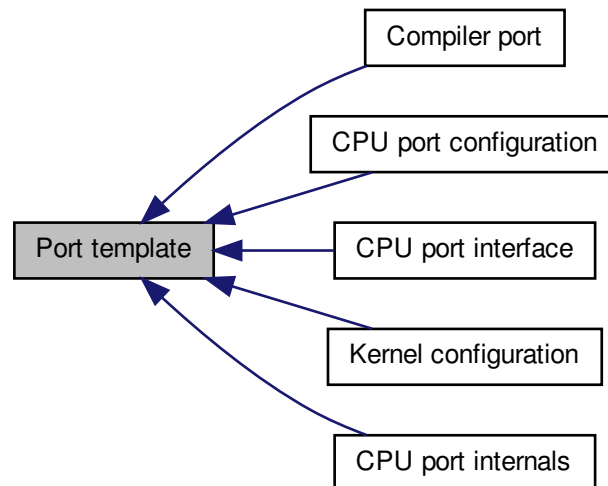
9.4.2.15 `#define CFG_HOOK_PRE_CTX_SW 0U`

Pre context switch hook function.

9.5 Port template

Templates.

Collaboration diagram for Port template:



Modules

- [CPU port configuration](#)
CPU port specific configuration options.
- [CPU port interface](#)
CPU port macros and functions.
- [CPU port internals](#)
CPU port inner work.
- [Compiler port](#)
Compiler provided macros and data types.
- [Kernel configuration](#)
Default Kernel configuration options.

9.5.1 Detailed Description

Templates.

9.6 Compiler port

Compiler provided macros and data types.

Collaboration diagram for Compiler port:



Compiler provided macros

Port interface macros and port specific macros

These macros are used to ease the writing of ports. All macros prefixed with **PORT_** are part of the port interface.

- #define **PORT_C_INLINE** inline
C extension - make a function inline.
- #define **PORT_C_INLINE_ALWAYS** inline
C extension - make a function inline - always.
- #define **PORT_C_NAKED**
Omit function prologue/epilogue sequences.
- #define **PORT_C_FUNC** "unknown"
Provides function name for assert macros.
- #define **PORT_C_WEAK**
Declares a weak function.
- #define **PORT_C_ALIGNED**(expr)
This attribute specifies a minimum alignment (in bytes) for variables of the specified type.
- #define **PORT_HWREG_SET**(reg, mask, val)
A standardized way of properly setting the value of HW register.

Compiler provided data types

The compiler port must provide some C90 (C99) data types

The compiler port must:

- declare sets of integer types having specified widths, standard type definitions and shall define corresponding sets of macros.

Types are defined in the following categories:

- Integer types having certain exact widths
- Fastest integer types having at least certain specified widths
- Integer types wide enough to hold pointers to objects
- standard type definitions

The following exact-width integer types are required:

- `int8_t`
- `int16_t`
- `int32_t`
- `uint8_t`
- `uint16_t`
- `uint32_t`

The following fastest minimum-width integer types are required:

- `int_fast8_t`
- `int_fast16_t`
- `int_fast32_t`
- `uint_fast8_t`
- `uint_fast16_t`
- `uint_fast32_t`

The following integer types capable of holding object pointers are required:

- `intptr_t`
- `uintptr_t`

The following standard type definitions are required:

- `NULL`
- `ptrdiff_t`
- `size_t`
- `enum boolType {
 TRUE = 1U,
 FALSE = 0U }`
 Bool data type.
- `typedef enum boolType bool_T`
 Bool data type.

9.6.1 Detailed Description

Compiler provided macros and data types.

9.6.2 Macro Definition Documentation

9.6.2.1 `#define PORT_C_INLINE` inline

C extension - make a function inline.

The point of making a function `inline` is to hint to the compiler that it is worth making some form of extra effort to call the function faster than it would otherwise - generally by substituting the code of the function into its caller. As well as eliminating the need for a call and return sequence, it might allow the compiler to perform certain optimizations between the bodies of both functions.

9.6.2.2 #define PORT_C_INLINE_ALWAYS inline

C extension - make a function inline - always.

Generally, functions are not inlined unless optimization is specified. For functions declared inline, this attribute inlines the function even if no optimization level was specified.

9.6.2.3 #define PORT_C_NAKED

Omit function prologue/epilogue sequences.

This attribute will indicate that the specified function does not need prologue/epilogue sequences generated by the compiler. It is up to the programmer to provide these sequences. The only statements that can be safely included in naked functions are `asm` statements that do not have operands. All other statements, including declarations of local variables, `if` statements, and so forth, should be avoided. Naked functions should be used to implement the body of an assembly function, while allowing the compiler to construct the requisite function declaration for the assembler.

9.6.2.4 #define PORT_C_FUNC "unknown"

Provides function name for assert macros.

9.6.2.5 #define PORT_C_WEAK

Declares a weak function.

The weak attribute causes the declaration to be emitted as a weak symbol rather than a global. This is primarily useful in defining library functions that can be overridden in user code, though it can also be used with non-function declarations.

9.6.2.6 #define PORT_C_ALIGNED(*expr*)

This attribute specifies a minimum alignment (in bytes) for variables of the specified type.

Note

The alignment of any given struct or union type is required by the ISO C standard to be at least a perfect multiple of the lowest common multiple of the alignments of all of the members of the struct or union in question.

9.6.2.7 #define PORT_HWREG_SET(*reg*, *mask*, *val*)

Value:

```
do {
    portReg_T tmp;
    tmp = (reg);
    tmp &= ~(mask);
    tmp |= ((mask) & (val));
    (reg) = tmp;
} while (0U)
```

\\
\\
\\
\\

A standardized way of properly setting the value of HW register.

Parameters

<i>reg</i>	Register which will be written to
<i>mask</i>	The bit mask which will be applied to register and <i>val</i> argument
<i>val</i>	Value to be written into the register

9.6.3 Typedef Documentation

9.6.3.1 typedef enum boolType bool_T

Bool data type.

9.6.4 Enumeration Type Documentation

9.6.4.1 enum boolType

Bool data type.

Enumerator

TRUE TRUE. TRUE

FALSE FALSE. FALSE

9.7 CPU port interface

CPU port macros and functions.

Collaboration diagram for CPU port interface:



Data Structures

- struct `portStck`
Stack structure used for stack declaration in order to force the alignment Alignment of stack structure.
- struct `portCtx`
Port context structure.

Typedefs

- typedef uint8_t `portReg_T`
Data type which corresponds to the general purpose register.
- typedef struct `portStck` `portStck_T`
Stack type.

Variables

- `portReg_T gPortIsrNesting_`
Variable to keep track of ISR nesting.
- const PORT_C_ROM `portReg_T pwr2LKP [PORT_DATA_WIDTH_VAL]`
Look up table for: 2^n expression.

Port constants

- #define `PORT_DATA_WIDTH_VAL` 8U
This macro specifies the bit width of CPU data registers.
- #define `PORT_STCK_MINSIZE_VAL` sizeof(struct `portCtx`)
This macro specifies the minimal size of the thread stack.
- #define `PORT_SYSTMRR_RELOAD_VAL` (CFG_SYSTMRR_CLOCK_FREQUENCY / CFG_SYSTMRR_EVENT_FREQUENCY)
System timer reload value for one tick.
- #define `PORT_SYSTMRR_MAX_VAL` 0xFFU
System timer maximum value.
- #define `PORT_SYSTMRR_MAX_TICKS_VAL` (PORT_SYSTMRR_MAX_VAL / PORT_SYSTMRR_RELOAD_VAL)
Maximum number of ticks the system timer can accept.

Interrupt management

- void `portIntDisable_` (void)
Disable interrupts.
- `portReg_T portIntGet_` (void)
Get the current status of enabled/disabled interrupts.
- void `portIntSet_` (`portReg_T` status)
Set the status of interrupts according to the `status` argument.
- #define `PORT_INT_DISABLE()` `portIntDisable_()`
Disable all interrupt sources.
- #define `PORT_ISR_ENTER()`
Enter ISR. Increment `gPortIsrNesting_` variable to keep track of ISR nesting.
- #define `PORT_ISR_EXIT()`
Exit ISR. Decrement `gPortIsrNesting_` variable to keep track of ISR nesting.
- #define `PORT_ISR_IS_LAST()` (0U == `gPortIsrNesting` ? TRUE : FALSE)
If `isrNesting` variable is zero then the last ISR is executing and scheduler should be invoked.

Critical section management

Disable/enable interrupts by preserving the status of interrupts.

Generally speaking these macros would store the status of the interrupt disable flag in the local variable declared by `PORT_CRITICAL_DECL` and then disable interrupts. Local variable is allocated in all of eSolid RTOS functions that need to disable interrupts. Macros would restore the interrupt status by copying back the allocated variable into the CPU's status register.

- #define `PORT_CRITICAL_DECL()` `portReg_T intStatus_`
Declare the interrupt status variable.
- #define `PORT_CRITICAL_ENTER()`
Enter critical section.
- #define `PORT_CRITICAL_EXIT()` `portIntSet_(intStatus_)`
Exit critical section.

Scheduler support

Note

These functions are extensively used by the scheduler and therefore they should be optimized for the architecture being used.

- `uint_fast8_t portFindLastSet_` (`portReg_T` val)
Find last set bit in a word.
- void `portSysTmrInit_` (void)
Initialize systick timer and associated interrupt.
- void `portSysTmrTerm_` (void)
Stop the sistem timer.
- void `portSysTmrReload_` (`esTick_T` ticks)
Reload the system timer.
- void `portSysTmrEnable_` (void)
Enable the system timer.
- void `portSysTmrDisable_` (void)
Disable the system timer.
- void `portSysTmrIsrEnable_` (void)

- *Disable the system timer interrupt.*
void `portSysTmrIsrDisable_` (void)
- *Enable the system timer interrupt.*
void `portThdStart_` (void)
- *Start the first thread.*
#define `PORT_FIND_LAST_SET`(val) `portFindLastSet_`(val)
Find last set bit in a word.
- #define `PORT_PWR2`(pwr) (1U << (pwr))
Helper macro: calculate 2^{pwr} expression.
- #define `PORT_SYSTMR_INIT`() `portSysTmrInit_`()
Initialize system timer and associated interrupt.
- #define `PORT_SYSTMR_TERM`() `portSysTmrTerm_`()
Stop the timer if it is running and disable associated interrupt.
- #define `PORT_SYSTMR_RELOAD`(ticks) `portSysTmrReload_`(ticks)
Reload the system timer with specified number of ticks.
- #define `PORT_SYSTMR_ENABLE`() `portSysTmrEnable_`()
Enable the system timer.
- #define `PORT_SYSTMR_DISABLE`() `portSysTmrDisable_`()
Disable the system timer.
- #define `PORT_SYSTMR_ISR_ENABLE`() `portSysTmrIsrEnable_`()
Enable the system timer interrupt.
- #define `PORT_SYSTMR_ISR_DISABLE`() `portSysTmrIsrDisable_`()
Disable the system timer interrupt.

Dispatcher context switching

- void * `portCtxInit_` (void *stck, size_t stckSize, void(*thdf)(void *), void *arg)
Initialize the thread context.
- void `portCtxSw_` (void)
Do the context switch - invoked from API.
- void `portCtxSwIsr_` (void)
Do the context switch - invoked from ISR.
- #define `PORT_CTX_INIT`(stck, stackSize, thread, arg) `portCtxInit_`(stck, stackSize, thread, arg)
Initialize the thread context.
- #define `PORT_CTX_SW`() `portCtxSw_`()
Do the context switch - invoked from API level.
- #define `PORT_CTX_SW_ISR`() `portCtxSwIsr_`()
Do the context switch - invoked from ISR level.
- #define `PORT_THD_START`() `portThdStart_`()
Start the first thread.

General port macros

- #define `PORT_STCK_SIZE`(size)
Calculate the stack size.
- #define `PORT_CRITICAL_EXIT_SLEEP`() `PORT_CRITICAL_EXIT`()
TODO.
- #define `PORT_INIT_EARLY`() `portInitEarly_`()
Early port initialization.
- #define `PORT_INIT`() `portInit_`()
Port initialization.
- #define `PORT_INIT_LATE`() `portInitLate_`()
Late port initialization.

General port functions

- void `portInitEarly_` (void)
Early port initialization.
- void `portInit_` (void)
Port initialization.
- void `portInitLate_` (void)
Late port initialization.

9.7.1 Detailed Description

CPU port macros and functions. Since this header file is included with the API of the kernel a few naming conventions are defined in order to avoid name clashing with the names of objects from libraries included by application code.

1) Macro naming conventions

For macro naming try to follow these rules:

- All standard PORT API macro names are prefixed with: **PORT_**.
- All other macros which are specific to the port used are prefixed with: **CPU_**.

2) Type declaration naming conventions

For type declaration naming try to follow these rules:

- All type declaration names are prefixed with: **cpu**.

3) Global variable naming conventions

For global variable naming try to follow these rules:

- All global variable names are prefixed with: **cpu**.

4) Function naming conventions

For functions naming try to follow these rules:

- All function names are prefixed with: **port** and postfixed with: **_** (underscore).
- All other functions which are specific to the port used are prefixed with: **cpu** and postfixed with: **_** (underscore).
- The `exception` to above two rules are the names of functions used for Interrupt Service Routines. They can have any name required by port.

9.7.2 Macro Definition Documentation

9.7.2.1 `#define PORT_DATA_WIDTH_VAL 8U`

This macro specifies the bit width of CPU data registers.

9.7.2.2 `#define PORT_STCK_MINSIZE_VAL sizeof(struct portCtx)`

This macro specifies the minimal size of the thread stack.

Generally minimal stack size is equal to the size of context structure

9.7.2.3 `#define PORT_SYSTMTR_RELOAD_VAL (CFG_SYSTMTR_CLOCK_FREQUENCY / CFG_SYSTMTR_EVENT_FREQUENCY)`

System timer reload value for one tick.

This is a calculated value for one system tick period

9.7.2.4 #define PORT_SYSTMTR_MAX_VAL 0xFFU

System timer maximum value.

This macro specifies maximum value that can be reloaded into system timer counter. For example, if the system timer is a 8-bit counter than this macro would have the value of 0xFFU.

9.7.2.5 #define PORT_SYSTMTR_MAX_TICKS_VAL (PORT_SYSTMTR_MAX_VAL / PORT_SYSTMTR_RELOAD_VAL)

Maximum number of ticks the system timer can accept.

9.7.2.6 #define PORT_INT_DISABLE() portIntDisable_()

Disable all interrupt sources.

9.7.2.7 #define PORT_ISR_ENTER()

Value:

```
do {
    gPortIsrNesting++;
    esKernIsrPrologueI();
} while (0U)
```

Enter ISR. Increment gPortIsrNesting_ variable to keep track of ISR nesting.

Variable gPortIsrNesting_ is needed only if the port does not support any other method of detecting when the last ISR is executing.

9.7.2.8 #define PORT_ISR_EXIT()

Value:

```
do {
    gPortIsrNesting--;
    esKernIsrEpilogueI();
} while (0U)
```

Exit ISR. Decrement gPortIsrNesting_ variable to keep track of ISR nesting.

Variable gPortIsrNesting_ is needed only if the port does not support any other method of detecting when the last ISR is executing.

9.7.2.9 #define PORT_ISR_IS_LAST() (0U == gPortIsrNesting ? TRUE : FALSE)

If isrNesting variable is zero then the last ISR is executing and scheduler should be invoked.

Returns

Is the currently executed ISR the last one?

Return values

<i>TRUE</i>	- this is last ISR
<i>FALSE</i>	- this is not the last ISR

9.7.2.10 #define PORT_CRITICAL_DECL() portReg_T intStatus_

Declare the interrupt status variable.

This variable is used to store the current state of enabled ISRs.

9.7.2.11 `#define PORT_CRITICAL_ENTER()`**Value:**

```
do {
    intStatus_ = portIntGet_();
    portIntDisable_();
} while (0U)
```

Enter critical section.

9.7.2.12 `#define PORT_CRITICAL_EXIT() portIntSet_(intStatus_)`

Exit critical section.

9.7.2.13 `#define PORT_FIND_LAST_SET(val) portFindLastSet_(val)`

Find last set bit in a word.

This function is used by the scheduler to efficiently determine the highest priority of thread ready for execution. For algorithm details see: http://en.wikipedia.org/wiki/Find_first_set.

Returns

The position of the last set bit in a word

9.7.2.14 `#define PORT_PWR2(pwr) (1U << (pwr))`

Helper macro: calculate 2^{pwr} expression.

Some ports may want to use look up tables instead of shifting operation

9.7.2.15 `#define PORT_SYSTMTR_INIT() portSysTmrInit_()`

Initialize system timer and associated interrupt.

This macro will only initialize system timer and associated interrupt. It MUST NOT start the system timer in this stage. Responsibility:

- initialize system timer
- initialize system timer interrupt

9.7.2.16 `#define PORT_SYSTMTR_TERM() portSysTmrTerm_()`

Stop the timer if it is running and disable associated interrupt.

Responsibility:

- disable system timer interrupt
- stop and disable system timer

9.7.2.17 `#define PORT_SYSTMTR_RELOAD(ticks) portSysTmrReload_(ticks)`

Reload the system timer with specified number of ticks.

Responsibility:

- calculate the reload value based on `PORT_SYSTMTR_RELOAD_VAL`
- reload the system timer

9.7.2.18 #define PORT_SYSTMTR_ENABLE() portSysTmrEnable_()

Enable the system timer.

Responsibility:

- enable (run) the system timer counter

9.7.2.19 #define PORT_SYSTMTR_DISABLE() portSysTmrDisable_()

Disable the system timer.

Responsibility:

- disable (stop) the system timer counter

9.7.2.20 #define PORT_SYSTMTR_ISR_ENABLE() portSysTmrIsrEnable_()

Enable the system timer interrupt.

Responsibility:

- allow system timer interrupt to occur

9.7.2.21 #define PORT_SYSTMTR_ISR_DISABLE() portSysTmrIsrDisable_()

Disable the system timer interrupt.

Responsibility:

- disallow system timer interrupt to occur

9.7.2.22 #define PORT_CTX_INIT(*stck*, *stackSize*, *thread*, *arg*) portCtxInit_(stck, stackSize, thread, arg)

Initialize the thread context.

Parameters

<i>in, out</i>	<i>stck</i>	Pointer to the allocated thread stck. The pointer points to the beginning of the memory as defined per C language. It's up to port function to adjust the pointer according to the stck type: full descending or full ascending one.
	<i>stackSize</i>	The size of allocated stck in bytes.
<i>in</i>	<i>thread</i>	Pointer to the thread function.
<i>in</i>	<i>arg</i>	Argument that will be passed to thread function at the starting of execution.

Returns

The new top of stck after thread context initialization.

9.7.2.23 #define PORT_CTX_SW() portCtxSw_()

Do the context switch - invoked from API level.

9.7.2.24 #define PORT_CTX_SW_ISR() portCtxSwIsr_()

Do the context switch - invoked from ISR level.

9.7.2.25 #define PORT_THD_START() portThdStart_()

Start the first thread.

9.7.2.26 `#define PORT_STCK_SIZE(size)`

Value:

```
((((size + PORT_STCK_MINSIZE_VAL) + (sizeof(struct portStck) /
    sizeof(portReg_T)) - 1U) / (sizeof(struct portStck)/sizeof(
    portReg_T)))
```

Calculate the stack size.

This macro is used when specifying the size of thread stack. Responsibility:

- add to `size` the minimal stack size specified by `PORT_STCK_MINSIZE_VAL`.
- if it is needed by the port make sure the alignment is correct.

9.7.2.27 `#define PORT_CRITICAL_EXIT_SLEEP() PORT_CRITICAL_EXIT()`

TODO.

9.7.2.28 `#define PORT_INIT_EARLY() portInitEarly_()`

Early port initialization.

This macro will be called at early initialization stage from `esKernInit()` function. It is called before any kernel data initialization. Usually this macro would be used to setup memory space, fill the memory with debug value or something similar.

9.7.2.29 `#define PORT_INIT() portInit_()`

Port initialization.

This macro will be called after kernel data structure initialization from `esKernInit()` function.

9.7.2.30 `#define PORT_INIT_LATE() portInitLate_()`

Late port initialization.

This macro will be called just a moment before the multitasking is started. The macro is called from `esKernStart()` function.

9.7.3 Typedef Documentation

9.7.3.1 `typedef uint8_t portReg_T`

Data type which corresponds to the general purpose register.

9.7.3.2 `typedef struct portStck portStck_T`

Stack type.

9.7.4 Function Documentation

9.7.4.1 `void portIntDisable_ (void)`

Disable interrupts.

9.7.4.2 `portReg_T portIntGet_ (void)`

Get the current status of enabled/disabled interrupts.

Returns

Interrupt status

9.7.4.3 void portIntSet_(portReg_T status)

Set the status of interrupts according to the `status` argument.

Parameters

<i>status</i>	The status of interrupts that will be set by the function.
---------------	--

9.7.4.4 uint_fast8_t portFindLastSet_(portReg_T val)

Find last set bit in a word.

Parameters

<i>val</i>	Value which needs to be evaluated
------------	-----------------------------------

This function is used by the scheduler to efficiently determine the highest priority of thread ready for execution. For algorithm details see: http://en.wikipedia.org/wiki/Find_first_set.

Returns

The position of the last set bit in a word

9.7.4.5 void portSysTmrInit_(void)

Initialize systick timer and associated interrupt.

This function will be called just a moment before the multitasking is started. The function is called from `esKernStart()` function. It should setup:

- systick timer (scheduler uses tick event to switch between threads of same priority)
- systick timer interrupt

Note

This function MUST NOT enable system timer events. System timer events are enabled/disabled by `portSysTmrEnable_()` and `portSysTmrDisable_()` functions.

9.7.4.6 void portSysTmrTerm_(void)

Stop the system timer.

9.7.4.7 void portSysTmrReload_(esTick_T ticks)

Reload the system timer.

Parameters

<i>ticks</i>	How much ticks is needed to delay
--------------	-----------------------------------

9.7.4.8 void portSysTmrEnable_(void)

Enable the system timer.

9.7.4.9 void portSysTmrDisable_ (void)

Disable the system timer.

9.7.4.10 void portSysTmrIsrEnable_ (void)

Disable the system timer interrupt.

9.7.4.11 void portSysTmrIsrDisable_ (void)

Enable the system timer interrupt.

9.7.4.12 void portThdStart_ (void)

Start the first thread.

9.7.4.13 void* portCtxInit_ (void * *stck*, size_t *stckSize*, void(*)(void *) *thdf*, void * *arg*)

Initialize the thread context.

Parameters

<i>in, out</i>	<i>stck</i>	Pointer to the allocated thread stck. The pointer points to the beginning of the memory as defined per C language. It's up to port function to adjust the pointer according to the stck type: full descending or full ascending one.
	<i>stckSize</i>	The size of allocated stck in bytes.
<i>in</i>	<i>thdf</i>	Pointer to the thread function.
<i>in</i>	<i>arg</i>	Argument that will be passed to thread function at the starting of execution.

Returns

The new top of stck after thread context initialization.

9.7.4.14 void portCtxSw_ (void)

Do the context switch - invoked from API.

9.7.4.15 void portCtxSwIsr_ (void)

Do the context switch - invoked from ISR.

9.7.4.16 void portInitEarly_ (void)

Early port initialization.

This function will be called at early initialization stage from [esKernInit\(\)](#) function. It is called before any kernel data initialization. Usually this function would be used to setup memory space, fill the memory with debug value or something similar.

9.7.4.17 void portInit_ (void)

Port initialization.

This function will be called after kernel data structure initialization from [esKernInit\(\)](#) function.

9.7.4.18 void portInitLate_ (void)

Late port initialization.

This function will be called just a moment before the multitasking is started. The function is called from [esKernStart\(\)](#) function.

9.7.5 Variable Documentation

9.7.5.1 `portReg_T gPortIsrNesting_`

Variable to keep track of ISR nesting.

9.7.5.2 `const PORT_C_ROM portReg_T pwr2LKP[PORT_DATA_WIDTH_VAL]`

Look up table for: 2^n expression.

This look up table can be used to accelerate the Logical Shift Left operations which are needed to set bits inside the priority bit map. In plain C this operation would be written as: `(1U << n)`, but in many 8-bit CPUs this operation can be lengthy. If there is a need for faster operation than this table can be used instead of the mentioned C code.

To use the look up table change `PORT_PWR2` macro implementation from: `(1U << (pwr))` to `pwr2LKP[pwr]`

9.8 CPU port internals

CPU port inner work.

Collaboration diagram for CPU port internals:



Functions

- `uint_fast8_t portFindLastSet_ (portReg_T val)`
Find last set bit in a word.
- `void * portCtxInit_ (void *stck, size_t stckSize, void(*thdf)(void *), void *arg)`
Initialize the thread context.

Variables

- `portReg_T gPortIsrNesting_`
Variable to keep track of ISR nesting.
- `const PORT_C_ROM portReg_T pwr2LKP [PORT_DATA_WIDTH_VAL]`
Look up table for: 2^n expression.

9.8.1 Detailed Description

CPU port inner work.

9.8.2 Function Documentation

9.8.2.1 `uint_fast8_t portFindLastSet_ (portReg_T val)`

Find last set bit in a word.

Parameters

<i>val</i>	Value which needs to be evaluated
------------	-----------------------------------

This function is used by the scheduler to efficiently determine the highest priority of thread ready for execution. For algorithm details see: http://en.wikipedia.org/wiki/Find_first_set.

Returns

The position of the last set bit in a word

9.8.2.2 `void* portCtxInit_ (void * stck, size_t stckSize, void(*) (void *) thdf, void * arg)`

Initialize the thread context.

Parameters

<code>in, out</code>	<code>stck</code>	Pointer to the allocated thread stck. The pointer points to the beginning of the memory as defined per C language. It's up to port function to adjust the pointer according to the stck type: full descending or full ascending one.
	<code>stckSize</code>	The size of allocated stck in bytes.
<code>in</code>	<code>thdf</code>	Pointer to the thread function.
<code>in</code>	<code>arg</code>	Argument that will be passed to thread function at the starting of execution.

Returns

The new top of stck after thread context initialization.

9.8.3 Variable Documentation**9.8.3.1 `portReg_T gPortIsrNesting`**

Variable to keep track of ISR nesting.

9.8.3.2 `const PORT_C_ROM portReg_T pwr2LKP[PORT_DATA_WIDTH_VAL]`**Initial value:**

```
= {
    (1U << 0), (1U << 1), (1U << 2), (1U << 3),
    (1U << 4), (1U << 5), (1U << 6), (1U << 7),

}
```

Look up table for: 2^n expression.

This look up table can be used to accelerate the Logical Shift Left operations which are needed to set bits inside the priority bit map. In plain C this operation would be written as: $(1U \ll n)$, but in many 8-bit CPUs this operation can be lengthy. If there is a need for faster operation than this table can be used instead of the mentioned C code.

To use the look up table change `PORT_PWR2` macro implementation from: $(1U \ll (pwr))$ to `pwr2LKP[pwr]`

9.9 CPU port configuration

CPU port specific configuration options.

Collaboration diagram for CPU port configuration:



Port General configuration

Configuration options and settings which are available for every port.

Note

- 1) All port General configuration macros are prefixed with `CFG_` string.
- 2) All port specific options and constants are prefixed with `CPU_` string.

- `#define CFG_CRITICAL_PRIO 1U`
Priority of critical sections in kernel.
- `#define CFG_SYSTM_CLOCK_FREQUENCY 1000000UL`
The frequency of clock which is used for the system timer.

9.9.1 Detailed Description

CPU port specific configuration options. Each configuration option or setting has its own default value when not defined by the application. When application needs to change a setting it just needs to define a configuration macro with another value and the default configuration macro will be overridden.

9.9.2 Macro Definition Documentation

9.9.2.1 `#define CFG_CRITICAL_PRIO 1U`

Priority of critical sections in kernel.

This option varies with the MCU used. In the simplest case when the MCU does not support interrupt priorities than only one priority level is available. In that case critical section will simply disable interrupts on entry and enable them at exit.

9.9.2.2 `#define CFG_SYSTM_CLOCK_FREQUENCY 1000000UL`

The frequency of clock which is used for the system timer.

Specify here the clock value so the kernel can properly manage system tick event generation. Usually system timer will use the clock of the processor. A hardware timer is configured to generate an interrupt at a rate between 10 and 1000 Hz which provides the system tick. The rate of interrupt is application specific and depends on the desired resolution system tick time source. However, the faster the tick rate, the higher the overhead will be imposed on the system.

9.10 Kernel configuration

Default Kernel configuration options.

Collaboration diagram for Kernel configuration:



Kernel configuration options and settings

Kernel default configuration

- #define `CFG_API_VALIDATION` 1U
Enable/disable API arguments validation.
- #define `CFG_SCHED_PRIO_LVL` 8U
Scheduler priority levels.
- #define `CFG_SCHED_TIME_QUANTUM` 10U
Scheduler Round-Robin time quantum.
- #define `CFG_SCHED_POWER_SAVE` 0U
Enable/disable scheduler power savings mode.
- #define `CFG_SYSTMTR_ADAPTIVE_MODE` 0U
System timer mode.
- #define `CFG_SYSTMTR_EVENT_FREQUENCY` 100UL
The frequency of system tick event.
- #define `CFG_SYSTMTR_TICK_TYPE` 2U
The size of the system timer counter.

Kernel hooks

- #define `CFG_HOOK_SYSTMTR_EVENT` 0U
System timer event hook function.
- #define `CFG_HOOK_KERN_INIT` 0U
Kernel initialization hook function.
- #define `CFG_HOOK_KERN_START` 0U
Kernel start hook function.
- #define `CFG_HOOK_THD_INIT_END` 0U
Thread initialization hook function.
- #define `CFG_HOOK_THD_TERM` 0U
Thread termination hook function.
- #define `CFG_HOOK_CTX_SW` 0U
Kernel context switch hook function.

9.10.1 Detailed Description

Default Kernel configuration options. Each configuration option or setting has its own default value when not defined by the application. When application needs to change a setting it just needs to define a configuration macro with another value and the default configuration macro will be overridden.

9.10.2 Macro Definition Documentation

9.10.2.1 `#define CFG_API_VALIDATION 1U`

Enable/disable API arguments validation.

During the development cycle of the application this option should be turned on. When this configuration option is turned on the kernel API functions will also check arguments passed to them. If an invalid argument is detected the execution of the application will stop and the user will be informed about the error condition.

Possible values:

- 0U - API validation is disabled
- 1U - API validation is enabled

Note

The error checking use [userAssert\(\)](#) hook function to provide the information about the error condition.

9.10.2.2 `#define CFG_SCHED_PRIO_LVL 8U`

Scheduler priority levels.

The number of priority levels. Each priority level can have several threads. Possible values:

- Min: 3U (three priority levels)
- Max: 256U

9.10.2.3 `#define CFG_SCHED_TIME_QUANTUM 10U`

Scheduler Round-Robin time quantum.

This constant is the number of system ticks allowed for the threads before preemption occurs. Setting this value to zero disables the preemption for threads with equal priority and the round robin becomes cooperative. Note that higher priority threads can still preempt, the kernel is always preemptive.

Note

Disabling the round robin preemption makes the kernel more compact and generally faster.

9.10.2.4 `#define CFG_SCHED_POWER_SAVE 0U`

Enable/disable scheduler power savings mode.

Possible values are:

- 0U - power saving is disabled
- 1U - power saving is enabled

9.10.2.5 `#define CFG_SYSTMTR_ADAPTIVE_MODE 0U`

System timer mode.

Possible values are:

- 0U - adaptive mode is disabled
- 1U - adaptive mode is enabled

9.10.2.6 `#define CFG_SYSTMTR_EVENT_FREQUENCY 100UL`

The frequency of system tick event.

Specify the desired resolution system tick time source. This setting is valid only if configuration option [CFG_SYSTMTR_CLOCK_FREQUENCY](#) is properly set in port configuration file [cpu_cfg.h](#)

9.10.2.7 `#define CFG_SYSTMTR_TICK_TYPE 2U`

The size of the system timer counter.

Possible values are:

- 0U - 8 bit counter
- 1U - 16 bit counter
- 2U - 32 bit counter

9.10.2.8 `#define CFG_HOOK_SYSTMTR_EVENT 0U`

System timer event hook function.

This hook is called just a moment before a system timer event is processed.

Note

This hook will call `userSysTmr()` function.

9.10.2.9 `#define CFG_HOOK_KERN_INIT 0U`

Kernel initialization hook function.

This hook is called at the beginning of [esKernInit\(\)](#) function.

Note

This hook will call `userKernInit()` function.

9.10.2.10 `#define CFG_HOOK_KERN_START 0U`

Kernel start hook function.

This hook is called at the beginning of [esKernStart\(\)](#) function.

Note

This hook will call `userKernStart()` function.

9.10.2.11 `#define CFG_HOOK_THD_INIT_END 0U`

Thread initialization hook function.

This hook is called at the end of `esThdInit()` function.

Note

This hook will call `userThdInitEnd()` function.

9.10.2.12 `#define CFG_HOOK_THD_TERM 0U`

Thread termination hook function.

This hook is called when a thread terminates.

Note

This hook will call `userThdTerm()` function.

9.10.2.13 `#define CFG_HOOK_CTX_SW 0U`

Kernel context switch hook function.

This hook is called at each context switch.

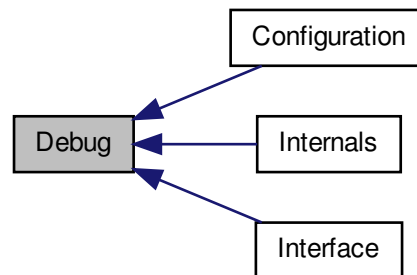
Note

This hook will call `userCtxSw()` function.

9.11 Debug

Overview.

Collaboration diagram for Debug:



Modules

- [Configuration](#)
Configuration settings.
- [Interface](#)
Application programming interface.
- [Internals](#)
Debug inner work.

9.11.1 Detailed Description

Overview.

9.12 Interface

Application programming interface.

Collaboration diagram for Interface:



Error checking

Some basic infrastructure for error checking

For more details see [Error checking](#).

- `PORT_C_NORETURN void esDbgAssert (const char *fnName, const char *expr, enum esDbgMsg msg)`
An assertion has failed.
- `#define ES_DBG_ASSERT(msg, expr)`
Generic assert macro.
- `#define ES_DBG_ASSERT_ALWAYS(msg, text)`
Assert macro that will always execute (no conditional).

Internal checking

These macros are enabled/disabled using the option [CFG_DBG_INTERNAL_CHECK](#).

- `#define ES_DBG_INTERNAL(msg, expr) ES_DBG_ASSERT(msg, expr)`
Assert macro used for internal execution checking.

API contract validation

These macros are enabled/disabled using the option [CFG_DBG_API_VALIDATION](#).

- `enum esDbgMsg {`
`ES_DBG_OUT_OF_RANGE,`
`ES_DBG_OBJECT_NOT_VALID,`
`ES_DBG_POINTER_NULL,`
`ES_DBG_USAGE_FAILURE,`
`ES_DBG_NOT_ENOUGH_MEM,`
`ES_DBG_UNKNOWN_ERROR = 0xFFU }`
Debug messages.
- `#define ES_DBG_API_OBLIGATION(expr) expr`
Execute code to fulfill the contract.
- `#define ES_DBG_API_REQUIRE(msg, expr) ES_DBG_ASSERT(msg, expr)`
Make sure the caller has fulfilled all contract preconditions.
- `#define ES_DBG_API_ENSURE(msg, expr) ES_DBG_ASSERT(msg, expr)`
Make sure the callee has fulfilled all contract postconditions.

Debug hook functions

Note

1) The definition of this functions must be written by the user.

- void `userAssert` (const char *fnName, const char *expr, const char *msg)
An assertion has failed. This function should inform the user about failed assertion.

9.12.1 Detailed Description

Application programming interface.

9.12.2 Macro Definition Documentation

9.12.2.1 `#define ES_DBG_ASSERT(msg, expr)`

Value:

```
do {
    if (! (expr)) {
        esDbgAssert (PORT_C_FUNC, #expr, msg);
    }
} while (0U)
```

Generic assert macro.

Parameters

<i>msg</i>	Enumerator enum esDbgMsg: enumerated debug message.
<i>expr</i>	Condition expression which must be TRUE.

9.12.2.2 `#define ES_DBG_ASSERT_ALWAYS(msg, text)`

Value:

```
do {
    esDbgAssert (PORT_C_FUNC, text, msg);
} while (0U)
```

Assert macro that will always execute (no conditional).

Parameters

<i>msg</i>	Enumerator enum esDbgMsg: enumerated kernel message.
<i>text</i>	Pointer to string: a text which will be printed when this assert macro is executed.

9.12.2.3 `#define ES_DBG_INTERNAL(msg, expr) ES_DBG_ASSERT(msg, expr)`

Assert macro used for internal execution checking.

Parameters

<i>msg</i>	Enumerator enum esDbgMsg: enumerated debug message.
<i>expr</i>	Expression which must be satisfied

9.12.2.4 `#define ES_DBG_API_OBLIGATION(expr) expr`

Execute code to fulfill the contract.

Parameters

<i>expr</i>	Expression to be executed only if contracts need to be validated.
-------------	---

9.12.2.5 `#define ES_DBG_API_REQUIRE(msg, expr) ES_DBG_ASSERT(msg, expr)`

Make sure the caller has fulfilled all contract preconditions.

Parameters

<i>msg</i>	Enumerator enum <code>esDbgMsg</code> : enumerated debug message.
<i>expr</i>	Expression which must be satisfied

9.12.2.6 `#define ES_DBG_API_ENSURE(msg, expr) ES_DBG_ASSERT(msg, expr)`

Make sure the callee has fulfilled all contract postconditions.

Parameters

<i>msg</i>	Enumerator enum <code>esDbgMsg</code> : enumerated debug message.
<i>expr</i>	Expression which must be satisfied

9.12.3 Enumeration Type Documentation

9.12.3.1 `enum esDbgMsg`

Debug messages.

Enumerator

- `ES_DBG_OUT_OF_RANGE`** Value is out of valid range.
- `ES_DBG_OBJECT_NOT_VALID`** Object is not valid.
- `ES_DBG_POINTER_NULL`** Pointer has NULL value.
- `ES_DBG_USAGE_FAILURE`** Object usage failure.
- `ES_DBG_NOT_ENOUGH_MEM`** Not enough memory available.
- `ES_DBG_UNKNOWN_ERROR`** Unknown error.

9.12.4 Function Documentation

9.12.4.1 `PORT_C.NORETURN void esDbgAssert (const char * fnName, const char * expr, enum esDbgMsg msg)`

An assertion has failed.

Parameters

<i>fnName</i>	Function name: is pointer to the function name string where the assertion has failed. Macro will automatically fill in the function name.
<i>expr</i>	Expression: is pointer to the string containing the expression that failed to evaluate to <code>TRUE</code> .
<i>msg</i>	Message: is enum <code>esDbgMsg</code> containing some information about the error.

Precondition

- 1) `NULL != fnName`
- 2) `NULL != expr`

Note

- 1) This function is called only if [CFG_DBG_API_VALIDATION](#) is active.

Function will just print the information which was given by the macros.

Object class:

Not API object, this object is not part of the application programming interface and it is intended for internal use only.

9.12.4.2 void userAssert (const char * *fnName*, const char * *expr*, const char * *msg*)

An assertion has failed. This function should inform the user about failed assertion.

Parameters

<i>fnName</i>	Function name: is pointer to the function name string where the assertion has failed. Macro will automatically fill in the function name.
<i>expr</i>	Expression: is pointer to the string containing the expression that failed to evaluate to <code>TRUE</code> .
<i>msg</i>	Message: is a pointer to the string containing some information about the error.

Precondition

- 1) `NULL != fnName`
- 2) `NULL != expr`
- 3) `NULL != msg`

Note

- 1) This function is called only if [CFG_DBG_ENABLE](#) is active.
- 2) The function is called with interrupts disabled.

Function will just print the information which was given by the macros.

9.13 Internals

Debug inner work.

Collaboration diagram for Internals:



Functions

- `PORT_C_NORETURN void esDbgAssert (const char *fnName, const char *expr, enum esDbgMsg msg)`
An assertion has failed.

9.13.1 Detailed Description

Debug inner work.

9.13.2 Function Documentation

9.13.2.1 `PORT_C_NORETURN void esDbgAssert (const char * fnName, const char * expr, enum esDbgMsg msg)`

An assertion has failed.

Parameters

<i>fnName</i>	Function name: is pointer to the function name string where the assertion has failed. Macro will automatically fill in the function name.
<i>expr</i>	Expression: is pointer to the string containing the expression that failed to evaluate to <code>TRUE</code> .
<i>msg</i>	Message: is enum <code>esDbgMsg</code> containing some information about the error.

Precondition

- 1) `NULL != fnName`
- 2) `NULL != expr`

Note

- 1) This function is called only if `CFG_DBG_API_VALIDATION` is active.

Function will just print the information which was given by the macros.

Object class:

Not API object, this object is not part of the application programming interface and it is intended for internal use only.

9.14 Configuration

Configuration settings.

Collaboration diagram for Configuration:



Macros

- `#define CFG_DBG_ENABLE 1U`
Enable/disable Debug module.
- `#define CFG_DBG_API_VALIDATION 1U`
Enable/disable API arguments validation.
- `#define CFG_DBG_INTERNAL_CHECK 1U`
Enable/disable internal checks.

9.14.1 Detailed Description

Configuration settings.

9.14.2 Macro Definition Documentation

9.14.2.1 `#define CFG_DBG_ENABLE 1U`

Enable/disable Debug module.

Possible values:

- 0U - Debug is disabled
- 1U - Debug is enabled

9.14.2.2 `#define CFG_DBG_API_VALIDATION 1U`

Enable/disable API arguments validation.

Possible values:

- 0U - API validation is disabled
- 1U - API validation is enabled

Note

This option is enabled only if `CFG_DBG_ENABLE` is enabled, too.

9.14.2.3 #define CFG_DBG_INTERNAL_CHECK 1U

Enable/disable internal checks.

Possible values:

- 0U - API validation is disabled
- 1U - API validation is enabled

Note

This option is enabled only if [CFG_DBG_ENABLE](#) is enabled, too.

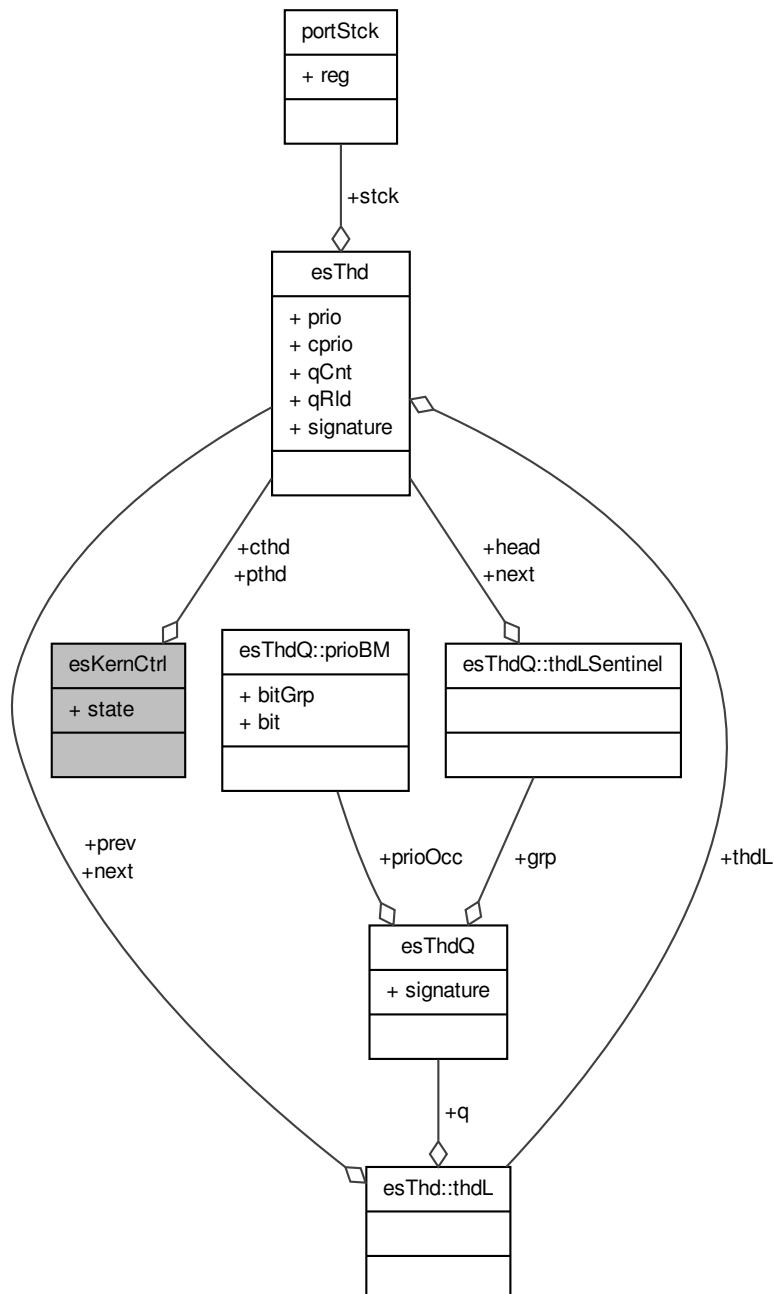
10 Data Structure Documentation

10.1 esKernCtrl Struct Reference

Kernel control block structure.

```
#include <kernel.h>
```

Collaboration diagram for esKernCtrl:



Data Fields

- struct [esThd](#) * [cthd](#)
Pointer to the Current Thread.
- struct [esThd](#) * [pthd](#)
Pointer to the Pending Thread to be switched.
- enum [esKernState](#) [state](#)
State of kernel.

10.1.1 Detailed Description

Kernel control block structure.

This structure holds important status data about the kernel. Since all data within the structure is somewhat related and accessed within the same pieces of code it was decided it is better to group all kernel data into the structure. This way the compiler can generate code that gets the address of the structure and then use relative indirect addressing to access all members of the structure. This results in more efficient code on architectures that have relative indirect addressing capability.

Object class:

Regular **API** object, this object is part of the application programming interface.

10.1.2 Field Documentation

10.1.2.1 struct [esThd](#)* [esKernCtrl::cthd](#)

Pointer to the Current Thread.

10.1.2.2 struct [esThd](#)* [esKernCtrl::pthd](#)

Pointer to the Pending Thread to be switched.

10.1.2.3 enum [esKernState](#) [esKernCtrl::state](#)

State of kernel.

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

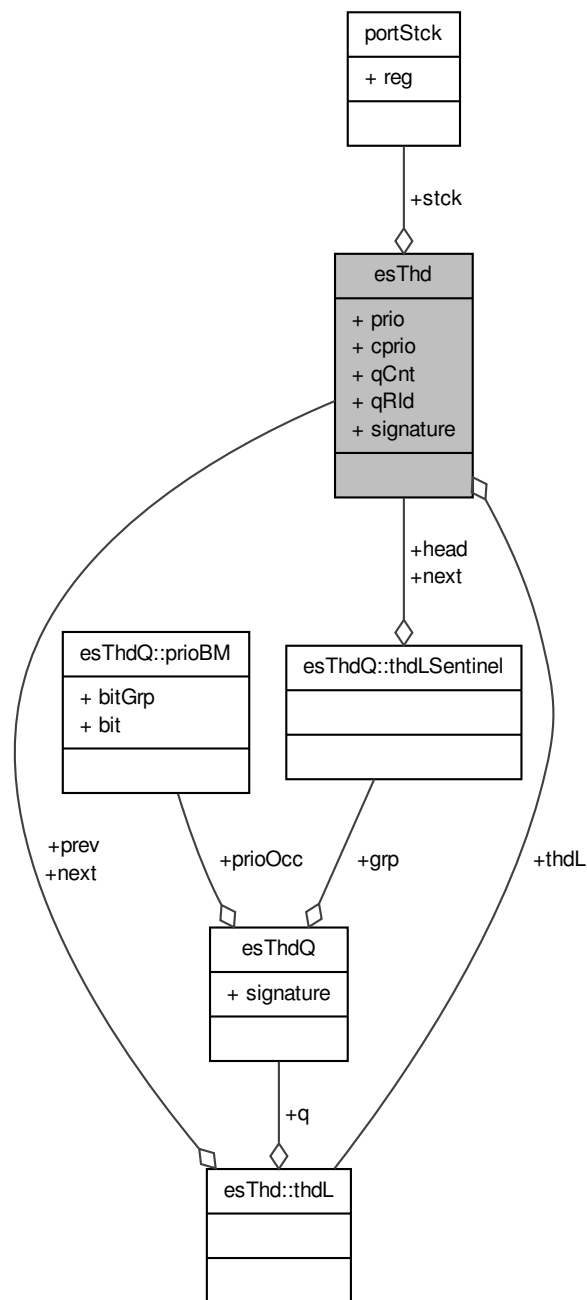
- [kernel.h](#)

10.2 esThd Struct Reference

Thread structure.

```
#include <kernel.h>
```

Collaboration diagram for esThd:



Data Structures

- struct [thdL](#)

Thread linked List structure.

Data Fields

- [portStck_T * stck](#)
Pointer to thread's Top Of Stack.
- [struct esThd::thdL thdL](#)
Thread linked list.
- [uint_fast8_t prio](#)
Thread current priority level.
- [uint_fast8_t cprio](#)
Constant Thread Priority level.
- [uint_fast8_t qCnt](#)
Quantum counter.
- [uint_fast8_t qRld](#)
Quantum counter reload value.
- [portReg_T signature](#)
Thread structure signature, see [Error checking](#).

10.2.1 Detailed Description

Thread structure.

A thread structure is a data structure used by kernel to maintain information about a thread. Each thread requires its own ID structure and the structure is allocated in user memory space (RAM). The address of the thread's ID structure is provided to OS thread-related services.

Thread structure is used as thread ID and a thread is always referenced using this structure.

Object class:

Regular **API** object, this object is part of the application programming interface.

10.2.2 Field Documentation

10.2.2.1 [portStck_T* esThd::stck](#)

Pointer to thread's Top Of Stack.

10.2.2.2 [struct esThd::thdL esThd::thdL](#)

Thread linked list.

10.2.2.3 [uint_fast8_t esThd::prio](#)

Thread current priority level.

10.2.2.4 [uint_fast8_t esThd::cprio](#)

Constant Thread Priority level.

10.2.2.5 [uint_fast8_t esThd::qCnt](#)

Quantum counter.

10.2.2.6 [uint_fast8_t esThd::qRld](#)

Quantum counter reload value.

10.2.2.7 portReg_T esThd::signature

Thread structure signature, see [Error checking](#).

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

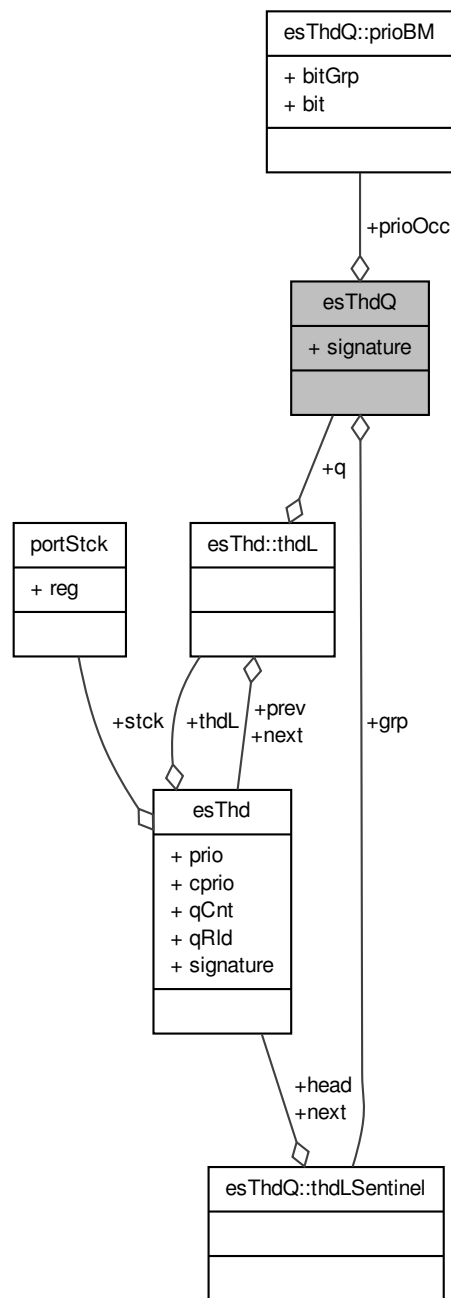
- [kernel.h](#)

10.3 esThdQ Struct Reference

Thread Queue structure.

```
#include <kernel.h>
```


Collaboration diagram for esThdQ:



Data Structures

- struct [prioBM](#)
Priority Bit Map structure.
- struct [thdLSentinel](#)
Thread linked list sentinel structure.

Data Fields

- struct [esThdQ::prioBM prioOcc](#)
Priority Occupancy.
- struct [esThdQ::thdLSentinel grp](#) [CFG_SCHED_PRIO_LVL]
Array of thread linked list sentinel structures.
- [portReg_T signature](#)
Thread Queue struct signature, see [Error checking](#).

10.3.1 Detailed Description

Thread Queue structure.

Object class:

Regular **API** object, this object is part of the application programming interface.

10.3.2 Field Documentation

10.3.2.1 struct [esThdQ::prioBM esThdQ::prioOcc](#)

Priority Occupancy.

10.3.2.2 struct [esThdQ::thdLSentinel esThdQ::grp](#)[CFG_SCHED_PRIO_LVL]

Array of thread linked list sentinel structures.

10.3.2.3 [portReg_T esThdQ::signature](#)

Thread Queue struct signature, see [Error checking](#).

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

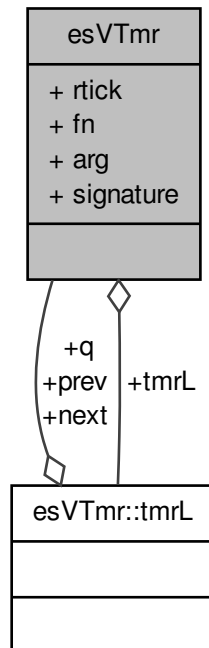
- [kernel.h](#)

10.4 esVTmr Struct Reference

Virtual Timer structure.

```
#include <kernel.h>
```

Collaboration diagram for esVTmr:



Data Structures

- struct [tmrL](#)
Virtual Timer linked list structure.

Data Fields

- struct [esVTmr::tmrL](#) [tmrL](#)
Virtual Timer linked List.
- [esTick_T](#) [rtick](#)
Relative tick value.
- `void(* fn)(void *)`
Callback function pointer.
- `void * arg`
Callback function argument.
- [portReg_T](#) [signature](#)
Timer structure signature, see [Error checking](#).

10.4.1 Detailed Description

Virtual Timer structure.

10.4.2 Field Documentation

10.4.2.1 struct esVTmr::tmrL esVTmr::tmrL

Virtual Timer linked List.

10.4.2.2 esTick_T esVTmr::rtick

Relative tick value.

10.4.2.3 void(* esVTmr::fn)(void *)

Callback function pointer.

10.4.2.4 void* esVTmr::arg

Callback function argument.

10.4.2.5 portReg_T esVTmr::signature

Timer structure signature, see [Error checking](#).

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

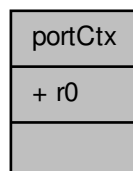
- [kernel.h](#)

10.5 portCtx Struct Reference

Port context structure.

```
#include <cpu.h>
```

Collaboration diagram for portCtx:



Data Fields

- [portReg_T r0](#)

Data pushed on stack during context switching.

10.5.1 Detailed Description

Port context structure.

10.5.2 Field Documentation

10.5.2.1 portReg_T portCtx::r0

Data pushed on stack during context switching.

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

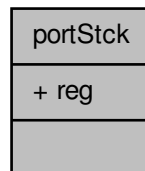
- [cpu.h](#)

10.6 portStck Struct Reference

Stack structure used for stack declaration in order to force the alignment Alignment of stack structure.

```
#include <cpu.h>
```

Collaboration diagram for portStck:



Data Fields

- [portReg_T reg](#)

A structure field representing stack data.

10.6.1 Detailed Description

Stack structure used for stack declaration in order to force the alignment Alignment of stack structure.

10.6.2 Field Documentation

10.6.2.1 portReg_T portStck::reg

A structure field representing stack data.

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

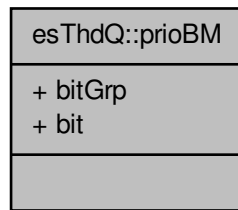
- [cpu.h](#)

10.7 esThdQ::prioBM Struct Reference

Priority Bit Map structure.

```
#include <kernel.h>
```

Collaboration diagram for esThdQ::prioBM:



Data Fields

- [portReg_T bitGrp](#)
Bit group indicator.
- [portReg_T bit \[PRIO_BM_GRP_INDX\]](#)
Bit priority indicator.

10.7.1 Detailed Description

Priority Bit Map structure.

10.7.2 Field Documentation

10.7.2.1 `portReg_T esThdQ::prioBM::bitGrp`

Bit group indicator.

10.7.2.2 `portReg_T esThdQ::prioBM::bit[PRIO_BM_GRP_INDX]`

Bit priority indicator.

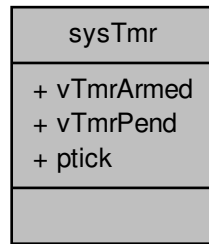
The documentation for this struct was generated from the following file:

- [kernel.h](#)

10.8 sysTmr Struct Reference

Main System Timer structure.

Collaboration diagram for sysTmr:



Data Fields

- `uint_fast16_t vTmrArmed`
The number of armed virtual timers in system.
- `uint_fast16_t vTmrPend`
The number of pending timers for arming.
- `esTick_T ptick`
Pending ticks during the timer sleep mode.

10.8.1 Detailed Description

Main System Timer structure.

Note

- 1) Member `ptick` exists only if ADAPTIVE mode is selected. When this mode is selected then kernel supports more aggressive power savings.

10.8.2 Field Documentation

10.8.2.1 `uint_fast16_t sysTmr::vTmrArmed`

The number of armed virtual timers in system.

10.8.2.2 `uint_fast16_t sysTmr::vTmrPend`

The number of pending timers for arming.

10.8.2.3 `esTick_T sysTmr::ptick`

Pending ticks during the timer sleep mode.

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

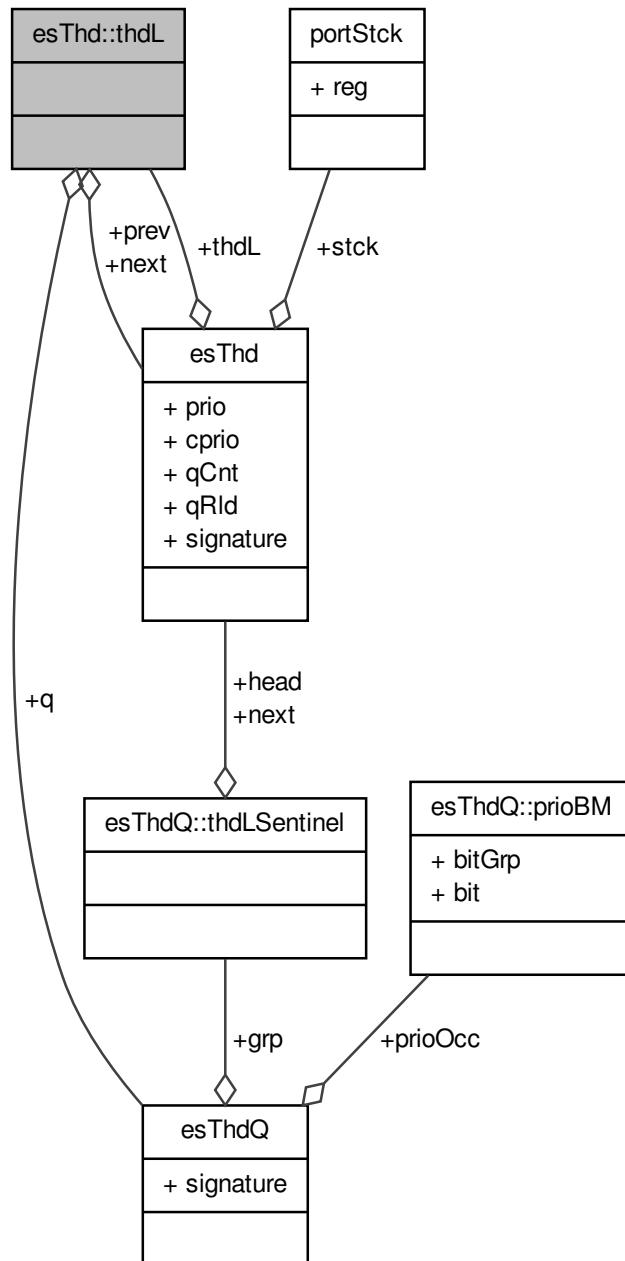
- [kernel.c](#)

10.9 esThd::thdL Struct Reference

Thread linked List structure.

```
#include <kernel.h>
```

Collaboration diagram for esThd::thdL:



Data Fields

- struct `esThdQ` * `q`

- Points to parent thread queue.*
 - struct [esThd](#) * [next](#)
Next thread in linked list.
 - struct [esThd](#) * [prev](#)
Previous thread in linked list.

10.9.1 Detailed Description

Thread linked List structure.

10.9.2 Field Documentation

10.9.2.1 struct [esThdQ](#)* [esThd::thdL::q](#)

Points to parent thread queue.

10.9.2.2 struct [esThd](#)* [esThd::thdL::next](#)

Next thread in linked list.

10.9.2.3 struct [esThd](#)* [esThd::thdL::prev](#)

Previous thread in linked list.

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

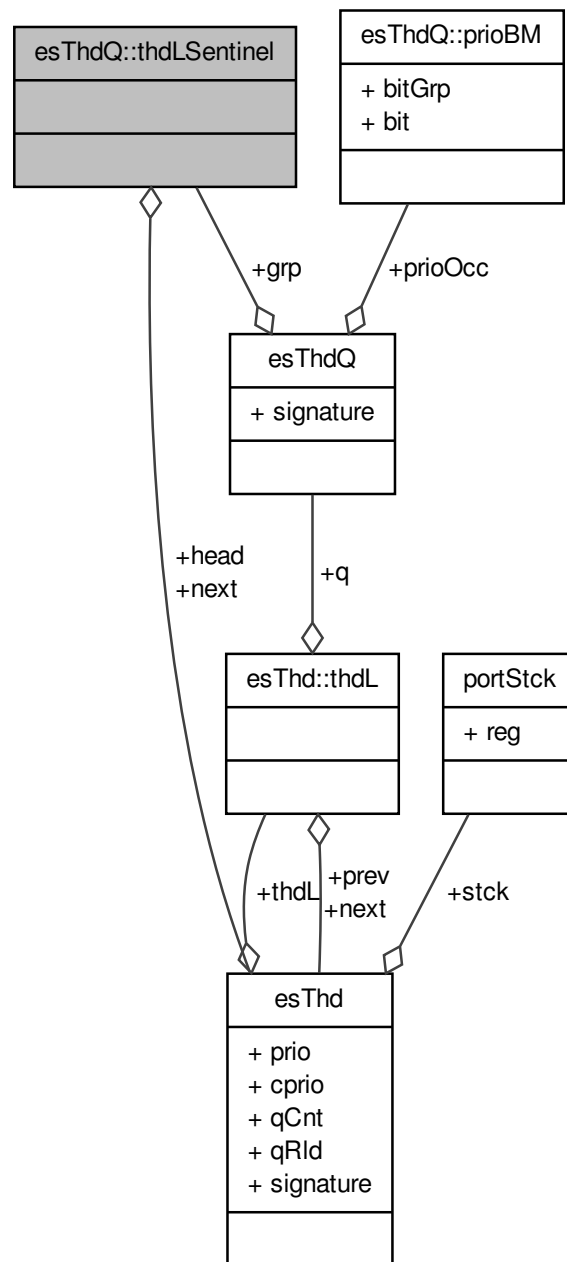
- [kernel.h](#)

10.10 esThdQ::thdLSentinel Struct Reference

Thread linked list sentinel structure.

```
#include <kernel.h>
```

Collaboration diagram for esThdQ::thdLSentinel:



Data Fields

- struct **esThd** * **head**
Points to the first thread in linked list.
- struct **esThd** * **next**
Points to the next thread in linked list.

10.10.1 Detailed Description

Thread linked list sentinel structure.

10.10.2 Field Documentation

10.10.2.1 struct esThd* esThdQ::thdLSentinel::head

Points to the first thread in linked list.

10.10.2.2 struct esThd* esThdQ::thdLSentinel::next

Points to the next thread in linked list.

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

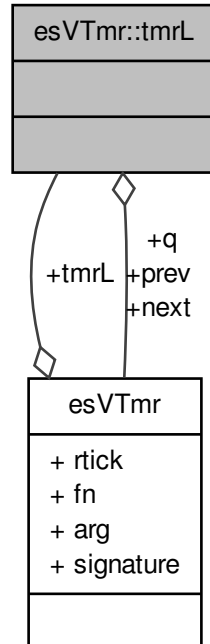
- [kernel.h](#)

10.11 esVTmr::tmrL Struct Reference

Virtual Timer linked list structure.

```
#include <kernel.h>
```

Collaboration diagram for esVTmr::tmrL:



Data Fields

- struct [esVTmr](#) * [q](#)

- *Points to parent timer list.*
- struct `esVTmr * next`
Next thread in Virtual Timer linked list.
- struct `esVTmr * prev`
Previous thread in virtual timer linked list.

10.11.1 Detailed Description

Virtual Timer linked list structure.

10.11.2 Field Documentation

10.11.2.1 struct `esVTmr* esVTmr::tmrL::q`

Points to parent timer list.

10.11.2.2 struct `esVTmr* esVTmr::tmrL::next`

Next thread in Virtual Timer linked list.

10.11.2.3 struct `esVTmr* esVTmr::tmrL::prev`

Previous thread in virtual timer linked list.

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

- [kernel.h](#)

11 File Documentation

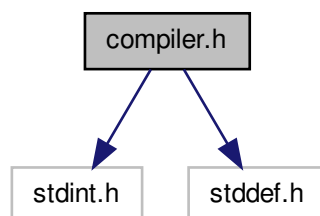
11.1 compiler.h File Reference

Interface of Compiler port - Template.

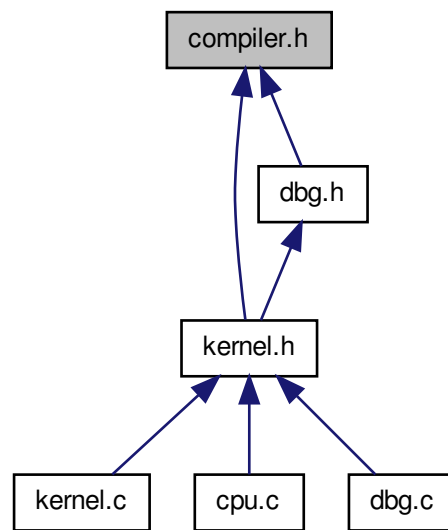
```
#include <stdint.h>
```

```
#include <stddef.h>
```

Include dependency graph for compiler.h:



This graph shows which files directly or indirectly include this file:



Macros

Compiler provided macros

Port interface macros and port specific macros

*These macros are used to ease the writing of ports. All macros prefixed with **PORT_** are part of the port interface.*

- #define **PORT_C_INLINE** inline
C extension - make a function inline.
- #define **PORT_C_INLINE_ALWAYS** inline
C extension - make a function inline - always.
- #define **PORT_C_NAKED**
Omit function prologue/epilogue sequences.
- #define **PORT_C_FUNC** "unknown"
Provides function name for assert macros.
- #define **PORT_C_WEAK**
Declares a weak function.
- #define **PORT_C_ALIGNED**(expr)
This attribute specifies a minimum alignment (in bytes) for variables of the specified type.
- #define **PORT_HWREG_SET**(reg, mask, val)
A standardized way of properly setting the value of HW register.

Compiler provided data types

The compiler port must provide some C90 (C99) data types

The compiler port must:

- declare sets of integer types having specified widths, standard type definitions and shall define corresponding sets of macros.

Types are defined in the following categories:

- Integer types having certain exact widths
- Fastest integer types having at least certain specified widths
- Integer types wide enough to hold pointers to objects
- standard type definitions

The following exact-width integer types are required:

- `int8_t`
- `int16_t`
- `int32_t`
- `uint8_t`
- `uint16_t`
- `uint32_t`

The following fastest minimum-width integer types are required:

- `int_fast8_t`
- `int_fast16_t`
- `int_fast32_t`
- `uint_fast8_t`
- `uint_fast16_t`
- `uint_fast32_t`

The following integer types capable of holding object pointers are required:

- `intptr_t`
- `uintptr_t`

The following standard type definitions are required:

- `NULL`
- `ptrdiff_t`
- `size_t`
- `enum boolType {
 TRUE = 1U,
 FALSE = 0U }`
 Bool data type.
- `typedef enum boolType bool_T`
 Bool data type.

11.1.1 Detailed Description

Interface of Compiler port - Template.

Author

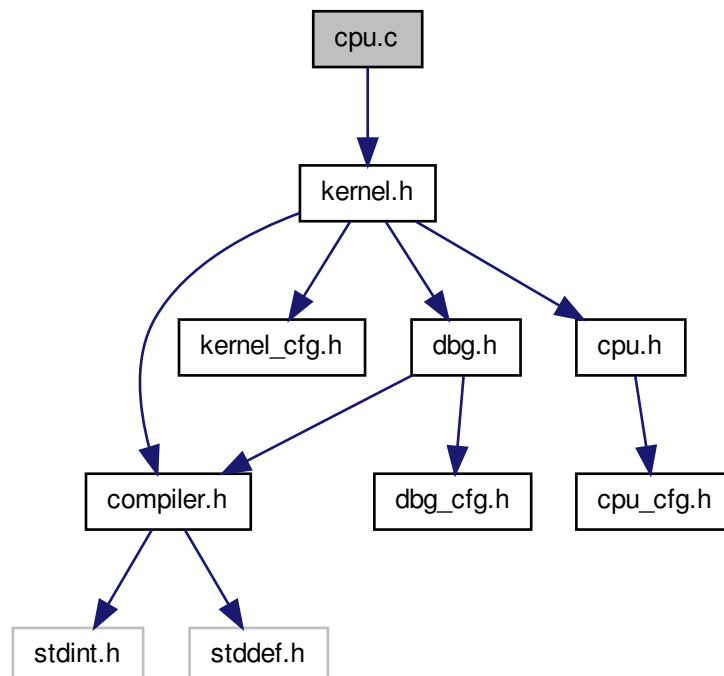
Nenad Radulovic

11.2 **cpu.c File Reference**

Implementation of CPU port - Template.

```
#include "kernel.h"
```

Include dependency graph for cpu.c:



Functions

- `uint_fast8_t portFindLastSet_ (portReg_T val)`
Find last set bit in a word.
- `void * portCtxInit_ (void *stck, size_t stckSize, void(*thdf)(void *), void *arg)`
Initialize the thread context.

Variables

- `portReg_T gPortIsrNesting_`
Variable to keep track of ISR nesting.

- const PORT_C_ROM [portReg_T](#) [pwr2LKP](#) [[PORT_DATA_WIDTH_VAL](#)]
Look up table for: 2^n expression.

11.2.1 Detailed Description

Implementation of CPU port - Template.

Author

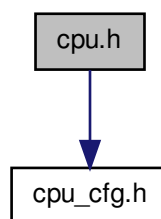
Nenad Radulovic

11.3 cpu.h File Reference

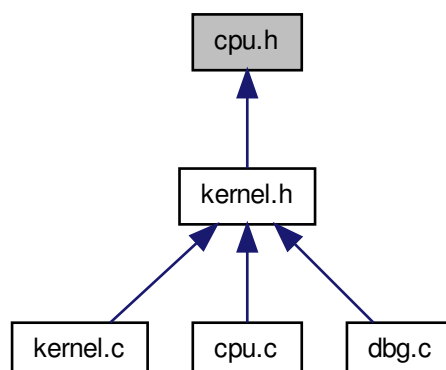
Interface of CPU port - Template.

```
#include "cpu_cfg.h"
```

Include dependency graph for cpu.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [portStck](#)
Stack structure used for stack declaration in order to force the alignment Alignment of stack structure.
- struct [portCtx](#)
Port context structure.

Macros

Port constants

- #define [PORT_DATA_WIDTH_VAL](#) 8U
This macro specifies the bit width of CPU data registers.
- #define [PORT_STCK_MINSIZE_VAL](#) sizeof(struct [portCtx](#))
This macro specifies the minimal size of the thread stack.
- #define [PORT_SYSTMTR_RELOAD_VAL](#) (CFG_SYSTMTR_CLOCK_FREQUENCY / CFG_SYSTMTR_EVENT_FREQUENCY)
System timer reload value for one tick.
- #define [PORT_SYSTMTR_MAX_VAL](#) 0xFFU
System timer maximum value.
- #define [PORT_SYSTMTR_MAX_TICKS_VAL](#) (PORT_SYSTMTR_MAX_VAL / PORT_SYSTMTR_RELOAD_VAL)
Maximum number of ticks the system timer can accept.

Critical section management

Disable/enable interrupts by preserving the status of interrupts.

Generally speaking these macros would store the status of the interrupt disable flag in the local variable declared by [PORT_CRITICAL_DECL](#) and then disable interrupts. Local variable is allocated in all of eSolid RTOS functions that need to disable interrupts. Macros would restore the interrupt status by copying back the allocated variable into the CPU's status register.

- #define [PORT_CRITICAL_DECL](#)() [portReg_T](#) intStatus_
Declare the interrupt status variable.
- #define [PORT_CRITICAL_ENTER](#)()
Enter critical section.
- #define [PORT_CRITICAL_EXIT](#)() [portIntSet](#)_(intStatus_)
Exit critical section.

General port macros

- #define [PORT_STCK_SIZE](#)(size)
Calculate the stack size.
- #define [PORT_CRITICAL_EXIT_SLEEP](#)() [PORT_CRITICAL_EXIT](#)()
TODO.
- #define [PORT_INIT_EARLY](#)() [portInitEarly](#)_
Early port initialization.
- #define [PORT_INIT](#)() [portInit](#)_
Port initialization.
- #define [PORT_INIT_LATE](#)() [portInitLate](#)_
Late port initialization.

Typedefs

- typedef uint8_t [portReg_T](#)
Data type which corresponds to the general purpose register.
- typedef struct [portStck](#) [portStck_T](#)
Stack type.

Functions

General port functions

- void `portInitEarly_` (void)
Early port initialization.
- void `portInit_` (void)
Port initialization.
- void `portInitLate_` (void)
Late port initialization.

Variables

- `portReg_T gPortIsrNesting_`
Variable to keep track of ISR nesting.
- const `PORT_C_ROM portReg_T pwr2LKP [PORT_DATA_WIDTH_VAL]`
Look up table for: 2^n expression.

Interrupt management

- `#define PORT_INT_DISABLE() portIntDisable_()`
Disable all interrupt sources.
- `#define PORT_ISR_ENTER()`
Enter ISR. Increment `gPortIsrNesting_` variable to keep track of ISR nesting.
- `#define PORT_ISR_EXIT()`
Exit ISR. Decrement `gPortIsrNesting_` variable to keep track of ISR nesting.
- `#define PORT_ISR_IS_LAST() (0U == gPortIsrNesting ? TRUE : FALSE)`
If `isrNesting` variable is zero then the last ISR is executing and scheduler should be invoked.
- void `portIntDisable_` (void)
Disable interrupts.
- `portReg_T portIntGet_` (void)
Get the current status of enabled/disabled interrupts.
- void `portIntSet_` (`portReg_T` status)
Set the status of interrupts according to the `status` argument.

Scheduler support

Note

These functions are extensively used by the scheduler and therefore they should be optimized for the architecture being used.

- `#define PORT_FIND_LAST_SET(val) portFindLastSet_(val)`
Find last set bit in a word.
- `#define PORT_PWR2(pwr) (1U << (pwr))`
Helper macro: calculate 2^{pwr} expression.
- `#define PORT_SYSTMTR_INIT() portSysTmrInit_()`
Initialize system timer and associated interrupt.
- `#define PORT_SYSTMTR_TERM() portSysTmrTerm_()`
Stop the timer if it is running and disable associated interrupt.
- `#define PORT_SYSTMTR_RELOAD(ticks) portSysTmrReload_(ticks)`
Reload the system timer with specified number of ticks.
- `#define PORT_SYSTMTR_ENABLE() portSysTmrEnable_()`

- Enable the system timer.*
 - #define `PORT_SYSTMTR_DISABLE()` `portSysTmrDisable_()`
- Disable the system timer.*
 - #define `PORT_SYSTMTR_ISR_ENABLE()` `portSysTmrIsrEnable_()`
- Enable the system timer interrupt.*
 - #define `PORT_SYSTMTR_ISR_DISABLE()` `portSysTmrIsrDisable_()`
- Disable the system timer interrupt.*
 - `uint_fast8_t portFindLastSet_ (portReg_T val)`
- Find last set bit in a word.*
 - void `portSysTmrInit_ (void)`
- Initialize systick timer and associated interrupt.*
 - void `portSysTmrTerm_ (void)`
- Stop the sistem timer.*
 - void `portSysTmrReload_ (esTick_T ticks)`
- Reload the system timer.*
 - void `portSysTmrEnable_ (void)`
- Enable the system timer.*
 - void `portSysTmrDisable_ (void)`
- Disable the system timer.*
 - void `portSysTmrIsrEnable_ (void)`
- Disable the system timer interrupt.*
 - void `portSysTmrIsrDisable_ (void)`
- Enable the system timer interrupt.*
 - void `portThdStart_ (void)`
- Start the first thread.*

Dispatcher context switching

- #define `PORT_CTX_INIT(stck, stackSize, thread, arg)` `portCtxInit_(stck, stackSize, thread, arg)`*Initialize the thread context.*
- #define `PORT_CTX_SW()` `portCtxSw_()`*Do the context switch - invoked from API level.*
- #define `PORT_CTX_SW_ISR()` `portCtxSwIsr_()`*Do the context switch - invoked from ISR level.*
- #define `PORT_THD_START()` `portThdStart_()`*Start the first thread.*
- void * `portCtxInit_ (void *stck, size_t stckSize, void(*thdf)(void *), void *arg)`*Initialize the thread context.*
- void `portCtxSw_ (void)`*Do the context switch - invoked from API.*
- void `portCtxSwIsr_ (void)`*Do the context switch - invoked from ISR.*

11.3.1 Detailed Description

Interface of CPU port - Template.

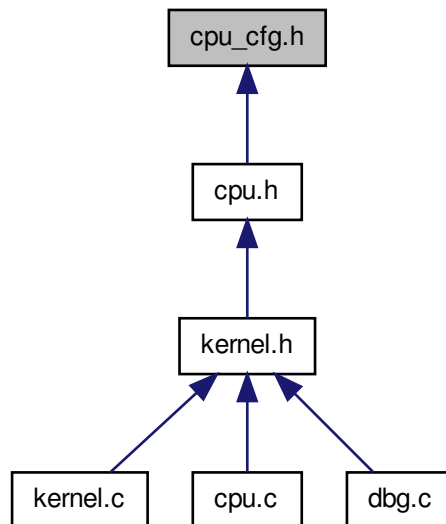
Author

Nenad Radulovic

11.4 `cpu_cfg.h` File Reference

Configuration of CPU port - Template.

This graph shows which files directly or indirectly include this file:



Macros

Port General configuration

Configuration options and settings which are available for every port.

Note

- 1) All port General configuration macros are prefixed with `CFG_` string.
- 2) All port specific options and constants are prefixed with `CPU_` string.

- `#define CFG_CRITICAL_PRIO 1U`
Priority of critical sections in kernel.
- `#define CFG_SYSTM_CLOCK_FREQUENCY 1000000UL`
The frequency of clock which is used for the system timer.

11.4.1 Detailed Description

Configuration of CPU port - Template.

Author

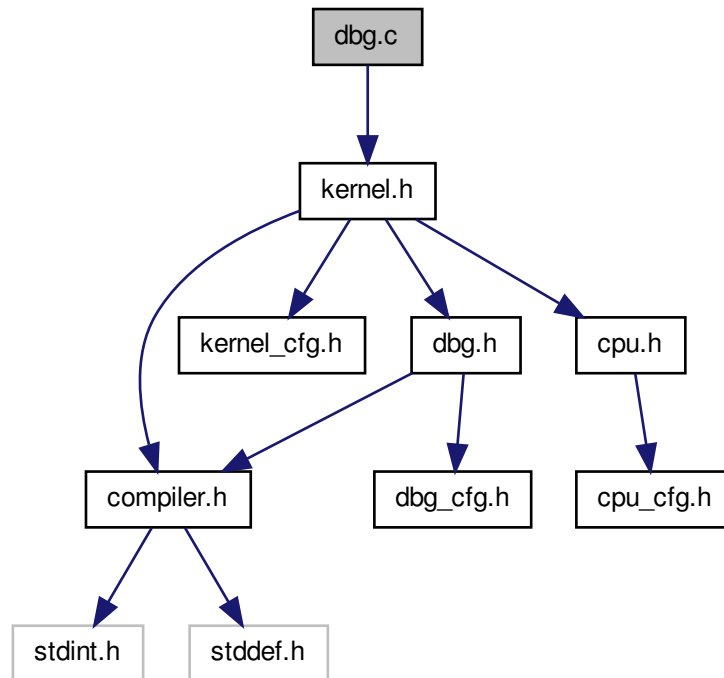
Nenad Radulovic

11.5 `dbg.c` File Reference

Implementation of Debug module.

```
#include "kernel.h"
```

Include dependency graph for dbg.c:



Functions

- `PORT_C_NORETURN void esDbgAssert (const char *fnName, const char *expr, enum esDbgMsg msg)`
An assertion has failed.

11.5.1 Detailed Description

Implementation of Debug module.

Author

Nenad Radulovic

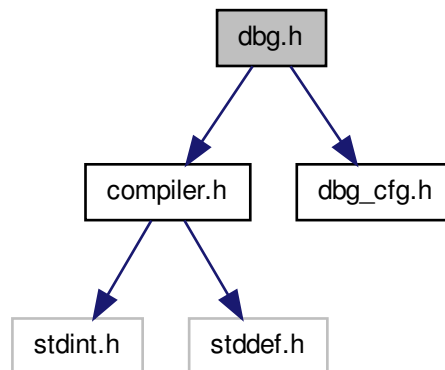
11.6 dbg.dox File Reference

11.7 dbg.h File Reference

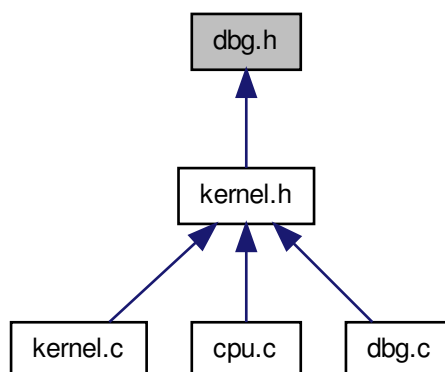
Debug basic functionality.

```
#include "compiler.h"
#include "dbg_cfg.h"
```

Include dependency graph for dbg.h:



This graph shows which files directly or indirectly include this file:



Macros

Internal checking

These macros are enabled/disabled using the option `CFG_DBG_INTERNAL_CHECK`.

- `#define ES_DBG_INTERNAL(msg, expr) ES_DBG_ASSERT(msg, expr)`
Assert macro used for internal execution checking.

Functions

Debug hook functions

Note

1) The definition of this functions must be written by the user.

- void [userAssert](#) (const char *fnName, const char *expr, const char *msg)
An assertion has failed. This function should inform the user about failed assertion.

Error checking

Some basic infrastructure for error checking

For more details see [Error checking](#).

- #define [ES_DBG_ASSERT](#)(msg, expr)
Generic assert macro.
- #define [ES_DBG_ASSERT_ALWAYS](#)(msg, text)
Assert macro that will always execute (no conditional).
- PORT_C_NORETURN void [esDbgAssert](#) (const char *fnName, const char *expr, enum [esDbgMsg](#) msg)
An assertion has failed.

API contract validation

These macros are enabled/disabled using the option [CFG_DBG_API_VALIDATION](#).

- #define [ES_DBG_API_OBLIGATION](#)(expr) expr
Execute code to fulfill the contract.
- #define [ES_DBG_API_REQUIRE](#)(msg, expr) [ES_DBG_ASSERT](#)(msg, expr)
Make sure the caller has fulfilled all contract preconditions.
- #define [ES_DBG_API_ENSURE](#)(msg, expr) [ES_DBG_ASSERT](#)(msg, expr)
Make sure the callee has fulfilled all contract postconditions.
- enum [esDbgMsg](#) {
 [ES_DBG_OUT_OF_RANGE](#),
 [ES_DBG_OBJECT_NOT_VALID](#),
 [ES_DBG_POINTER_NULL](#),
 [ES_DBG_USAGE_FAILURE](#),
 [ES_DBG_NOT_ENOUGH_MEM](#),
 [ES_DBG_UNKNOWN_ERROR](#) = 0xFFU }
Debug messages.

11.7.1 Detailed Description

Debug basic functionality.

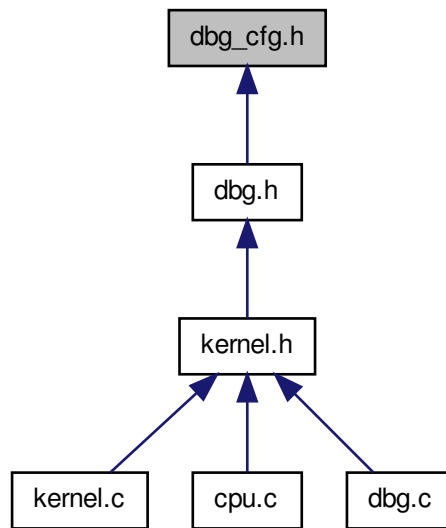
Author

Nenad Radulovic

11.8 dbg_cfg.h File Reference

Configuration of Debug.

This graph shows which files directly or indirectly include this file:



Macros

- `#define CFG_DBG_ENABLE 1U`
Enable/disable Debug module.
- `#define CFG_DBG_API_VALIDATION 1U`
Enable/disable API arguments validation.
- `#define CFG_DBG_INTERNAL_CHECK 1U`
Enable/disable internal checks.

11.8.1 Detailed Description

Configuration of Debug.

Author

Nenad Radulovic

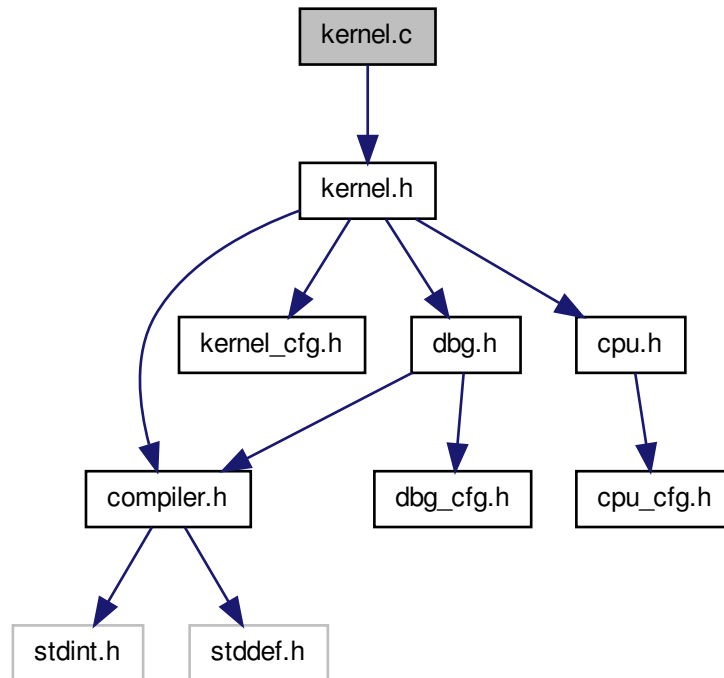
11.9 kernel-example.dox File Reference

11.10 kernel.c File Reference

Implementation of port independent code.


```
#include "kernel.h"
```

Include dependency graph for kernel.c:



Data Structures

- struct [sysTmr](#)
Main System Timer structure.

Macros

- #define [PRIO_BM_DATA_WIDTH_LOG2](#)
Priority Bit Map log base 2: $\log_2(\text{PORT_DATA_WIDTH_VAL})$
- #define [SCHED_STATE_INTSRV_MSK](#) (1U << 0)
Kernel state variable bit position which defines if the kernel is in interrupt servicing state.
- #define [SCHED_STATE_LOCK_MSK](#) (1U << 1)
Kernel state variable bit position which defines if the kernel is locked or not.
- #define [THD_CONTRACT_SIGNATURE](#) ((portReg_T)0xFEEDBEEFUL)
Thread structure signature.
- #define [THDQ_CONTRACT_SIGNATURE](#) ((portReg_T)0xFEEDBEEFUL)
Thread Queue structure signature.
- #define [VTMR_CONTRACT_SIGNATURE](#) ((portReg_T)0xFEEDBCCCUL)
Timer structure signature.
- #define [DLIST_IS_ENTRY_FIRST](#)(list, entry) ((entry) == (entry)->list.next)
DList macro: is the thread the first one in the list.
- #define [DLIST_IS_ENTRY_LAST](#)(list, entry) [DLIST_IS_ENTRY_FIRST](#)(list, entry)

- DList macro: is the thread the last one in the list.*
- #define [DLIST_IS_ENTRY_SINGLE](#)(list, entry) [DLIST_IS_ENTRY_FIRST](#)(list, entry)
- DList macro: is the thread single in the list.*
- #define [DLIST_ENTRY_NEXT](#)(list, entry) (entry)->list.next
- DList macro: get the next entry.*
- #define [DLIST_ENTRY_INIT](#)(list, entry)
- DList macro: initialize entry.*
- #define [DLIST_ENTRY_ADD_AFTER](#)(list, current, entry)
- DList macro: add new entry after current entry.*
- #define [DLIST_ENTRY_RM](#)(list, entry)
- DList macro: remove the entry from a list.*

Typedefs

Threads Queue

- typedef struct thdLSentinel [thdLSentinel_T](#)
- Thread list sentinel type.*

Functions

- void [esKernInit](#) (void)
- Initialize kernel internal data structures.*
- PORT_C_NORETURN void [esKernStart](#) (void)
- Start the multi-threading.*
- void [esKernSysTmr](#) (void)
- Process the system timer event.*
- void [esKernIsrPrologueI](#) (void)
- Enter Interrupt Service Routine.*
- void [esKernIsrEpilogueI](#) (void)
- Exit Interrupt Service Routine.*
- void [esThdInit](#) ([esThd_T](#) *thd, void(*fn)(void *), void *arg, [portStck_T](#) *stck, size_t stckSize, uint8_t prio)
- Initialize the specified thread.*
- void [esThdTerm](#) ([esThd_T](#) *thd)
- Terminate the specified thread.*
- void [esThdSetPriol](#) ([esThd_T](#) *thd, uint8_t prio)
- Set the priority of a thread.*
- void [esThdPostI](#) ([esThd_T](#) *thd)
- Post to thread semaphore.*
- void [esThdPost](#) ([esThd_T](#) *thd)
- Post to thread semaphore.*
- void [esThdWaitI](#) (void)
- Wait for thread semaphore.*
- void [esThdWait](#) (void)
- Wait for thread semaphore.*
- void [esThdQInit](#) ([esThdQ_T](#) *thdQ)
- Initialize Thread Queue.*
- void [esThdQTerm](#) ([esThdQ_T](#) *thdQ)
- Terminate Thread Queue.*
- void [esThdQAddI](#) ([esThdQ_T](#) *thdQ, [esThd_T](#) *thd)
- Add a thread to the Thread Queue.*

- void `esThdQRmI` (`esThdQ_T` *thdQ, `esThd_T` *thd)
Removes the thread from the Thread Queue.
- `esThd_T` * `esThdQFetchI` (const `esThdQ_T` *thdQ)
Fetch the first high priority thread from the Thread Queue.
- `esThd_T` * `esThdQFetchRotatel` (`esThdQ_T` *thdQ, `uint_fast8_t` prio)
Fetch the next thread and rotate thread linked list.
- `bool_T` `esThdQIsEmpty` (const `esThdQ_T` *thdQ)
Is thread queue empty.
- void `esSchedRdyAddI` (`esThd_T` *thd)
Add thread `thd` to the ready thread list and notify the scheduler.
- void `esSchedRdyRmI` (`esThd_T` *thd)
Remove thread `thd` from the ready thread list and notify the scheduler.
- void `esSchedYieldI` (void)
Force the scheduler invocation which will evaluate all ready threads and switch to ready thread with the highest priority.
- void `esSchedYieldIsrcI` (void)
Force the scheduler invocation which will evaluate all ready threads and switch to ready thread with the highest priority.
- void `esSchedLockEnterI` (void)
Lock the scheduler.
- void `esSchedLockExitI` (void)
Unlock the scheduler.
- void `esSchedLockEnter` (void)
Lock the scheduler.
- void `esSchedLockExit` (void)
Unlock the scheduler.
- void `esVTmrIInitI` (`esVTmr_T` *vTmr, `esTick_T` tick, void(*fn)(void *), void *arg)
Add and start a new virtual timer.
- void `esVTmrIInit` (`esVTmr_T` *vTmr, `esTick_T` tick, void(*fn)(void *), void *arg)
Add and start a new virtual timer.
- void `esVTmrTermI` (`esVTmr_T` *vTmr)
Cancel and remove a virtual timer.
- void `esVTmrTerm` (`esVTmr_T` *vTmr)
Cancel and remove a virtual timer.
- void `esVTmrDelay` (`esTick_T` tick)
Delay for specified amount of ticks.

Virtual Timer and Virtual Timer kernel thread

- static `PORT_C_INLINE` void `vTmrSleep` (`esTick_T` ticks)
Set up system timer for different tick period during sleeping.
- static `PORT_C_INLINE` void `vTmrEvaluatel` (void)
Evaluate armed virtual timers.
- static void `vTmrAddArmedS` (`esVTmr_T` *vTmr)
Add a virtual timer into sorted list.
- static `PORT_C_INLINE` void `vTmrImportPendI` (void)
Import timers from pending list to armed list.
- static void `vTmrImportPend` (void)
Import timers from pending list to armed list.
- static void `kVTmrInit` (void)
Initialization of Virtual Timer kernel thread.
- static void `kVTmr` (void *arg)
Virtual Timer thread code.

Variables

- static uint_fast8_t [gKernLockCnt](#)
Kernel Lock Counter.
- const volatile [esKernCtrl_T](#) [gKernCtrl](#)
Kernel control initialization.

System timer

- typedef struct [sysTmr](#) [sysTmr_T](#)
System Timer type.
- static [sysTmr_T](#) [gSysTmr](#)
Main System Timer structure.
- static [esVTmr_T](#) [gVTmrArmed](#)
List of virtual timers to armed expire.
- static [esVTmr_T](#) [gVTmrPend](#)
Virtual timers pending to be inserted into waiting list.
- static [esThd_T](#) [gKVTmr](#)
Virtual timer thread ID.
- static [PORT_C_INLINE](#) void [sysTmrInit](#) (void)
Initialize system timer hardware.
- static [PORT_C_INLINE](#) void [sysTmrActivate](#) (void)
Try to activate system timer.
- static [PORT_C_INLINE](#) void [sysTmrDeactivate](#) (void)
Try to deactivate system timer.

Priority Bit Map

- typedef struct [prioBM](#) [prioBM_T](#)
Priority Bit Map type.
- static [PORT_C_INLINE](#) void [prioBMInit](#) ([prioBM_T](#) *bm)
Initialize bitmap.
- static [PORT_C_INLINE](#) void [prioBMSet](#) ([prioBM_T](#) *bm, uint_fast8_t prio)
Set the bit corresponding to the prio argument.
- static [PORT_C_INLINE](#) void [prioBMClear](#) ([prioBM_T](#) *bm, uint_fast8_t prio)
Clear the bit corresponding to the prio argument.
- static [PORT_C_INLINE](#) uint_fast8_t [prioBMGet](#) (const [prioBM_T](#) *bm)
Get the highest priority set.
- static [PORT_C_INLINE](#) bool_T [prioBMIsEmpty](#) (const [prioBM_T](#) *bm)
Is bit map empty?

Scheduler

- static [esThdQ_T](#) [gRdyQueue](#)
Ready Thread queue.
- static [PORT_C_INLINE](#) void [schedInit](#) (void)
Initialize Ready Thread Queue structure [gRdyQueue](#) and Kernel control structure [esKernCtrl](#).
- static [PORT_C_INLINE](#) void [schedStart](#) (void)
Set the scheduler data structures for multi-threading.
- static [PORT_C_INLINE](#) void [schedSleep](#) (void)
Set the scheduler to sleep.

- static `PORT_C_INLINE` void `schedWakeUpI` (void)
Wake up the scheduler.
- static `PORT_C_INLINE` void `schedRdyAddInitI` (`esThd_T` *thd)
Initialize scheduler ready structure during the thread add operation.
- static `PORT_C_INLINE` void `schedQmNextI` (void)
Fetch and try to schedule the next thread of the same priority as the current thread.
- static `PORT_C_INLINE` void `schedQmI` (void)
Do the Quantum (Round-Robin) scheduling.

Idle kernel thread

- static `esThd_T` `gKIdle`
Idle thread ID.
- static void `kIdleInit` (void)
Initialization of Idle thread.
- static void `kIdle` (void *arg)
Idle thread code.

11.10.1 Detailed Description

Implementation of port independent code.

Author

Nenad Radulovic

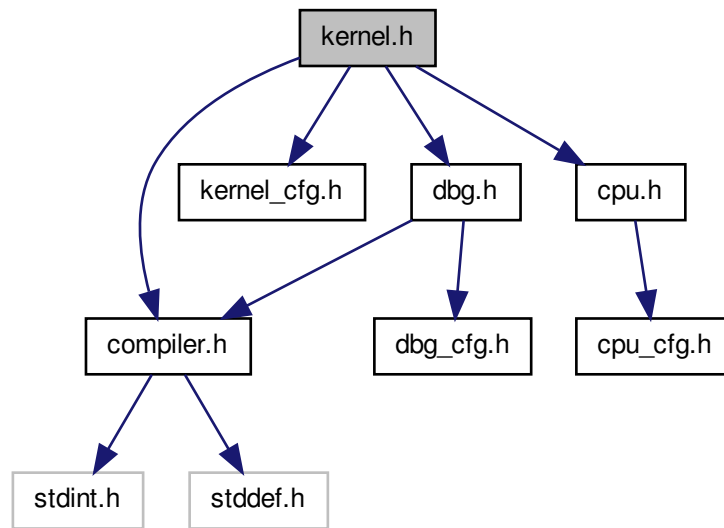
11.11 kernel.dox File Reference

11.12 kernel.h File Reference

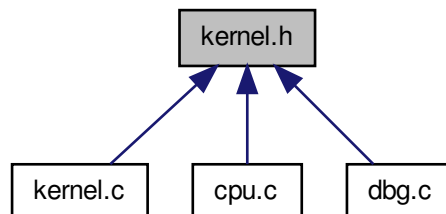
Interface of kernel.

```
#include "compiler.h"
#include "kernel_cfg.h"
#include "cpu.h"
#include "dbg.h"
```

Include dependency graph for kernel.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [esThd](#)
Thread structure.
- struct [esThd::thdL](#)
Thread linked List structure.
- struct [esVTmr](#)
Virtual Timer structure.
- struct [esVTmr::tmrL](#)
Virtual Timer linked list structure.
- struct [esThdQ](#)

- *Thread Queue structure.*
- struct [esThdQ::prioBM](#)
Priority Bit Map structure.
- struct [esThdQ::thdLSentinel](#)
Thread linked list sentinel structure.
- struct [esKernCtrl](#)
Kernel control block structure.

Macros

Kernel identification and version number

- #define [ES_KERN_VER](#) 0x10000UL
Identifies the underlying kernel version number.
- #define [ES_KERN_ID](#) "eSolid Kernel v1.0"
Kernel identification string.

Critical section management

These macros are used to prevent interrupts on entry into the critical section, and restoring interrupts to their previous state on exit from critical section.

For more details see [Critical sections](#).

- #define [ES_CRITICAL_DECL\(\)](#) [PORT_CRITICAL_DECL\(\)](#)
Critical section status variable declaration.
- #define [ES_CRITICAL_ENTER\(\)](#) [PORT_CRITICAL_ENTER\(\)](#)
Enter a critical section.
- #define [ES_CRITICAL_EXIT\(\)](#) [PORT_CRITICAL_EXIT\(\)](#)
Exit from critical section.
- #define [ES_CRITICAL_ENTER_LOCK_EXIT\(\)](#)
Enter critical section and exit scheduler lock.
- #define [ES_CRITICAL_EXIT_LOCK_ENTER\(\)](#)
Exit critical section and enter scheduler lock.

Functions

General kernel functions

There are several groups of functions:

- *kernel initialization and start*
- *ISR prologue and epilogue*
- void [esKernInit](#) (void)
Initialize kernel internal data structures.
- [PORT_C_NORETURN](#) void [esKernStart](#) (void)
Start the multi-threading.
- void [esKernSysTmr](#) (void)
Process the system timer event.
- void [esKernIsrPrologueI](#) (void)
Enter Interrupt Service Routine.
- void [esKernIsrEpilogueI](#) (void)
Exit Interrupt Service Routine.

Scheduler notification and invocation

- void [esSchedRdyAddI](#) ([esThd_T](#) *thd)
Add thread `thd` to the ready thread list and notify the scheduler.

- void [esSchedRdyRml](#) ([esThd_T](#) *thd)
Remove thread `thd` from the ready thread list and notify the scheduler.
- void [esSchedYieldl](#) (void)
Force the scheduler invocation which will evaluate all ready threads and switch to ready thread with the highest priority.
- void [esSchedYieldlsl](#) (void)
Force the scheduler invocation which will evaluate all ready threads and switch to ready thread with the highest priority.
- void [esSchedLockEnterl](#) (void)
Lock the scheduler.
- void [esSchedLockExitl](#) (void)
Unlock the scheduler.
- void [esSchedLockEnter](#) (void)
Lock the scheduler.
- void [esSchedLockExit](#) (void)
Unlock the scheduler.

Kernel hook functions

Note

1) The definition of this functions must be written by the user.

- void [userPreSysTmr](#) (void)
System timer hook function, called from system system timer ISR function before the kernel functions.
- void [userPreKernInit](#) (void)
Kernel initialization hook function, called from [esKernInit\(\)](#) function before kernel initialization.
- void [userPostKernInit](#) (void)
Kernel initialization hook function, called from [esKernInit\(\)](#) function after kernel initialization.
- void [userPreKernStart](#) (void)
Kernel start hook function, called from [esKernStart\(\)](#) function.
- void [userPostThdInit](#) ([esThd_T](#) *thd)
Thread initialization end hook function, called from [esThdInit\(\)](#) function.
- void [userPreThdTerm](#) (void)
Thread terminate hook function, called from [esThdTerm\(\)](#) or when a thread terminates itself.
- void [userPreIdle](#) (void)
Pre idle hook function, called from idle thread, just before entering idle period.
- void [userPostIdle](#) (void)
Post idle hook function, called from idle thread, just after exiting idle period.
- void [userPreCtxSw](#) ([esThd_T](#) *oldThd, [esThd_T](#) *newThd)
Kernel context switch hook function, called from [esSchedYieldl\(\)](#) and [esSchedYieldlsl\(\)](#) functions just before context switch.

Thread management

Basic thread management services

For more details see [Thread Management](#).

- `#define` [ES_STCK_SIZE](#)(elem) [PORT_STCK_SIZE](#)(elem)
Converts the required stack elements into the stack array index.
- `typedef struct` [esThd](#) [esThd_T](#)
Thread type.
- `typedef` [portStck_T](#) [esStck_T](#)
Stack type.
- void [esThdInit](#) ([esThd_T](#) *thd, void(*fn)(void *), void *arg, [portStck_T](#) *stck, `size_t` stckSize, `uint8_t` prio)
Initialize the specified thread.
- void [esThdTerm](#) ([esThd_T](#) *thd)

- Terminate the specified thread.*

 - static `PORT_C_INLINE esThd_T * esThdGetId` (void)

Get the current thread ID.
- static `PORT_C_INLINE uint8_t esThdGetPrio` (esThd_T *thd)

Get the priority of a thread.
- void `esThdSetPrio` (esThd_T *thd, uint8_t prio)

Set the priority of a thread.
- void `esThdPostl` (esThd_T *thd)

Post to thread semaphore.
- void `esThdPost` (esThd_T *thd)

Post to thread semaphore.
- void `esThdWaitl` (void)

Wait for thread semaphore.
- void `esThdWait` (void)

Wait for thread semaphore.

Virtual Timer management

- typedef uint_fast32_t `esTick_T`

Timer tick type.
- typedef struct `esVTmr esVTmr_T`

Virtual Timer type.
- void `esVTmrInitl` (esVTmr_T *vTmr, esTick_T tick, void(*fn)(void *), void *arg)

Add and start a new virtual timer.
- void `esVTmrInit` (esVTmr_T *vTmr, esTick_T tick, void(*fn)(void *), void *arg)

Add and start a new virtual timer.
- void `esVTmrTerml` (esVTmr_T *vTmr)

Cancel and remove a virtual timer.
- void `esVTmrTerm` (esVTmr_T *vTmr)

Cancel and remove a virtual timer.
- void `esVTmrDelay` (esTick_T tick)

Delay for specified amount of ticks.

Thread Queue management

- #define `PRIOR_BM_GRP_IDX` ((CFG_SCHED_Prio_LVL + PORT_DATA_WIDTH_VAL - 1U) / PORT_DATA_WIDTH_VAL)

Priority Bit Map Group Index.
- typedef struct `esThdQ esThdQ_T`

Thread queue type.
- void `esThdQInit` (esThdQ_T *thdQ)

Initialize Thread Queue.
- void `esThdQTerm` (esThdQ_T *thdQ)

Terminate Thread Queue.
- void `esThdQAddl` (esThdQ_T *thdQ, esThd_T *thd)

Add a thread to the Thread Queue.
- void `esThdQRml` (esThdQ_T *thdQ, esThd_T *thd)

Removes the thread from the Thread Queue.
- esThd_T * `esThdQFetchl` (const esThdQ_T *thdQ)

Fetch the first high priority thread from the Thread Queue.
- esThd_T * `esThdQFetchRotatel` (esThdQ_T *thdQ, uint_fast8_t prio)

Fetch the next thread and rotate thread linked list.

- `bool_T esThdQIsEmpty` (const `esThdQ_T *thdQ`)
Is thread queue empty.

Kernel control block

- enum `esKernState` {
 `ES_KERN_RUN` = 0x00U,
 `ES_KERN_INTSRV_RUN` = 0x01U,
 `ES_KERN_LOCK` = 0x02U,
 `ES_KERN_INTSRV_LOCK` = 0x03U,
 `ES_KERN_SLEEP` = 0x06U,
 `ES_KERN_INIT` = 0x08U,
 `ES_KERN_INACTIVE` = 0x10U }
Kernel state enumeration.
- typedef enum `esKernState esKernState_T`
Kernel state type.
- typedef struct `esKernCtrl esKernCtrl_T`
Kernel control block type.
- const volatile `esKernCtrl_T gKernCtrl`
Kernel control block.

11.12.1 Detailed Description

Interface of kernel.

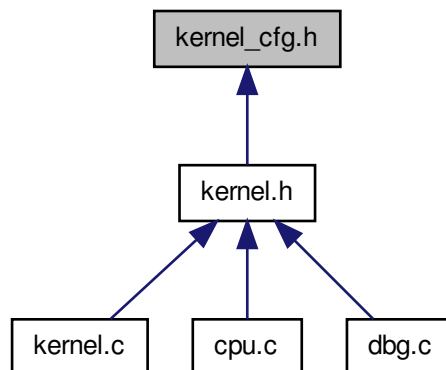
Author

Nenad Radulovic

11.13 kernel_cfg.h File Reference

Configuration of Kernel.

This graph shows which files directly or indirectly include this file:



Macros

Kernel configuration options and settings

- #define `CFG_SCHED_PRIO_LVL` 8U
Scheduler priority levels.
- #define `CFG_SCHED_TIME_QUANTUM` 10U
Scheduler Round-Robin time quantum.
- #define `CFG_SCHED_POWER_SAVE` 0U
Enable/disable scheduler power savings mode.
- #define `CFG_SYSTMTR_ADAPTIVE_MODE` 0U
System timer adaptive mode.
- #define `CFG_SYSTMTR_EVENT_FREQUENCY` 100UL
The frequency of system timer tick event.
- #define `CFG_SYSTMTR_TICK_TYPE` 2U
The size of the system timer tick event counter.

Kernel pre hooks

- #define `CFG_HOOK_PRE_SYSTMTR_EVENT` 0U
System timer event hook function.
- #define `CFG_HOOK_PRE_KERN_INIT` 0U
Pre kernel initialization hook function.
- #define `CFG_HOOK_POST_KERN_INIT` 0U
Post kernel initialization hook function.
- #define `CFG_HOOK_PRE_KERN_START` 0U
Pre kernel start hook function.
- #define `CFG_HOOK_POST_THD_INIT` 0U
Post thread initialization hook function.
- #define `CFG_HOOK_PRE_THD_TERM` 0U
Pre thread termination hook function.
- #define `CFG_HOOK_PRE_IDLE` 0U
Pre idle hook function.
- #define `CFG_HOOK_POST_IDLE` 0U
Post idle hook function.
- #define `CFG_HOOK_PRE_CTX_SW` 0U
Pre context switch hook function.

11.13.1 Detailed Description

Configuration of Kernel.

Author

Nenad Radulovic

11.14 kernel_cfg.h File Reference

Configuration of Kernel - Template.

Macros

Kernel configuration options and settings

Kernel default configuration

- #define `CFG_API_VALIDATION` 1U
Enable/disable API arguments validation.

- #define `CFG_SCHED_PRIO_LVL` 8U
Scheduler priority levels.
- #define `CFG_SCHED_TIME_QUANTUM` 10U
Scheduler Round-Robin time quantum.
- #define `CFG_SCHED_POWER_SAVE` 0U
Enable/disable scheduler power savings mode.
- #define `CFG_SYSTMTR_ADAPTIVE_MODE` 0U
System timer mode.
- #define `CFG_SYSTMTR_EVENT_FREQUENCY` 100UL
The frequency of system tick event.
- #define `CFG_SYSTMTR_TICK_TYPE` 2U
The size of the system timer counter.

Kernel hooks

- #define `CFG_HOOK_SYSTMTR_EVENT` 0U
System timer event hook function.
- #define `CFG_HOOK_KERN_INIT` 0U
Kernel initialization hook function.
- #define `CFG_HOOK_KERN_START` 0U
Kernel start hook function.
- #define `CFG_HOOK_THD_INIT_END` 0U
Thread initialization hook function.
- #define `CFG_HOOK_THD_TERM` 0U
Thread termination hook function.
- #define `CFG_HOOK_CTX_SW` 0U
Kernel context switch hook function.

11.14.1 Detailed Description

Configuration of Kernel - Template.

Author

Nenad Radulovic

Index

- arg
 - esVTmr, [102](#)
- bit
 - esThdQ::prioBM, [104](#)
- bitGrp
 - esThdQ::prioBM, [104](#)
- bool_T
 - Compiler port, [66](#)
- boolType
 - Compiler port, [67](#)
- CFG_API_VALIDATION
 - Kernel configuration, [83](#)
- CFG_CRITICAL_PRIO
 - CPU port configuration, [81](#)
- CFG_DBG_ENABLE
 - Configuration, [92](#)
- CFG_HOOK_CTX_SW
 - Kernel configuration, [85](#)
- CFG_HOOK_KERN_INIT
 - Kernel configuration, [84](#)
- CFG_HOOK_POST_IDLE
 - Configuration, [62](#)
- CFG_HOOK_PRE_IDLE
 - Configuration, [62](#)
- CFG_HOOK_THD_TERM
 - Kernel configuration, [85](#)
- CFG_SCHED_PRIO_LVL
 - Configuration, [61](#)
 - Kernel configuration, [83](#)
- CPU port configuration, [81](#)
 - CFG_CRITICAL_PRIO, [81](#)
- CPU port interface, [68](#)
 - gPortIsrNesting_, [78](#)
 - PORT_CRITICAL_DECL, [72](#)
 - PORT_CRITICAL_EXIT, [73](#)
 - PORT_CTX_INIT, [74](#)
 - PORT_CTX_SW, [74](#)
 - PORT_CTX_SW_ISR, [74](#)
 - PORT_INIT, [75](#)
 - PORT_INIT_EARLY, [75](#)
 - PORT_INIT_LATE, [75](#)
 - PORT_INT_DISABLE, [72](#)
 - PORT_ISR_ENTER, [72](#)
 - PORT_ISR_EXIT, [72](#)
 - PORT_ISR_IS_LAST, [72](#)
 - PORT_PWR2, [73](#)
 - PORT_STCK_SIZE, [74](#)
 - PORT_SYSTMTR_ENABLE, [73](#)
 - PORT_SYSTMTR_INIT, [73](#)
 - PORT_SYSTMTR_RELOAD, [73](#)
 - PORT_SYSTMTR_TERM, [73](#)
 - PORT_THD_START, [74](#)
 - portCtxInit_, [77](#)
 - portCtxSw_, [77](#)
 - portCtxSwIsr_, [77](#)
 - portFindLastSet_, [76](#)
 - portInit_, [77](#)
 - portInitEarly_, [77](#)
 - portInitLate_, [77](#)
 - portIntDisable_, [75](#)
 - portIntGet_, [75](#)
 - portIntSet_, [76](#)
 - portReg_T, [75](#)
 - portStck_T, [75](#)
 - portSysTmrDisable_, [76](#)
 - portSysTmrEnable_, [76](#)
 - portSysTmrInit_, [76](#)
 - portSysTmrIsrDisable_, [77](#)
 - portSysTmrIsrEnable_, [77](#)
 - portSysTmrReload_, [76](#)
 - portSysTmrTerm_, [76](#)
 - portThdStart_, [77](#)
 - pwr2LKP, [78](#)
- CPU port internals, [79](#)
 - gPortIsrNesting_, [80](#)
 - portCtxInit_, [79](#)
 - portFindLastSet_, [79](#)
 - pwr2LKP, [80](#)
- Compiler port, [64](#)
 - bool_T, [66](#)
 - boolType, [67](#)
 - FALSE, [67](#)
 - PORT_C_ALIGNED, [66](#)
 - PORT_C_FUNC, [66](#)
 - PORT_C_INLINE, [65](#)
 - PORT_C_NAKED, [66](#)
 - PORT_C_WEAK, [66](#)
 - PORT_HWREG_SET, [66](#)
 - TRUE, [67](#)
- compiler.h, [110](#)
- Configuration, [60](#), [92](#)
 - CFG_DBG_ENABLE, [92](#)
 - CFG_HOOK_POST_IDLE, [62](#)
 - CFG_HOOK_PRE_IDLE, [62](#)
 - CFG_SCHED_PRIO_LVL, [61](#)
- cprio
 - esThd, [97](#)
- cpu.c, [113](#)
- cpu.h, [114](#)
- cpu_cfg.h, [118](#)
- cthd
 - esKernCtrl, [95](#)
- DLIST_ENTRY_INIT
 - Internals, [42](#)
- DLIST_ENTRY_NEXT
 - Internals, [42](#)
- DLIST_ENTRY_RM
 - Internals, [42](#)

DLIST_IS_ENTRY_LAST
 Internals, 42
dbg.c, 118
dbg.dox, 119
dbg.h, 119
dbg_cfg.h, 121
Debug, 86

ES_DBG_NOT_ENOUGH_MEM
 Interface, 89
ES_DBG_OBJECT_NOT_VALID
 Interface, 89
ES_DBG_OUT_OF_RANGE
 Interface, 89
ES_DBG_POINTER_NULL
 Interface, 89
ES_DBG_UNKNOWN_ERROR
 Interface, 89
ES_DBG_USAGE_FAILURE
 Interface, 89
ES_KERN_INACTIVE
 Interface, 21
ES_KERN_INIT
 Interface, 21
ES_KERN_INTSRV_LOCK
 Interface, 21
ES_KERN_INTSRV_RUN
 Interface, 21
ES_KERN_LOCK
 Interface, 21
ES_KERN_RUN
 Interface, 21
ES_KERN_SLEEP
 Interface, 21
ES_CRITICAL_DECL
 Interface, 20
ES_CRITICAL_ENTER
 Interface, 20
ES_CRITICAL_EXIT
 Interface, 20
ES_DBG_API_ENSURE
 Interface, 89
ES_DBG_API_REQUIRE
 Interface, 89
ES_DBG_ASSERT
 Interface, 88
ES_DBG_INTERNAL
 Interface, 88
ES_KERN_ID
 Interface, 20
ES_KERN_VER
 Interface, 20
ES_STCK_SIZE
 Interface, 20
esDbgAssert
 Interface, 89
 Internals, 91
esDbgMsg
 Interface, 89

esKernCtrl, 94
 cthd, 95
 pthd, 95
 state, 95
esKernCtrl_T
 Interface, 21
esKernInit
 Interface, 22
 Internals, 46
esKernIsrEpiloguel
 Interface, 23
 Internals, 47
esKernIsrProloguel
 Interface, 22
 Internals, 47
esKernStart
 Interface, 22
 Internals, 46
esKernState
 Interface, 21
esKernState_T
 Interface, 21
esKernSysTmr
 Interface, 22
 Internals, 46
esSchedLockEnter
 Interface, 31
 Internals, 55
esSchedLockEnterI
 Interface, 31
 Internals, 54
esSchedLockExit
 Interface, 32
 Internals, 55
esSchedLockExitI
 Interface, 31
 Internals, 55
esSchedRdyAddI
 Interface, 30
 Internals, 53
esSchedRdyRml
 Interface, 30
 Internals, 54
esSchedYieldI
 Interface, 31
 Internals, 54
esSchedYieldIsrI
 Interface, 31
 Internals, 54
esStck_T
 Interface, 21
esThd, 95
 cprio, 97
 prio, 97
 qCnt, 97
 qRId, 97
 signature, 97
 stck, 97

- thdL, [97](#)
- esThd::thdL, [106](#)
 - next, [107](#)
 - prev, [107](#)
 - q, [107](#)
- esThd_T
 - Interface, [21](#)
- esThdGetId
 - Interface, [25](#)
- esThdGetPrio
 - Interface, [25](#)
- esThdInit
 - Interface, [23](#)
 - Internals, [47](#)
- esThdPost
 - Interface, [26](#)
 - Internals, [49](#)
- esThdPostI
 - Interface, [26](#)
 - Internals, [49](#)
- esThdQ, [98](#)
 - grp, [100](#)
 - prioOcc, [100](#)
 - signature, [100](#)
- esThdQ::prioBM, [103](#)
 - bit, [104](#)
 - bitGrp, [104](#)
- esThdQ::thdLSentinel, [107](#)
 - head, [109](#)
 - next, [109](#)
- esThdQ_T
 - Interface, [21](#)
- esThdQAddI
 - Interface, [28](#)
 - Internals, [51](#)
- esThdQFetchI
 - Interface, [28](#)
 - Internals, [52](#)
- esThdQFetchRotatel
 - Interface, [29](#)
 - Internals, [52](#)
- esThdQInit
 - Interface, [27](#)
 - Internals, [50](#)
- esThdQIsEmpty
 - Interface, [29](#)
 - Internals, [53](#)
- esThdQRml
 - Interface, [28](#)
 - Internals, [51](#)
- esThdQTerm
 - Interface, [27](#)
 - Internals, [51](#)
- esThdSetPrioI
 - Interface, [25](#)
 - Internals, [49](#)
- esThdTerm
 - Interface, [24](#)
- Internals, [48](#)
- esThdWait
 - Interface, [27](#)
 - Internals, [50](#)
- esThdWaitI
 - Interface, [26](#)
 - Internals, [50](#)
- esTick_T
 - Interface, [21](#)
- esVTmr, [100](#)
 - arg, [102](#)
 - fn, [102](#)
 - rtick, [102](#)
 - signature, [102](#)
 - tmrL, [102](#)
- esVTmr::tmrL, [109](#)
 - next, [110](#)
 - prev, [110](#)
 - q, [110](#)
- esVTmr_T
 - Interface, [21](#)
- esVTmrDelay
 - Interface, [34](#)
 - Internals, [57](#)
- esVTmrInit
 - Interface, [32](#)
 - Internals, [56](#)
- esVTmrInitI
 - Interface, [32](#)
 - Internals, [55](#)
- esVTmrTerm
 - Interface, [33](#)
 - Internals, [57](#)
- esVTmrTermI
 - Interface, [33](#)
 - Internals, [56](#)
- FALSE
 - Compiler port, [67](#)
- fn
 - esVTmr, [102](#)
- gKIdle
 - Internals, [58](#)
- gKVtmr
 - Internals, [58](#)
- gKernCtrl
 - Interface, [36](#)
 - Internals, [59](#)
- gKernLockCnt
 - Internals, [58](#)
- gPortIsrNesting_
 - CPU port interface, [78](#)
 - CPU port internals, [80](#)
- gRdyQueue
 - Internals, [58](#)
- gSysTmr
 - Internals, [58](#)
- gVTmrArmed

- Internals, 58
- gVTmrPend
 - Internals, 58
- grp
 - esThdQ, 100
- head
 - esThdQ::thdLSentinel, 109
- Interface, 16, 87
 - ES_DBG_NOT_ENOUGH_MEM, 89
 - ES_DBG_OBJECT_NOT_VALID, 89
 - ES_DBG_OUT_OF_RANGE, 89
 - ES_DBG_POINTER_NULL, 89
 - ES_DBG_UNKNOWN_ERROR, 89
 - ES_DBG_USAGE_FAILURE, 89
 - ES_KERN_INACTIVE, 21
 - ES_KERN_INIT, 21
 - ES_KERN_INTSRV_LOCK, 21
 - ES_KERN_INTSRV_RUN, 21
 - ES_KERN_LOCK, 21
 - ES_KERN_RUN, 21
 - ES_KERN_SLEEP, 21
 - ES_CRITICAL_DECL, 20
 - ES_CRITICAL_ENTER, 20
 - ES_CRITICAL_EXIT, 20
 - ES_DBG_API_ENSURE, 89
 - ES_DBG_API_REQUIRE, 89
 - ES_DBG_ASSERT, 88
 - ES_DBG_INTERNAL, 88
 - ES_KERN_ID, 20
 - ES_KERN_VER, 20
 - ES_STCK_SIZE, 20
 - esDbgAssert, 89
 - esDbgMsg, 89
 - esKernCtrl_T, 21
 - esKernInit, 22
 - esKernIsrEpiloguel, 23
 - esKernIsrProloguel, 22
 - esKernStart, 22
 - esKernState, 21
 - esKernState_T, 21
 - esKernSysTmr, 22
 - esSchedLockEnter, 31
 - esSchedLockEnterI, 31
 - esSchedLockExit, 32
 - esSchedLockExitI, 31
 - esSchedRdyAddI, 30
 - esSchedRdyRmI, 30
 - esSchedYieldI, 31
 - esSchedYieldIsrI, 31
 - esStck_T, 21
 - esThd_T, 21
 - esThdGetId, 25
 - esThdGetPrio, 25
 - esThdInit, 23
 - esThdPost, 26
 - esThdPostI, 26
 - esThdQ_T, 21
 - esThdQAddI, 28
 - esThdQFetchI, 28
 - esThdQFetchRotatel, 29
 - esThdQInit, 27
 - esThdQIsEmpty, 29
 - esThdQRmI, 28
 - esThdQTerm, 27
 - esThdSetPrioI, 25
 - esThdTerm, 24
 - esThdWait, 27
 - esThdWaitI, 26
 - esTick_T, 21
 - esVTmr_T, 21
 - esVTmrDelay, 34
 - esVTmrInit, 32
 - esVTmrInitI, 32
 - esVTmrTerm, 33
 - esVTmrTermI, 33
 - gKernCtrl, 36
 - PRIO_BM_GRP_INDX, 20
 - userAssert, 90
 - userPostIdle, 35
 - userPostKernInit, 34
 - userPostThdInit, 35
 - userPreCtxSw, 35
 - userPreIdle, 35
 - userPreKernInit, 34
 - userPreKernStart, 34
 - userPreSysTmr, 34
 - userPreThdTerm, 35
- Internals, 37, 91
 - DLIST_ENTRY_INIT, 42
 - DLIST_ENTRY_NEXT, 42
 - DLIST_ENTRY_RM, 42
 - DLIST_IS_ENTRY_LAST, 42
 - esDbgAssert, 91
 - esKernInit, 46
 - esKernIsrEpiloguel, 47
 - esKernIsrProloguel, 47
 - esKernStart, 46
 - esKernSysTmr, 46
 - esSchedLockEnter, 55
 - esSchedLockEnterI, 54
 - esSchedLockExit, 55
 - esSchedLockExitI, 55
 - esSchedRdyAddI, 53
 - esSchedRdyRmI, 54
 - esSchedYieldI, 54
 - esSchedYieldIsrI, 54
 - esThdInit, 47
 - esThdPost, 49
 - esThdPostI, 49
 - esThdQAddI, 51
 - esThdQFetchI, 52
 - esThdQFetchRotatel, 52
 - esThdQInit, 50
 - esThdQIsEmpty, 53
 - esThdQRmI, 51

- esThdQTerm, [51](#)
 - esThdSetPriol, [49](#)
 - esThdTerm, [48](#)
 - esThdWait, [50](#)
 - esThdWaitl, [50](#)
 - esVTmrDelay, [57](#)
 - esVTmrInit, [56](#)
 - esVTmrInitl, [55](#)
 - esVTmrTerm, [57](#)
 - esVTmrTermI, [56](#)
 - gKIdle, [58](#)
 - gKVTmr, [58](#)
 - gKernCtrl, [59](#)
 - gKernLockCnt, [58](#)
 - gRdyQueue, [58](#)
 - gSysTmr, [58](#)
 - gVTmrArmed, [58](#)
 - gVTmrPend, [58](#)
 - kIdle, [45](#)
 - kIdleInit, [45](#)
 - kVTmr, [45](#)
 - kVTmrInit, [45](#)
 - prioBM_T, [42](#)
 - prioBMClear, [43](#)
 - prioBMGet, [43](#)
 - prioBMInit, [43](#)
 - prioBMIsEmpty, [43](#)
 - prioBMSet, [43](#)
 - schedInit, [43](#)
 - schedQml, [44](#)
 - schedQmNextl, [44](#)
 - schedRdyAddInitl, [44](#)
 - schedSleep, [44](#)
 - schedStart, [44](#)
 - schedWakeUpl, [44](#)
 - sysTmr_T, [42](#)
 - sysTmrActivate, [44](#)
 - sysTmrDeactivateI, [44](#)
 - sysTmrInit, [44](#)
 - thdLSentinel_T, [42](#)
 - vTmrAddArmedS, [45](#)
 - vTmrEvaluatel, [45](#)
 - vTmrImportPend, [45](#)
 - vTmrImportPendI, [45](#)
 - vTmrSleep, [45](#)
- kIdle
- Internals, [45](#)
- kIdleInit
- Internals, [45](#)
- kVTmr
- Internals, [45](#)
- kVTmrInit
- Internals, [45](#)
- Kernel, [15](#)
- Kernel configuration, [82](#)
- CFG_API_VALIDATION, [83](#)
 - CFG_HOOK_CTX_SW, [85](#)
 - CFG_HOOK_KERN_INIT, [84](#)
 - CFG_HOOK_THD_TERM, [85](#)
 - CFG_SCHED_PRIO_LVL, [83](#)
- kernel-example.dox, [122](#)
- kernel.c, [122](#)
- kernel.dox, [127](#)
- kernel.h, [127](#)
- kernel_cfg.h, [132](#), [133](#)
- next
- esThd::thdL, [107](#)
 - esThdQ::thdLSentinel, [109](#)
 - esVTmr::tmrL, [110](#)
- PORT_C_ALIGNED
- Compiler port, [66](#)
- PORT_C_FUNC
- Compiler port, [66](#)
- PORT_C_INLINE
- Compiler port, [65](#)
- PORT_C_NAKED
- Compiler port, [66](#)
- PORT_C_WEAK
- Compiler port, [66](#)
- PORT_CRITICAL_DECL
- CPU port interface, [72](#)
- PORT_CRITICAL_ENTER
- CPU port interface, [72](#)
- PORT_CRITICAL_EXIT
- CPU port interface, [73](#)
- PORT_CTX_INIT
- CPU port interface, [74](#)
- PORT_CTX_SW
- CPU port interface, [74](#)
- PORT_CTX_SW_ISR
- CPU port interface, [74](#)
- PORT_FIND_LAST_SET
- CPU port interface, [73](#)
- PORT_HWREG_SET
- Compiler port, [66](#)
- PORT_INIT
- CPU port interface, [75](#)
- PORT_INIT_EARLY
- CPU port interface, [75](#)
- PORT_INIT_LATE
- CPU port interface, [75](#)
- PORT_INT_DISABLE
- CPU port interface, [72](#)
- PORT_ISR_ENTER
- CPU port interface, [72](#)
- PORT_ISR_EXIT
- CPU port interface, [72](#)
- PORT_ISR_IS_LAST
- CPU port interface, [72](#)
- PORT_PWR2
- CPU port interface, [73](#)
- PORT_STCK_SIZE
- CPU port interface, [74](#)
- PORT_SYSTMTR_DISABLE
- CPU port interface, [74](#)

PORT_SYSTMR_ENABLE
 CPU port interface, 73
 PORT_SYSTMR_INIT
 CPU port interface, 73
 PORT_SYSTMR_RELOAD
 CPU port interface, 73
 PORT_SYSTMR_TERM
 CPU port interface, 73
 PORT_THD_START
 CPU port interface, 74
 PRIO_BM_GRP_IDX
 Interface, 20
 Port template, 63
 portCtx, 102
 r0, 103
 portCtxInit_
 CPU port interface, 77
 CPU port internals, 79
 portCtxSw_
 CPU port interface, 77
 portCtxSwlsr_
 CPU port interface, 77
 portFindLastSet_
 CPU port interface, 76
 CPU port internals, 79
 portInit_
 CPU port interface, 77
 portInitEarly_
 CPU port interface, 77
 portInitLate_
 CPU port interface, 77
 portIntDisable_
 CPU port interface, 75
 portIntGet_
 CPU port interface, 75
 portIntSet_
 CPU port interface, 76
 portReg_T
 CPU port interface, 75
 portStck, 103
 reg, 103
 portStck_T
 CPU port interface, 75
 portSysTmrDisable_
 CPU port interface, 76
 portSysTmrEnable_
 CPU port interface, 76
 portSysTmrInit_
 CPU port interface, 76
 portSysTmrIsrDisable_
 CPU port interface, 77
 portSysTmrIsrEnable_
 CPU port interface, 77
 portSysTmrReload_
 CPU port interface, 76
 portSysTmrTerm_
 CPU port interface, 76
 portThdStart_
 CPU port interface, 77
 prev
 esThd::thdL, 107
 esVTmr::tmrL, 110
 prio
 esThd, 97
 prioBM_T
 Internals, 42
 prioBMClear
 Internals, 43
 prioBMGet
 Internals, 43
 prioBMInit
 Internals, 43
 prioBMIsEmpty
 Internals, 43
 prioBMSet
 Internals, 43
 prioOcc
 esThdQ, 100
 pthd
 esKernCtrl, 95
 ptick
 sysTmr, 105
 pwr2LKP
 CPU port interface, 78
 CPU port internals, 80

 q
 esThd::thdL, 107
 esVTmr::tmrL, 110
 qCnt
 esThd, 97
 qRld
 esThd, 97

 r0
 portCtx, 103
 reg
 portStck, 103
 rtick
 esVTmr, 102

 schedInit
 Internals, 43
 schedQml
 Internals, 44
 schedQmNextl
 Internals, 44
 schedRdyAddInitl
 Internals, 44
 schedSleep
 Internals, 44
 schedStart
 Internals, 44
 schedWakeUpI
 Internals, 44
 signature
 esThd, 97

- esThdQ, [100](#)
- esVTmr, [102](#)
- state
 - esKernCtrl, [95](#)
- stck
 - esThd, [97](#)
- sysTmr, [104](#)
 - ptick, [105](#)
 - vTmrArmed, [105](#)
 - vTmrPend, [105](#)
- sysTmr_T
 - Internals, [42](#)
- sysTmrActivate
 - Internals, [44](#)
- sysTmrDeactivateI
 - Internals, [44](#)
- sysTmrInit
 - Internals, [44](#)
- TRUE
 - Compiler port, [67](#)
- thdL
 - esThd, [97](#)
- thdLSentinel_T
 - Internals, [42](#)
- tmrL
 - esVTmr, [102](#)
- userAssert
 - Interface, [90](#)
- userPostIdle
 - Interface, [35](#)
- userPostKernInit
 - Interface, [34](#)
- userPostThdInit
 - Interface, [35](#)
- userPreCtxSw
 - Interface, [35](#)
- userPreIdle
 - Interface, [35](#)
- userPreKernInit
 - Interface, [34](#)
- userPreKernStart
 - Interface, [34](#)
- userPreSysTmr
 - Interface, [34](#)
- userPreThdTerm
 - Interface, [35](#)
- vTmrAddArmedS
 - Internals, [45](#)
- vTmrArmed
 - sysTmr, [105](#)
- vTmrEvaluateI
 - Internals, [45](#)
- vTmrImportPend
 - Internals, [45](#)
- vTmrImportPendI
 - Internals, [45](#)
- vTmrPend
 - sysTmr, [105](#)
- vTmrSleep
 - Internals, [45](#)