

eSolid - Real-Time Kernel
1.0BetaR01

Generated by Doxygen 1.8.3.1

Sat Jun 8 2013 15:48:08

Contents

1	eSolid Real-Time Kernel	1
1.1	eSolid RT Kernel Specification	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	Error checking	2
1.1.11	Profiling	3
2	Directory and file organization	3
2.1	Intro	3
2.1.1	What is a port?	3
2.2	Code Sections	3
2.2.1	Port independent code	3
2.2.2	Port dependent code	4
2.2.3	Template and example code	4
3	Kernel states	4
3.1	Intro	4
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	6
5	Critical sections	7
5.1	Intro	7
5.1.1	eSolid RT Kernel internal critical sections	7
5.2	Implementation	8
5.2.1	Disabling interrupts	8
5.2.2	Disabling Kernel	8
5.2.3	Using semaphores	8
6	Time complexity	8

6.1	Intro	9
6.1.1	Big O notation	9
6.2	Constant time	9
6.2.1	eSolid RT Kernel time complexity	9
7	Scheduler	9
7.1	Quantum	9
7.2	Threads List	10
7.3	Threads Queue	10
7.3.1	Simple Ready Threads Queue	10
7.3.2	Complex Ready Threads Queue	11
7.4	Ready Threads Queue	11
8	Error checking	11
8.1	Intro	11
9	Module Documentation	11
9.1	Kernel	11
9.1.1	Detailed Description	12
9.2	Interface	13
9.2.1	Detailed Description	16
9.2.2	Macro Definition Documentation	16
9.2.3	Typedef Documentation	18
9.2.4	Enumeration Type Documentation	18
9.2.5	Function Documentation	18
9.2.6	Variable Documentation	29
9.3	Internals	30
9.3.1	Detailed Description	32
9.3.2	Macro Definition Documentation	32
9.3.3	Typedef Documentation	34
9.3.4	Enumeration Type Documentation	34
9.3.5	Function Documentation	34
9.3.6	Variable Documentation	36
9.4	Configuration	38
9.4.1	Detailed Description	38
9.4.2	Macro Definition Documentation	39
9.5	Port template	41
9.5.1	Detailed Description	41
9.6	Compiler port	42
9.6.1	Detailed Description	43
9.6.2	Macro Definition Documentation	43

9.6.3	Typedef Documentation	44
9.6.4	Enumeration Type Documentation	45
9.7	CPU port interface	46
9.7.1	Detailed Description	49
9.7.2	Macro Definition Documentation	50
9.7.3	Typedef Documentation	53
9.7.4	Function Documentation	54
9.7.5	Variable Documentation	56
9.8	CPU port internals	57
9.8.1	Detailed Description	57
9.8.2	Function Documentation	57
9.8.3	Variable Documentation	58
9.9	CPU port configuration	59
9.9.1	Detailed Description	59
9.9.2	Macro Definition Documentation	59
9.10	Kernel configuration	60
9.10.1	Detailed Description	60
9.10.2	Macro Definition Documentation	61
10	Data Structure Documentation	63
10.1	esKernCtrl Struct Reference	63
10.1.1	Detailed Description	64
10.1.2	Field Documentation	64
10.2	esThd Struct Reference	64
10.2.1	Detailed Description	66
10.2.2	Field Documentation	66
10.3	esThdQ Struct Reference	67
10.3.1	Detailed Description	68
10.3.2	Field Documentation	68
10.4	esTmr Struct Reference	68
10.4.1	Detailed Description	69
10.4.2	Field Documentation	69
10.5	portCtx Struct Reference	70
10.5.1	Detailed Description	70
10.5.2	Field Documentation	70
10.6	portStck Struct Reference	71
10.6.1	Detailed Description	71
10.6.2	Field Documentation	71
10.7	esThdQ::prioBM Struct Reference	71
10.7.1	Detailed Description	72

10.7.2	Field Documentation	72
10.8	sysTmr Struct Reference	72
10.8.1	Detailed Description	73
10.8.2	Field Documentation	73
10.9	esThd::thdL Struct Reference	73
10.9.1	Detailed Description	74
10.9.2	Field Documentation	74
10.10	esTmr::tmrL Struct Reference	75
10.10.1	Detailed Description	76
10.10.2	Field Documentation	76
11	File Documentation	76
11.1	compiler.h File Reference	76
11.1.1	Detailed Description	78
11.2	cpu.c File Reference	79
11.2.1	Detailed Description	79
11.3	cpu.h File Reference	79
11.3.1	Detailed Description	83
11.3.2	Variable Documentation	83
11.4	cpu_cfg.h File Reference	84
11.4.1	Detailed Description	84
11.5	kernel-example.dox File Reference	84
11.6	kernel.c File Reference	85
11.6.1	Detailed Description	89
11.6.2	Variable Documentation	89
11.7	kernel.dox File Reference	90
11.8	kernel.h File Reference	90
11.8.1	Detailed Description	94
11.9	kernel_cfg.h File Reference	95
11.9.1	Detailed Description	96
11.10	kernel_cfg.h File Reference	96
11.10.1	Detailed Description	96
Index		96

1 eSolid Real-Time Kernel

1.1 eSolid RT Kernel Specification

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

All 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. Their execution time 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 types 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 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.

1.1.11 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`

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`

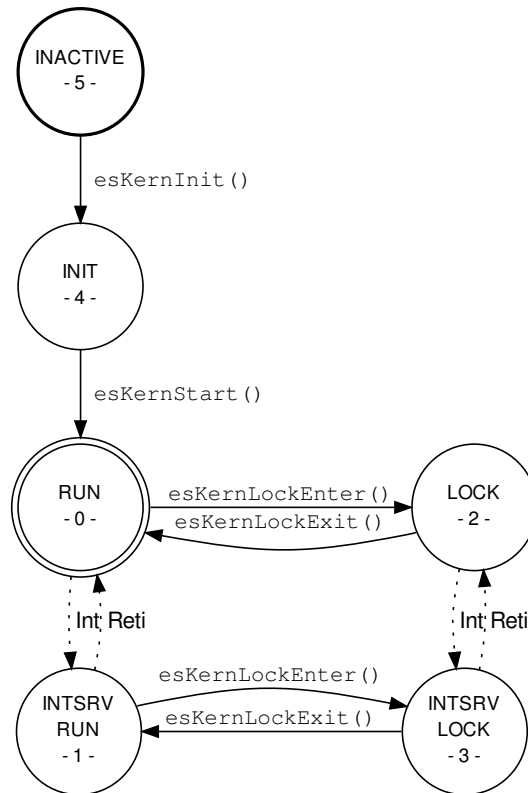
3 Kernel states

Details about kernel states

3.1 Intro

A Kernel state machine is a behavior model of the kernel core. Each state defines which functions can be called and what methods are allowed.

3.2 eSolid RT Kernel 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 [esThdInit\(\)](#) 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 (like 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

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_*` state. Each `INTSRV` state corresponds to the state where the execution was interrupted from and the kernel will return to it's original state.

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

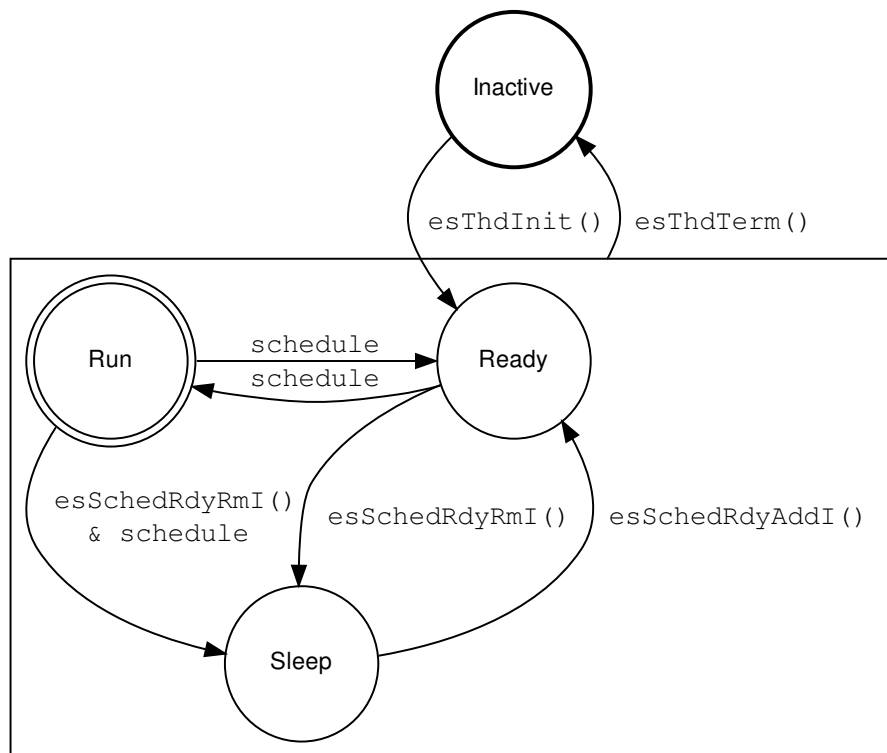
A thread, also called a thread of execution is the smallest sequence of program instructions that can be managed by an operating system scheduler. Multi-threading is usually implemented by time-division multiplexing where the processor switches between threads. Context switching occurs fast enough that 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.

4.1.2 Thread states

A thread can be in one of the following states:



5 Critical sections

How to deal with critical sections in 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 its 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. Then by using the macro `ES_CRITICAL_ENTER()` the state of enabled interrupts will be saved in `auto` variable declared earlier. Immediately after saving of 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 */
```

5.2.2 Disabling Kernel**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

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 each priority level. The number of bits used in this variable depends on [CFG_SCHED_PRIO_LVL](#) value. If a bit at `Nth` position is set then there is a thread inserted in Thread List at `Nth` priority.

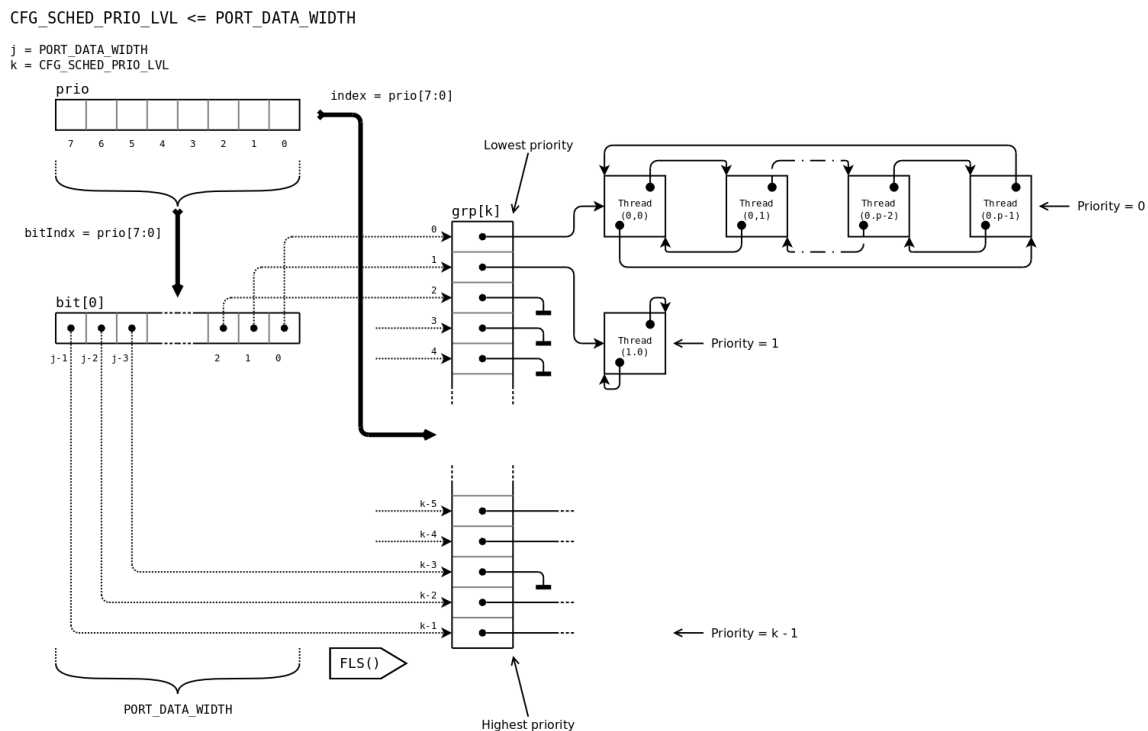


Figure 1: Ready Threads Queue - low number of priority levels

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)
```

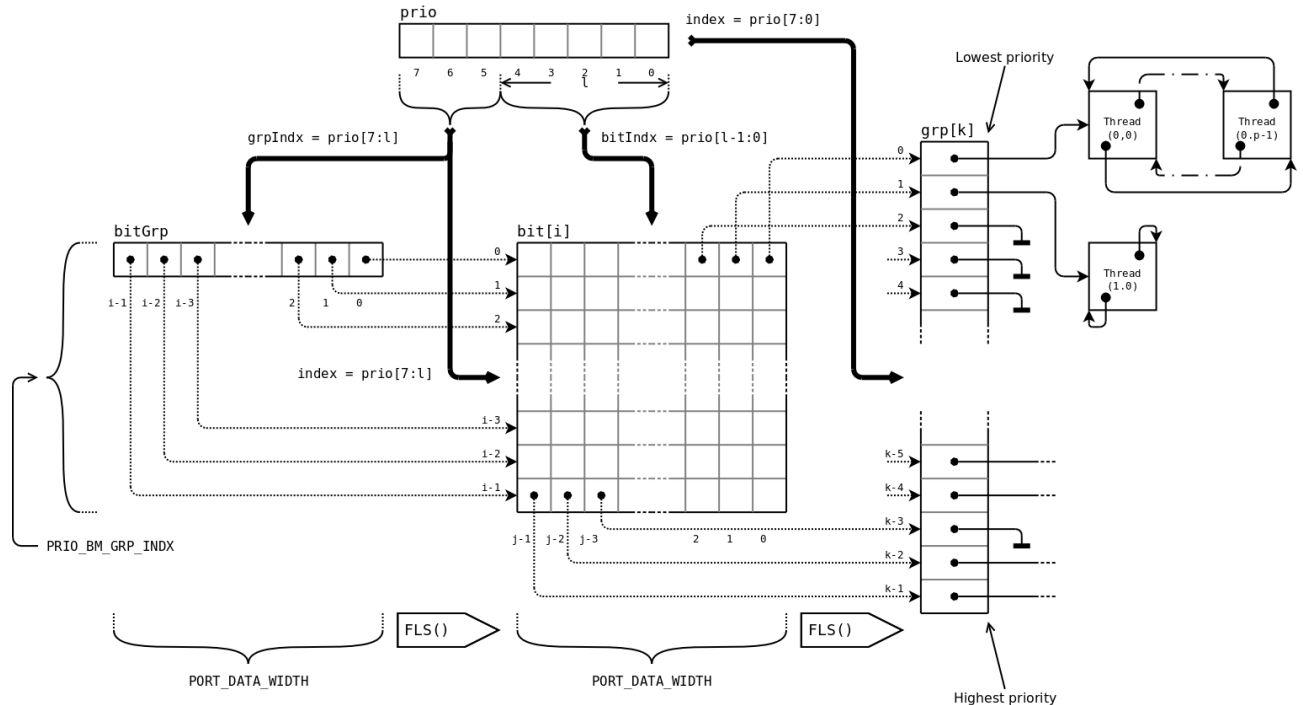


Figure 2: Ready Threads Queue - high number of priority levels

7.4 Ready Threads Queue

Ready Threads Queue holds threads that are ready for execution.

8 Error checking

How errors are detected

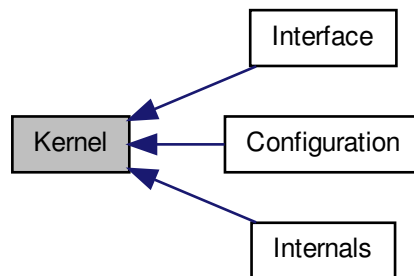
8.1 Intro

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

Kernel identification and version number

- #define [ES_KERNEL_VER](#) 0x10000UL
Identifies the underlying kernel version number.
- #define [ES_KERNEL_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](#).

- void [esKernLockEnterI](#) (void)
To do.
- void [esKernLockExitI](#) (void)
To do.
- #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.

Error checking

Some basic infrastructure for error checking

These macros provide basic detection of errors. For more details see [Error checking](#).

- `#define ES_ASSERT(expr)`
Generic assert macro.
- `#define ES_API_OBLIGATION(expr) expr`
Execute code to fulfill the contract.
- `#define ES_API_REQUIRE(expr) ES_ASSERT(expr)`
Make sure the caller has fulfilled all contract preconditions.
- `#define ES_API_ENSURE(expr) ES_ASSERT(expr)`
Make sure the callee has fulfilled all contract postconditions.

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(*thdf)(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 esThdSetPriol (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.

Timer services

- `typedef struct esTmr esTmr_T`
Timer type.
- `void esTmrAddl (esTmr_T *tmr, esTick_T tick, void(*fn)(void *), void *arg)`
Add a new timer.

Thread Queue management

- typedef struct `esThdQ` `esThdQ_T`
Thread queue type.
- void `esThdQInit` (`esThdQ_T` *thdQ)
Initialize Thread Queue.
- void `esThdQAddl` (`esThdQ_T` *thdQ, `esThd_T` *thd)
Add a thread to the tail of 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 linked list.
- `bool_T` `esThdQIsEmpty` (const `esThdQ_T` *thdQ)
Is thread queue empty.
- #define `PRIO_BM_GRP_IDX` ((`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_INIT` = 0x04U,
 `ES_KERN_INACTIVE` = 0x05U }
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 `esKernSysTmrI` (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 `esSchedRdyAddl (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 `esSchedYieldIsrl (void)`
Force the scheduler invocation which will evaluate all ready threads and switch to ready thread with the highest priority.

System timer

- void `esSysTmrEnable (void)`
Enable system timer tick events.
- void `esSysTmrDisable (void)`
Disable system timer tick events.

Kernel hook functions

- PORT_C_NORETURN void `userAssert (const char *funcName, const char *expr)`
An assertion has failed. This function should inform the user about failed assertion.
- void `userSysTmr (void)`
System timer hook function, called from system system timer ISR function.
- void `userKernInit (void)`
Kernel initialization hook function, called from `esKernInit()` function.
- void `userKernStart (void)`
Kernel start hook function, called from `esKernStart()` function.
- void `userThdInitEnd (void)`
Thread initialization end hook function, called from `esThdInit()` function.
- void `userThdTerm (void)`
Thread terminate hook function, called from `esThdTerm()` or when a thread terminates itself.
- void `userCtxSw (esThd_T *oldThd, esThd_T *newThd)`
Kernel context switch hook function, called from `esSchedYieldl()` and `esSchedYieldIsrl()` functions.

9.2.1 Detailed Description

Application programming interface.

9.2.2 Macro Definition Documentation

9.2.2.1 #define ES_KERNEL_VER 0x10000UL

Identifies the underlying kernel version number.

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

9.2.2.2 #define ES_KERNEL_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_ASSERT(*expr*)

Value:

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

Generic assert macro.

Parameters

<i>expr</i>	Expression which MUST be TRUE
-------------	-------------------------------

9.2.2.7 #define ES_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.
-------------	---

9.2.2.8 #define ES_API_REQUIRE(*expr*) ES_ASSERT(*expr*)

Make sure the caller has fulfilled all contract preconditions.

Parameters

<i>expr</i>	Expression which MUST be satisfied
-------------	------------------------------------

9.2.2.9 #define ES_API_ENSURE(*expr*) ES_ASSERT(*expr*)

Make sure the callee has fulfilled all contract postconditions.

Parameters

<i>expr</i>	Expression which MUST be satisfied
-------------	------------------------------------

9.2.2.10 #define ES_STCK_SIZE(*elem*) PORT_STCK_SIZE(*elem*)

Converts the required stack elements into the stack array index.

9.2.2.11 #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 struct esTmr esTmr_T

Timer type.

9.2.3.4 typedef struct esThdQ esThdQ_T

Thread queue type.

9.2.3.5 typedef enum esKernState esKernState_T

Kernel state type.

9.2.3.6 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_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.

This is the function which **MUST** be called first before any other kernel API. It initializes internal data structures which all other kernel API use.

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.

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.

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).

Precondition

- 1) The `kernel state == ES_KERN_INIT`, see [Kernel states](#).
- 2) At least one thread must be initialized and be in Threads Ready Queue before starting multi-threading, see [esThdInit\(\)](#).

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.

Object class:

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

9.2.5.3 void esKernSysTmrI (void)

Process the system timer event.

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

Precondition

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

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.

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.

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 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.

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.

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 [esKernLockEnterI\(\)](#))

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 esKernLockEnterI (void)

To do.

9.2.5.7 void esKernLockExitI (void)

To do.

9.2.5.8 void esThdInit (esThd_T * thd, void(*) (void *) thdf, 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_T . The structure will be used as information container for the thread. It is assumed that storage for the esThd_T structure is allocated by the user code.
<i>thdf</i>	Thread 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_T .
<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.
<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.

Threads must be created in order for OS to recognize them as threads. Initialize a thread by calling [esThdInit\(\)](#) and provide arguments specifying to OS 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.

Precondition

- 1) The kernel state `ES_KERN_INACTIVE`, see [Kernel states](#).
- 2) `thd != NULL`
- 3) `thdf != NULL`
- 4) `stckSize >= PORT_STCK_MINSIZE_VAL`, see [PORT_STCK_MINSIZE_VAL](#).
- 5) `0 <= prio <= CFG_SCHED_PRIO_LVL`, see [CFG_SCHED_PRIO_LVL](#).

Postcondition

- 1) `thd->signature == THD_CONTRACT_SIGNATURE`, each `esThd_T` structure will have valid signature after initialization.

Object class:

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

9.2.5.9 void esThdTerm (esThd_T * thd)

Terminate the specified thread.

Parameters

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

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 [esThd_T](#) 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 on a synchronization mechanism).

9.2.5.10 `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.

9.2.5.11 `static PORT_C_INLINE uint8_t esThdGetPrio (esThd_T * thd) [static]`

Get the priority of a thread.

Parameters

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

Returns

The priority of the thread pointed by `thd`.

Note

This is `inline` function.

9.2.5.12 `void esThdSetPrio (esThd_T * thd, uint8_t prio)`

Set the priority of a thread.

Parameters

<i>thd</i>	Thread: is a pointer to the thread structure, esThd_T .
<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 [esThd_T](#) structure.
- 4) `0 <= prio <= CFG_SCHED_PRIO_LVL`, 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.

9.2.5.13 `void esThdPostI (esThd_T * thd)`

Post to thread semaphore.

Parameters

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

Function class:

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

9.2.5.14 void esThdPost (esThd_T * thd)

Post to thread semaphore.

Parameters

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

Object class:

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

9.2.5.15 void esThdWaitl (void)

Wait for thread semaphore.

Function class:

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

9.2.5.16 void esThdWait (void)

Wait for thread semaphore.

Object class:

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

9.2.5.17 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`

Postcondition

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

Object class:

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

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

Add a thread to the tail of 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_T .

This function adds a thread at the tail of the specified Thread Queue.

Precondition

- 1) *thdQ* != NULL
- 2) *thdQ*->signature == THDQ_CONTRACT_SIGNATURE, the pointer must point to a [esThdQ_T](#) structure.
- 3) *thd* != NULL
- 4) *thd*->signature == THD_CONTRACT_SIGNATURE, the pointer must point to a [esThd_T](#) structure.
- 5) *thd*->thdL.q == NULL, thread MUST NOT be in any queue.
- 6) 0 <= *thd*->prio <= CFG_SCHED_PRIO_LVL, 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.

9.2.5.19 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_T .

Precondition

- 1) *thd* != NULL
- 2) *thd*->signature == THD_CONTRACT_SIGNATURE, the pointer must point to a [esThd_T](#) structure.
- 3) *thdQ* != NULL
- 4) *thdQ*->signature == THDQ_CONTRACT_SIGNATURE, the pointer must point to a [esThdQ_T](#) structure.
- 5) *thd*->thdL.q == *thdQ*, thread MUST BE in the queue.
- 6) 0 <= *thd*->prio <= CFG_SCHED_PRIO_LVL, 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.

9.2.5.20 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 [esThdQ_T](#) structure.

Function class:

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

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

Fetch the next thread and rotate 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 [esThdQ_T](#) structure.
- 3) `0 <= prio <= CFG_SCHED_PRIO_LVL`, see [CFG_SCHED_PRIO_LVL](#).

9.2.5.22 `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 [esThdQ_T](#) structure.

9.2.5.23 `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_T .
------------	---

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 [esThd_T](#) structure.
- 3) *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.

9.2.5.24 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_T .
------------	---

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 [esThd_T](#) 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.

9.2.5.25 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](#).

Warning

Scheduler will have undefined behavior if there is no ready thread to run (e.g. empty [gRdyQueue](#)) at the time it is invoked.

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 esSchedYieldIsrl (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](#).

Warning

Scheduler will have undefined behavior if there is no ready thread to run (e.g. empty [gRdyQueue](#)) at the time it is invoked.

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 esSysTmrEnable (void)

Enable system timer tick events.

This function will override scheduler power savings algorithm and force the system timer into running (active) state.

9.2.5.28 void esSysTmrDisable (void)

Disable system timer tick events.

This function will try to switch off the system timer. If the system timer is used by scheduler than the scheduler will take control of the system timer.

9.2.5.29 void esTmrAddl (esTmr_T * tmr, esTick_T tick, void(*)(void *) fn, void * arg)

Add a new timer.

Parameters

<i>tmr</i>	Pointer to the timer ID structure, esTmr .
<i>tick</i>	The timer delay expressed in system ticks
<i>fn</i>	Pointer to the callback function
<i>arg</i>	Pointer to the arguments of callback function

9.2.5.30 PORT_C.NORETURN void userAssert (const char * funcName, const char * expr)

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

Parameters

<i>funcName</i>	Function name: pointer to the function name string where the assertion failed. Macro will automatically fill in the function name for the function.
<i>expr</i>	Expression: pointer to the string containing the expression that failed.

Function should just print the information which was given by the macros. After the function informs the user it MUST go into infinite loop or HALT the processor.

Precondition

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

Note

- 1) The definition of this function must be written by the user.
- 2) This function is called only if [CFG_API_VALIDATION](#) is active.

9.2.5.31 void userSysTmr (void)

System timer hook function, called from system system timer ISR function.

This function is called whenever a system event is generated.

Note

- 1) The definition of this function must be written by the user.
- 2) This function is called only if [CFG_HOOK_SYSTMTR_EVENT](#) is active.

9.2.5.32 void userKernInit (void)

Kernel initialization hook function, called from [esKernInit\(\)](#) function.

This function is called before kernel initialization.

Note

- 1) The definition of this function must be written by the user.
- 2) This function is called only if [CFG_HOOK_KERN_INIT](#) is active.

9.2.5.33 void userKernStart (void)

Kernel start hook function, called from [esKernStart\(\)](#) function.

This function is called before kernel start.

Note

- 1) The definition of this function must be written by the user.
- 2) This function is called only if [CFG_HOOK_KERN_START](#) is active.

9.2.5.34 void userThdInitEnd (void)

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

This function is called after the thread initialization.

Note

- 1) The definition of this function must be written by the user.
- 2) This function is called only if [CFG_HOOK_THD_INIT_END](#) is active.

9.2.5.35 void userThdTerm (void)

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

Note

- 1) The definition of this function must be written by the user.
- 2) This function is called only if [CFG_HOOK_THD_TERM](#) is active.

9.2.5.36 void userCtxSw (esThd_T * oldThd, esThd_T * newThd)

Kernel context switch hook function, called from [esSchedYieldI\(\)](#) and [esSchedYieldIsrI\(\)](#) functions.

This function is called at each 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) The definition of this function must be written by the user.
- 2) This function is called only if [CFG_HOOK_CTX_SW](#) is active.

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 `PRIQ_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 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 `SYSTMTR_SCHED_QM_MSK` (1U << 0)
Scheduler is using system timer Quantum mask.
- #define `SYSTMTR_USR_QM_MSK` (1U << 1)
User is using system timer Quantum mask.
- #define `SCHED_POWER_SAVE` 0U
Enable/disable scheduler power savings mode.
- #define `THD_CONTRACT_SIGNATURE` ((portReg_T)0xFEEDBEEFU)
Thread structure signature.
- #define `THDQ_CONTRACT_SIGNATURE` ((portReg_T)0xFEEDBEEEU)
Thread Queue 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_PREV`(list, entry) (entry)->list.prev
Get the previous entry.
- #define `DLIST_ENTRY_NEXT`(list, entry) (entry)->list.next
Get the next entry.
- #define `DLIST_ENTRY_INIT`(list, entry)
Initialize entry.
- #define `DLIST_ENTRY_ADD_AFTER`(list, current, entry)

- *Add new entry after current entry.*
- `#define DLIST_ENTRY_RM(list, entry)`
Remove the entry from a list.
- `#define KTMR_STCK_SIZE PORT_STCK_SIZE(40U)`
System Timer kernel thread stack size.
- `#define KIDLE_STCK_SIZE PORT_STCK_SIZE(40U)`
Idle kernel thread stack size.

Variables

- static `esThdQ_T gRdyQueue`
Ready Thread queue.

System timer

- enum `sysTmrState` {
 `SYSTMR_ACTIVE`,
 `SYSTMR_INACTIVE` }
- *System timer state enumeration.*
- typedef struct `sysTmr sysTmr_T`
System Timer type.
- static void `sysTmrInit` (void)
Initialize system timer hardware.
- static void `sysTmrTryDeactivate` (void)
Try to deactivate system timer.
- static void `sysTmrTryActivate` (void)
Try to activate 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 `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 scheduler data structures ready for multi-threading.
- static void `schedRdyAddInitl` (`esThd_T *thd`)

Initialize scheduler ready structure during the thread add operation.

- static void `schedQm1` (void)
Do the Quantum (Round-Robin) scheduling.
- static `PORT_C_INLINE` void `schedQmActivate` (void)
Activate system timer Quantum mode.
- static `PORT_C_INLINE` void `schedQmDeactivate` (void)
Deactivate system timer Quantum mode.
- static void `schedQmEvaluatel` (`esThd_T` *thd)
Evaluate if the system timer Quantum mode is needed.

System timer kernel thread

- static void `tmrListAddSort` (`esTmr_T` *list, `esTmr_T` *tmr)
- static void `kTmrInit` (void)
Initialization of System Timer Thread.
- static void `kTmr` (void *arg)
System timer thread code.

Idle kernel thread

- 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: $\log_2(\text{PORT_DATA_WIDTH_VAL})$

9.3.2.2 #define SCHED_STATE_INTSRV_MSK (1U << 0)

Kernel state variable bit position which defines if 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 SYSTMTR_SCHED_QM_MSK (1U << 0)

Scheduler is using system timer Quantum mask.

9.3.2.5 #define SYSTMTR_USR_QM_MSK (1U << 1)

User is using system timer Quantum mask.

9.3.2.6 #define SCHED_POWER_SAVE 0U

Enable/disable scheduler power savings mode.

9.3.2.7 #define THD_CONTRACT_SIGNATURE ((portReg_T)0xFEEDBEEFU)

Thread structure signature.

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

9.3.2.8 #define THDQ_CONTRACT_SIGNATURE ((portReg_T)0xFEEDBEEEU)

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.

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

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

9.3.2.10 #define DLIST_IS_ENTRY_LAST(list, entry) DLIST_IS_ENTRY_FIRST(list, entry)

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

9.3.2.11 #define DLIST_IS_ENTRY_SINGLE(list, entry) DLIST_IS_ENTRY_FIRST(list, entry)

DList macro: is the thread single in the list.

9.3.2.12 #define DLIST_ENTRY_PREV(list, entry) (entry)->list.prev

Get the previous entry.

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

Get the next entry.

9.3.2.14 #define DLIST_ENTRY_INIT(list, entry)

Value:

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

Initialize entry.

9.3.2.15 #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)
```

Add new entry after current entry.

9.3.2.16 `#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)
```

Remove the `entry` from a list.

9.3.2.17 `#define KTMR_STCK_SIZE PORT_STCK_SIZE(40U)`

System Timer kernel thread stack size.

9.3.2.18 `#define KIDLE_STCK_SIZE PORT_STCK_SIZE(40U)`

Idle kernel thread stack size.

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.

9.3.4 Enumeration Type Documentation

9.3.4.1 `enum sysTmrState`

System timer state enumeration.

Enumerator

`SYSTMR_ACTIVE` System timer is running. `SYSTMR_ACTIVE`

`SYSTMR_INACTIVE` System timer is stopped. `SYSTMR_INACTIVE`

9.3.5 Function Documentation

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

Initialize bitmap.

Parameters

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

9.3.5.2 `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

9.3.5.3 `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

9.3.5.4 `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

9.3.5.5 `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

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

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

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

Set scheduler data structures ready for multi-threading.

This function is called just before multi-threading has commenced.

9.3.5.8 `static void schedRdyAddInitl (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.

9.3.5.9 `static void schedQml (void) [static]`

Do the Quantum (Round-Robin) scheduling.

9.3.5.10 `static PORT_C_INLINE void schedQmActivate (void) [static]`

Activate system timer Quantum mode.

9.3.5.11 `static PORT_C_INLINE void schedQmDeactivate (void) [static]`

Deactivate system timer Quantum mode.

9.3.5.12 `static void schedQmEvaluatel (esThd_T * thd) [static]`

Evaluate if the system timer Quantum mode is needed.

Parameters

<i>thd</i>	Pointer to the thread which is ready to be executed
------------	---

9.3.5.13 `static void sysTmrlnit (void) [static]`

Initialize system timer hardware.

9.3.5.14 `static void sysTmrTryDeactivate (void) [static]`

Try to deactivate system timer.

9.3.5.15 `static void sysTmrTryActivate (void) [static]`

Try to activate system timer.

9.3.5.16 `static void kTmrlnit (void) [static]`

Initialization of System Timer Thread.

9.3.5.17 `static void kTmr (void * arg) [static]`

System timer thread code.

Parameters

<i>arg</i>	NO ARGUMENTS - thread does not use argument
------------	---

9.3.5.18 `static void kIdlelnit (void) [static]`

Initialization of Idle thread.

9.3.5.19 `static void kIdle (void * arg) [static]`

Idle thread code.

Parameters

<i>arg</i>	NO ARGUMENTS - thread does not use argument
------------	---

9.3.6 Variable Documentation

9.3.6.1 `esThdQ_T gRdyQueue` `[static]`

Ready Thread queue.

9.4 Configuration

Configuration settings.

Collaboration diagram for Configuration:



Kernel configuration options and settings

- #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_SYSTMTR_MODE` 1U
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 end hook function.
- #define `CFG_HOOK_THD_TERM` 0U
Thread termination hook function.
- #define `CFG_HOOK_CTX_SW` 0U
Context switch hook function.

9.4.1 Detailed Description

Configuration settings.

9.4.2 Macro Definition Documentation

9.4.2.1 `#define CFG_API_VALIDATION 1U`

Enable/disable API arguments validation.

9.4.2.2 `#define CFG_SCHED_PRIO_LVL 8U`

Scheduler priority levels.

9.4.2.3 `#define CFG_SCHED_TIME_QUANTUM 10U`

Scheduler Round-Robin time quantum.

9.4.2.4 `#define CFG_SYSTMTR_MODE 1U`

System timer mode.

Possible values are:

- 0U - fixed mode
- 1U - inhibited mode
- 2U - adaptive mode

9.4.2.5 `#define CFG_SYSTMTR_EVENT_FREQUENCY 100UL`

The frequency of system 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 counter.

Possible values are:

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

9.4.2.7 `#define CFG_HOOK_SYSTMTR_EVENT 0U`

System timer event hook function.

9.4.2.8 `#define CFG_HOOK_KERN_INIT 0U`

Kernel initialization hook function.

9.4.2.9 `#define CFG_HOOK_KERN_START 0U`

Kernel start hook function.

9.4.2.10 `#define CFG_HOOK_THD_INIT_END 0U`

Thread initialization end hook function.

9.4.2.11 `#define CFG_HOOK_THD_TERM 0U`

Thread termination hook function.

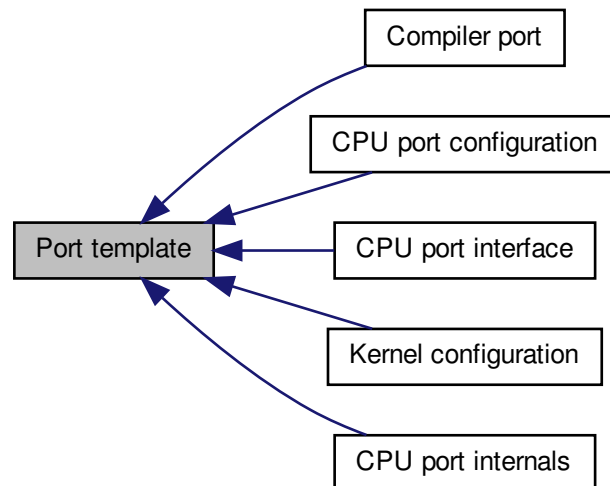
9.4.2.12 `#define CFG_HOOK_CTX_SW 0U`

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.
- 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.

Functions

- struct `portStck` `__attribute__((aligned(1)))`
Alignment of stack structure.

Variables

- struct `portCtx` `__attribute__((aligned(1)))`
- `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_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.

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 stack 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 struct portStck __attribute__((aligned(1)))

Alignment of stack structure.

9.7.4.2 void portIntDisable_(void)

Disable interrupts.

9.7.4.3 portReg_T portIntGet_(void)

Get the current status of enabled/disabled interrupts.

Returns

Interrupt status

9.7.4.4 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.
---------------	--

9.7.4.5 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

9.7.4.6 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.

9.7.4.7 void portSysTmrTerm_(void)

Stop the sistem timer.

9.7.4.8 void portSysTmrReload_(esTick.T ticks)

Reload the system timer.

Parameters

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

9.7.4.9 void portSysTmrEnable_(void)

Enable the system timer.

9.7.4.10 void portSysTmrDisable_(void)

Disable the system timer.

9.7.4.11 void portSysTmrIsrEnable_(void)

Disable the system timer interrupt.

9.7.4.12 void portSysTmrIsrDisable_(void)

Enable the system timer interrupt.

9.7.4.13 void portThdStart_(void)

Start the first thread.

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

Initialize the thread context.

Parameters

in, out	<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.
in	<i>thdf</i>	Pointer to the thread function.
in	<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.

9.7.4.15 void portCtxSw_(void)

Do the context switch - invoked from API.

9.7.4.16 void portCtxSwIsr_(void)

Do the context switch - invoked from ISR.

9.7.4.17 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.

9.7.4.18 void portInit_(void)

Port initialization.

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

9.7.4.19 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: $(1U \ll n)$, 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]`

9.8 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.

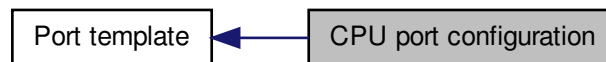
This look up table can be used to accelerate the Logical Shift Left operations which are needed to set bits inside the priority bit map. In plain C this operation would be written as: $(1U \ll n)$, but in many 8-bit CPUs this operation can be lengthy. If there is a need for faster operation than this 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_SYSTMTR_EVENT_FREQUENCY` 100UL
The frequency of system tick event.

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 OFF
- 1U - API validation is ON

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.

Possible values:

- Min: 2U (two priority levels)
- Max: 256U

Warning

Scheduler will have undefined behavior if there is no ready thread to run (e.g. empty [Ready Threads Queue](#)) at the time it is invoked.

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_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](#)

9.10.2.5 `#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.

9.10.2.6 #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.

9.10.2.7 #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.

9.10.2.8 #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.

9.10.2.9 #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.

9.10.2.10 #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.

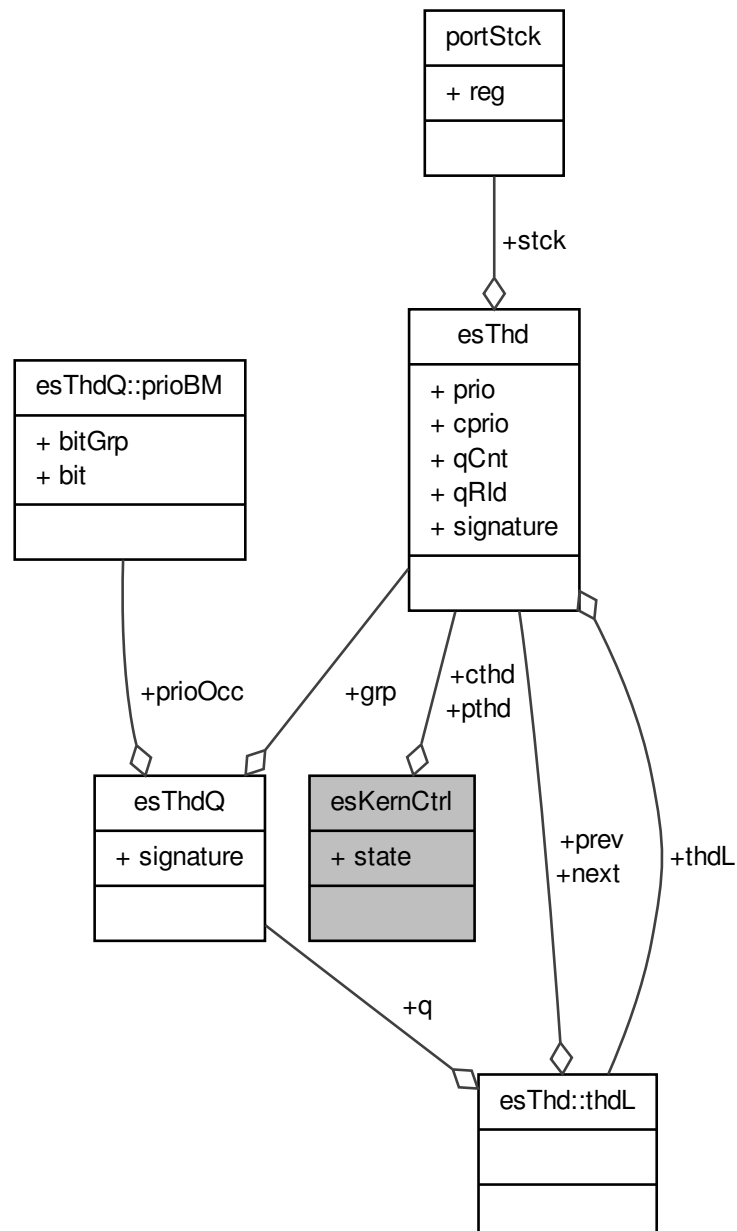
10 Data Structure Documentation

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:

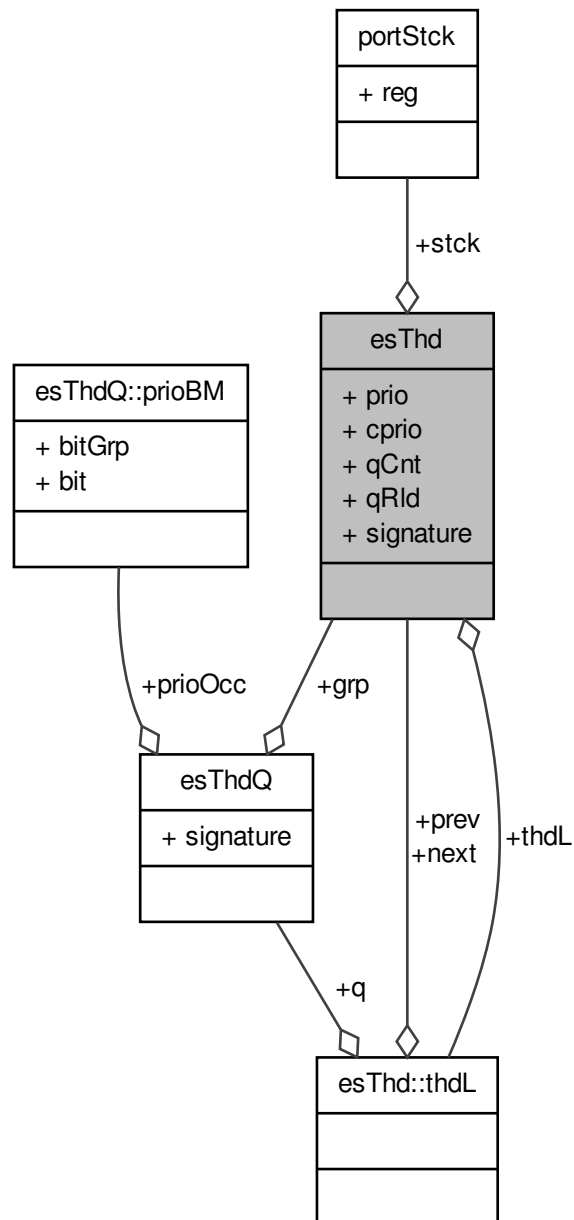
- [kernel.h](#)

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:

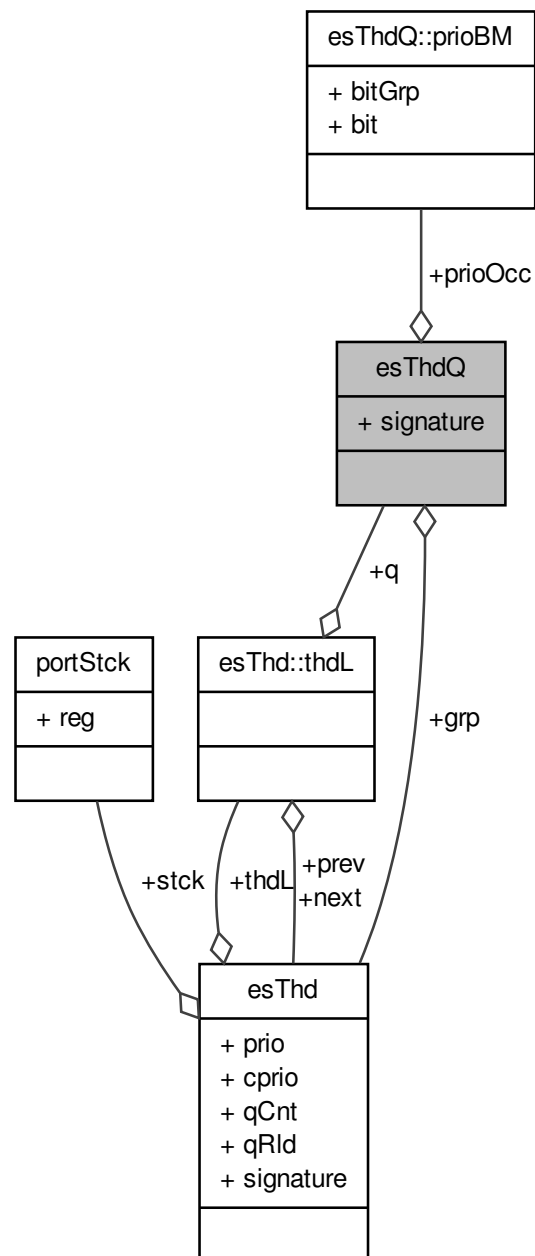
- [kernel.h](#)

10.3 esThdQ Struct Reference

Thread Queue structure.

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

Collaboration diagram for esThdQ:



Data Structures

- struct [prioBM](#)
Priority Bit Map structure.

Data Fields

- struct [esThdQ::prioBM](#) [prioOcc](#)
Priority Occupancy.
- struct [esThd * grp](#) [[CFG_SCHED_PRIO_LVL](#)]
Array of Group Head pointers to priority groups.
- [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 [esThd*](#) [esThdQ::grp](#)[[CFG_SCHED_PRIO_LVL](#)]

Array of Group Head pointers to priority groups.

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:

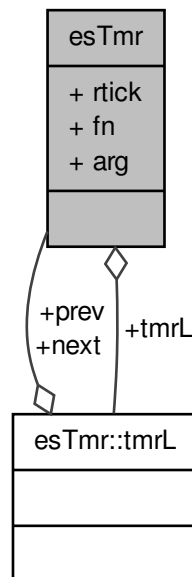
- [kernel.h](#)

10.4 esTmr Struct Reference

Timer structure.

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

Collaboration diagram for esTmr:



Data Structures

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

Data Fields

- struct [esTmr::tmrL](#) [tmrL](#)
Timer linked List.
- esTick_T [rtick](#)
Relative tick value.
- void(* [fn](#))(void *)
Callback function pointer.
- void * [arg](#)
Callback function argument.

10.4.1 Detailed Description

Timer structure.

10.4.2 Field Documentation

10.4.2.1 struct [esTmr::tmrL](#) [esTmr::tmrL](#)

Timer linked List.

10.4.2.2 esTick_T esTmr::rtick

Relative tick value.

10.4.2.3 void(* esTmr::fn)(void *)

Callback function pointer.

10.4.2.4 void* esTmr::arg

Callback function argument.

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

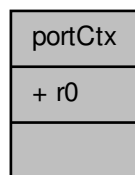
- [kernel.h](#)

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:

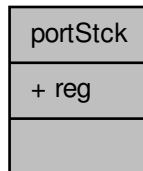
- [cpu.h](#)

10.6 portStck Struct Reference

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

```
#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.

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:

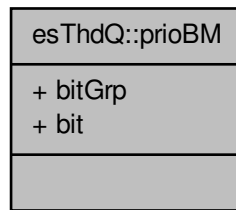
- [cpu.h](#)

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.

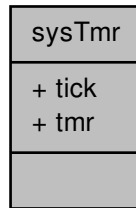
The documentation for this struct was generated from the following file:

- [kernel.h](#)

10.8 sysTmr Struct Reference

Main System Timer structure.

Collaboration diagram for sysTmr:



Data Fields

- `esTick_T tick`
Current system tick counter.
- `uint_fast16_t tmr`
The number of timers in system.

10.8.1 Detailed Description

Main System Timer structure.

10.8.2 Field Documentation

10.8.2.1 `esTick_T sysTmr::tick`

Current system tick counter.

10.8.2.2 `uint_fast16_t sysTmr::tmr`

The number of timers in system.

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

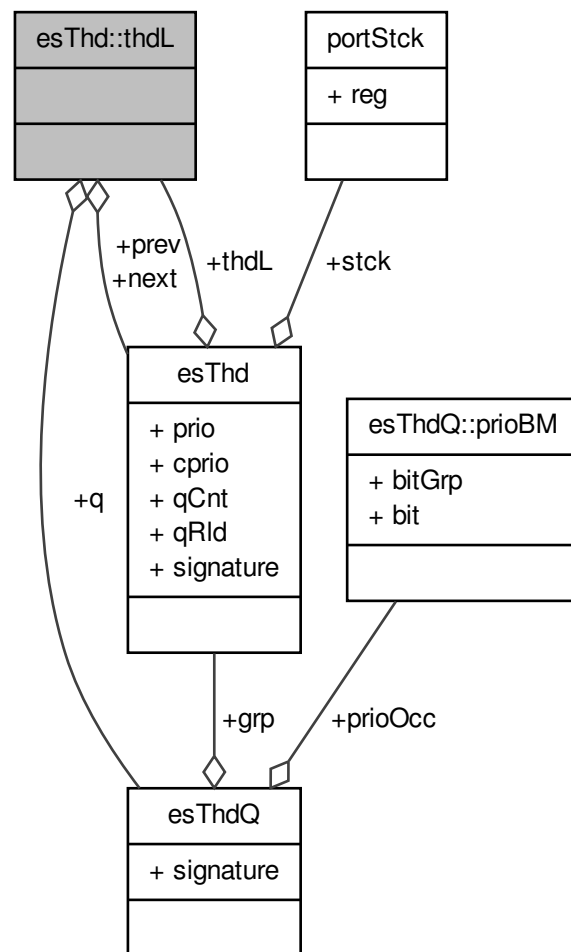
- [kernel.c](#)

10.9 esThd::thdL Struct Reference

Thread linked List structure.

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

Collaboration diagram for esThd::thdL:



Data Fields

- struct `esThdQ * q`
Indicates which queue is used.
- 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

Indicates which queue is used.

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:

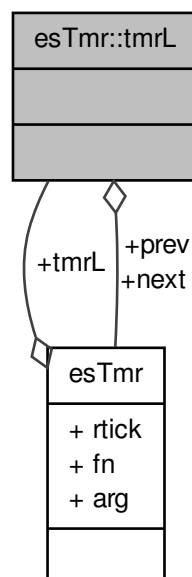
- [kernel.h](#)

10.10 esTmr::tmrL Struct Reference

Timer linked list structure.

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

Collaboration diagram for esTmr::tmrL:



Data Fields

- struct [esTmr](#) * [next](#)
Next thread in timer linked list.
- struct [esTmr](#) * [prev](#)
Previous thread in timer linked list.

10.10.1 Detailed Description

Timer linked list structure.

10.10.2 Field Documentation

10.10.2.1 `struct esTmr* esTmr::tmrL::next`

Next thread in timer linked list.

10.10.2.2 `struct esTmr* esTmr::tmrL::prev`

Previous thread in timer linked list.

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

- [kernel.h](#)

11 File Documentation

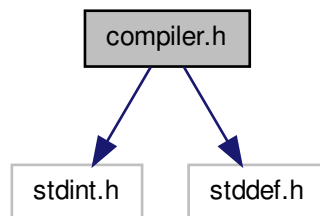
11.1 `compiler.h` File Reference

Interface of Compiler port - Template.

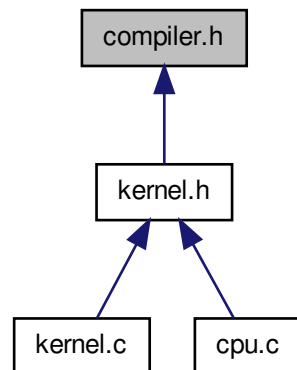
```
#include <stdint.h>
```

```
#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

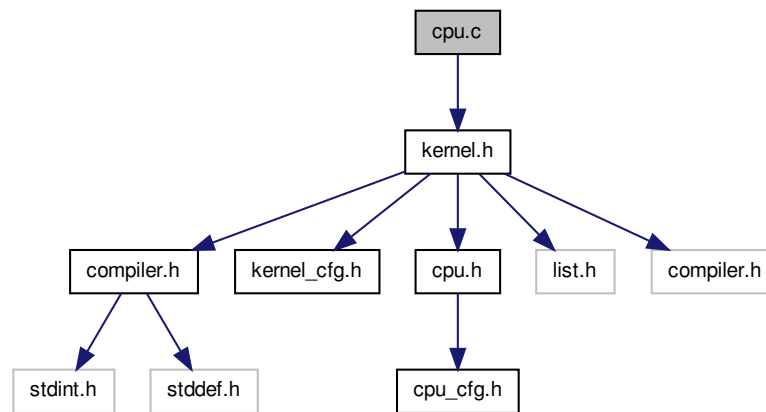
Nenad Radulovic

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

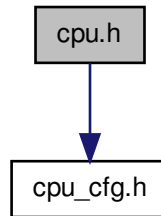
Nenad Radulovic

11.3 **cpu.h File Reference**

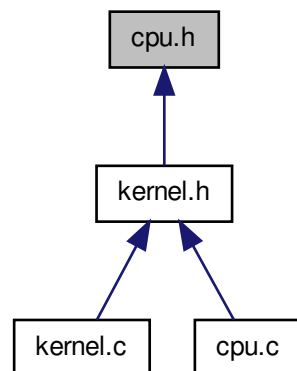
Interface of CPU port - Template.

```
#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.
- 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

- struct `portStck __attribute__((aligned(1)))`
Alignment of stack structure.

General port functions

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

Variables

- `portReg_T reg`
A structure field representing stack data.
- struct `portCtx __attribute__`
- `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

11.3.2 Variable Documentation

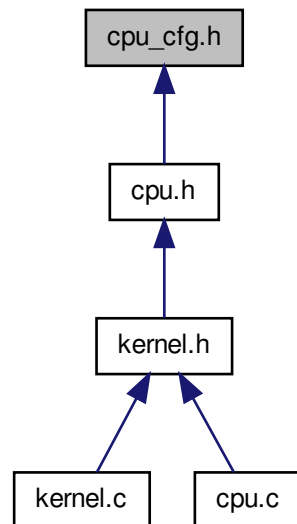
11.3.2.1 portReg_T reg

A structure field representing stack data.

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

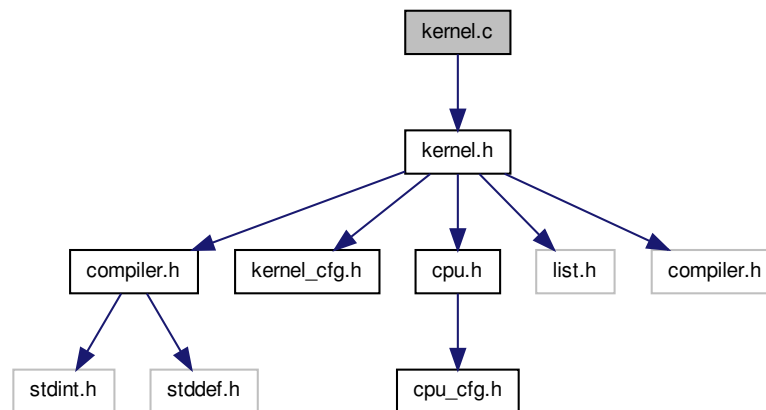
11.5 kernel-example.dox File Reference

11.6 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 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 [SYSTMTR_SCHED_QM_MSK](#) (1U << 0)
Scheduler is using system timer Quantum mask.
- #define [SYSTMTR_USR_QM_MSK](#) (1U << 1)
User is using system timer Quantum mask.
- #define [SCHED_POWER_SAVE](#) 0U
Enable/disable scheduler power savings mode.
- #define [THD_CONTRACT_SIGNATURE](#) ((portReg_T)0xFEEDBEEFU)
Thread structure signature.
- #define [THDQ_CONTRACT_SIGNATURE](#) ((portReg_T)0xFEEDBEEEU)
Thread Queue 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_PREV`(list, entry) (entry)->list.prev
Get the previous entry.
- #define `DLIST_ENTRY_NEXT`(list, entry) (entry)->list.next
Get the next entry.
- #define `DLIST_ENTRY_INIT`(list, entry)
Initialize entry.
- #define `DLIST_ENTRY_ADD_AFTER`(list, current, entry)
Add new entry after current entry.
- #define `DLIST_ENTRY_RM`(list, entry)
Remove the entry from a list.
- #define `KTMR_STCK_SIZE` `PORT_STCK_SIZE`(40U)
System Timer kernel thread stack size.
- #define `KIDLE_STCK_SIZE` `PORT_STCK_SIZE`(40U)
Idle kernel thread stack size.

Functions

- void `esKernInit` (void)
Initialize kernel internal data structures.
- `PORT_C_NORETURN` void `esKernStart` (void)
Start the multi-threading.
- void `esKernSysTmrl` (void)
Process the system timer event.
- void `esKernLockEnterI` (void)
To do.
- void `esKernLockExitI` (void)
To do.
- void `esKernIsrPrologueI` (void)
Enter Interrupt Service Routine.
- void `esKernIsrEpilogueI` (void)
Exit Interrupt Service Routine.
- void `esThdInit` (`esThd_T` *thd, void(*thdf)(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 `esThdQAddI` (`esThdQ_T` *thdQ, `esThd_T` *thd)

- Add a thread to the tail of 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 linked list.*

 - [bool_T](#) [esThdQIsEmpty](#) (const [esThdQ_T](#) *thdQ)
- Is thread queue empty.*

 - void [esSchedRdyAddl](#) ([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 [esSysTmrEnable](#) (void)
- Enable system timer tick events.*

 - void [esSysTmrDisable](#) (void)
- Disable system timer tick events.*

 - void [esTmrAddl](#) ([esTmr_T](#) *tmr, [esTick_T](#) tick, void(*fn)(void *), void *arg)
- Add a new timer.*

Scheduler

- 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 scheduler data structures ready for multi-threading.
- static void [schedRdyAddIniti](#) ([esThd_T](#) *thd)

Initialize scheduler ready structure during the thread add operation.
- static void [schedQml](#) (void)

Do the Quantum (Round-Robin) scheduling.
- static [PORT_C_INLINE](#) void [schedQmActivate](#) (void)

Activate system timer Quantum mode.
- static [PORT_C_INLINE](#) void [schedQmDeactivate](#) (void)

Deactivate system timer Quantum mode.
- static void [schedQmEvaluatel](#) ([esThd_T](#) *thd)

Evaluate if the system timer Quantum mode is needed.

Variables

- static [esThdQ_T](#) [gRdyQueue](#)

Ready Thread queue.
- static [uint_fast8_t](#) [gKernLockCnt](#)

Kernel Lock Counter.
- const volatile [esKernCtrl_T](#) [gKernCtrl](#)

Kernel control initialization.

System timer

- enum `sysTmrState` {
 `SYSTMR_ACTIVE`,
 `SYSTMR_INACTIVE` }
- System timer state enumeration.*
- typedef struct `sysTmr` `sysTmr_T`
 System Timer type.
- static void `sysTmrInit` (void)
 Initialize system timer hardware.
- static void `sysTmrTryDeactivate` (void)
 Try to deactivate system timer.
- static void `sysTmrTryActivate` (void)
 Try to activate 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?

System timer kernel thread

- static `sysTmr_T` `gSysTmr`
 Main System Timer structure.
- static `esTmr_T` `gTmrWait`
 Waiting list of timers to expire.
- static `esTmr_T` `gTmrPend`
 Timers pending to be inserted in waiting list.
- static `esThd_T` `gKTmrld`
 System timer thread Id.
- static `portStck_T` `gKTmrStck` [`KTMR_STCK_SIZE`]
 System timer thread stack.
- static void `tmrListAddSort` (`esTmr_T` *list, `esTmr_T` *tmr)
- static void `kTmrInit` (void)
 Initialization of System Timer Thread.
- static void `kTmr` (void *arg)
 System timer thread code.

Idle kernel thread

- static `esThd_T gKIdleId`
Idle thread Id.
- static `portStck_T gKIdleStck [KIDLE_STCK_SIZE]`
Idle thread stack.
- static void `kIdleInit` (void)
Initialization of Idle thread.
- static void `kIdle` (void *arg)
Idle thread code.

11.6.1 Detailed Description

Implementation of port independent code.

Author

Nenad Radulovic

11.6.2 Variable Documentation

11.6.2.1 `sysTmr_T gSysTmr` [static]

Initial value:

```
= {
    OU,
    OU,
}
```

Main System Timer structure.

11.6.2.2 `esTmr_T gTmrWait` [static]

Initial value:

```
= {
    {
        &gTmrWait,
        &gTmrWait
    },

    UINT_FAST8_MAX,

    NULL,
    NULL
}
```

Waiting list of timers to expire.

11.6.2.3 `esTmr_T gTmrPend` [static]

Initial value:

```

= {
    {
        &gTmrPend,
        &gTmrPend
    },
    0U,
    NULL,
    NULL
}

```

Timers pending to be inserted in waiting list.

11.6.2.4 esThd_T gKTmrId [static]

System timer thread Id.

11.6.2.5 portStck_T gKTmrStck[KTMR_STCK_SIZE] [static]

System timer thread stack.

11.6.2.6 esThd_T gKIdleId [static]

Idle thread Id.

11.6.2.7 portStck_T gKIdleStck[KIDLE_STCK_SIZE] [static]

Idle thread stack.

11.6.2.8 uint_fast8_t gKernLockCnt [static]

Kernel Lock Counter.

11.7 kernel.dox File Reference

11.8 kernel.h File Reference

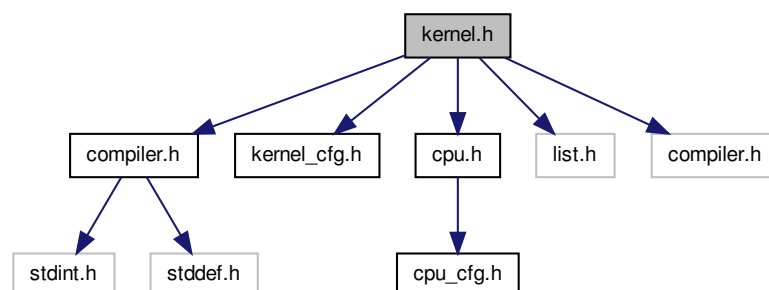
Interface of kernel.

```

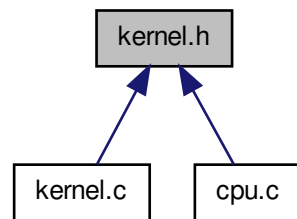
#include "compiler.h"
#include "kernel_cfg.h"
#include "cpu.h"
#include "list.h"

```

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 [esTmr](#)
Timer structure.
- struct [esTmr::tmrL](#)
Timer linked list structure.
- struct [esThdQ](#)
Thread Queue structure.
- struct [esThdQ::prioBM](#)
Priority Bit Map structure.
- struct [esKernCtrl](#)
Kernel control block structure.

Macros

Kernel identification and version number

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

Error checking

Some basic infrastructure for error checking

These macros provide basic detection of errors. For more details see [Error checking](#).

- #define [ES_ASSERT](#)(expr)
Generic assert macro.
- #define [ES_API_OBLIGATION](#)(expr) expr
Execute code to fulfill the contract.
- #define [ES_API_REQUIRE](#)(expr) [ES_ASSERT](#)(expr)
Make sure the caller has fulfilled all contract preconditions.
- #define [ES_API_ENSURE](#)(expr) [ES_ASSERT](#)(expr)
Make sure the callee has fulfilled all contract postconditions.

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 [esKernIsrPrologue](#) (void)
Enter Interrupt Service Routine.
- void [esKernIsrEpilogue](#) (void)
Exit Interrupt Service Routine.

Scheduler notification and invocation

- void [esSchedRdyAdd](#) ([esThd_T](#) *thd)
Add thread `thd` to the ready thread list and notify the scheduler.
- void [esSchedRdyRm](#) ([esThd_T](#) *thd)
Remove thread `thd` from the ready thread list and notify the scheduler.
- void [esSchedYield](#) (void)
Force the scheduler invocation which will evaluate all ready threads and switch to ready thread with the highest priority.
- void [esSchedYieldIsr](#) (void)
Force the scheduler invocation which will evaluate all ready threads and switch to ready thread with the highest priority.

System timer

- void [esSysTmrEnable](#) (void)
Enable system timer tick events.
- void [esSysTmrDisable](#) (void)
Disable system timer tick events.

Kernel hook functions

- PORT_C_NORETURN void [userAssert](#) (const char *funcName, const char *expr)
An assertion has failed. This function should inform the user about failed assertion.
- void [userSysTmr](#) (void)
System timer hook function, called from system system timer ISR function.
- void [userKernInit](#) (void)
Kernel initialization hook function, called from [esKernInit\(\)](#) function.
- void [userKernStart](#) (void)
Kernel start hook function, called from [esKernStart\(\)](#) function.
- void [userThdInitEnd](#) (void)
Thread initialization end hook function, called from [esThdInit\(\)](#) function.
- void [userThdTerm](#) (void)
Thread terminate hook function, called from [esThdTerm\(\)](#) or when a thread terminates itself.
- void [userCtxSw](#) ([esThd_T](#) *oldThd, [esThd_T](#) *newThd)
Kernel context switch hook function, called from [esSchedYield\(\)](#) and [esSchedYieldIsr\(\)](#) functions.

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.
- `void esKernLockEnterl (void)`
To do.
- `void esKernLockExitl (void)`
To do.

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(*thdf)(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 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.

Timer services

- typedef struct [esTmr](#) [esTmr_T](#)
Timer type.
- void [esTmrAddl](#) ([esTmr_T](#) *tmr, [esTick_T](#) tick, void(*fn)(void *), void *arg)
Add a new timer.

Thread Queue management

- #define [PRIO_BM_GRP_IND](#) (([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 [esThdQAddl](#) ([esThdQ_T](#) *thdQ, [esThd_T](#) *thd)
Add a thread to the tail of 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 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_INIT](#) = 0x04U,
 [ES_KERN_INACTIVE](#) = 0x05U }
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.

11.8.1 Detailed Description

Interface of kernel.

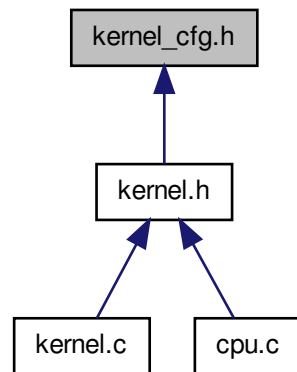
Author

Nenad Radulovic

11.9 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_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_SYSTMTR_MODE` 1U
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 end hook function.
- #define `CFG_HOOK_THD_TERM` 0U
Thread termination hook function.
- #define `CFG_HOOK_CTX_SW` 0U
Context switch hook function.

11.9.1 Detailed Description

Configuration of Kernel.

Author

Nenad Radulovic

11.10 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_SYSTMTR_EVENT_FREQUENCY` 100UL
The frequency of system tick event.

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.

11.10.1 Detailed Description

Configuration of Kernel - Template.

Author

Nenad Radulovic

Index

- `__attribute__`
 - CPU port interface, [54](#)
- arg
 - esTmr, [70](#)
- bit
 - esThdQ::prioBM, [72](#)
- bitGrp
 - esThdQ::prioBM, [72](#)
- bool_T
 - Compiler port, [44](#)
- boolType
 - Compiler port, [45](#)
- CFG_API_VALIDATION
 - Configuration, [39](#)
 - Kernel configuration, [61](#)
- CFG_CRITICAL_PRIO
 - CPU port configuration, [59](#)
- CFG_HOOK_CTX_SW
 - Configuration, [40](#)
 - Kernel configuration, [62](#)
- CFG_HOOK_KERN_INIT
 - Configuration, [39](#)
 - Kernel configuration, [61](#)
- CFG_HOOK_KERN_START
 - Configuration, [39](#)
- CFG_HOOK_THD_TERM
 - Configuration, [39](#)
 - Kernel configuration, [62](#)
- CFG_SCHED_PRIO_LVL
 - Configuration, [39](#)
 - Kernel configuration, [61](#)
- CFG_SYSTMTR_MODE
 - Configuration, [39](#)
- CPU port configuration, [59](#)
 - CFG_CRITICAL_PRIO, [59](#)
- CPU port interface, [46](#)
 - `__attribute__`, [54](#)
 - gPortIsrNesting_, [56](#)
 - PORT_CRITICAL_DECL, [51](#)
 - PORT_CRITICAL_EXIT, [51](#)
 - PORT_CTX_INIT, [52](#)
 - PORT_CTX_SW, [53](#)
 - PORT_CTX_SW_ISR, [53](#)
 - PORT_INIT, [53](#)
 - PORT_INIT_EARLY, [53](#)
 - PORT_INIT_LATE, [53](#)
 - PORT_INT_DISABLE, [50](#)
 - PORT_ISR_ENTER, [50](#)
 - PORT_ISR_EXIT, [50](#)
 - PORT_ISR_IS_LAST, [50](#)
 - PORT_PWR2, [51](#)
 - PORT_STCK_SIZE, [53](#)
 - PORT_SYSTMTR_ENABLE, [52](#)
 - PORT_SYSTMTR_INIT, [51](#)
 - PORT_SYSTMTR_RELOAD, [52](#)
 - PORT_SYSTMTR_TERM, [51](#)
 - PORT_THD_START, [53](#)
 - portCtxInit_, [55](#)
 - portCtxSw_, [55](#)
 - portCtxSwIsr_, [55](#)
 - portFindLastSet_, [54](#)
 - portInit_, [56](#)
 - portInitEarly_, [55](#)
 - portInitLate_, [56](#)
 - portIntDisable_, [54](#)
 - portIntGet_, [54](#)
 - portIntSet_, [54](#)
 - portReg_T, [53](#)
 - portStck_T, [53](#)
 - portSysTmrDisable_, [55](#)
 - portSysTmrEnable_, [55](#)
 - portSysTmrInit_, [54](#)
 - portSysTmrIsrDisable_, [55](#)
 - portSysTmrIsrEnable_, [55](#)
 - portSysTmrReload_, [55](#)
 - portSysTmrTerm_, [54](#)
 - portThdStart_, [55](#)
 - pwr2LKP, [56](#)
- CPU port internals, [57](#)
 - gPortIsrNesting_, [58](#)
 - portCtxInit_, [57](#)
 - portFindLastSet_, [57](#)
 - pwr2LKP, [58](#)
- Compiler port, [42](#)
 - bool_T, [44](#)
 - boolType, [45](#)
 - FALSE, [45](#)
 - PORT_C_ALIGNED, [44](#)
 - PORT_C_FUNC, [44](#)
 - PORT_C_INLINE, [43](#)
 - PORT_C_NAKED, [44](#)
 - PORT_C_WEAK, [44](#)
 - PORT_HWREG_SET, [44](#)
 - TRUE, [45](#)
- compiler.h, [76](#)
- Configuration, [38](#)
 - CFG_API_VALIDATION, [39](#)
 - CFG_HOOK_CTX_SW, [40](#)
 - CFG_HOOK_KERN_INIT, [39](#)
 - CFG_HOOK_KERN_START, [39](#)
 - CFG_HOOK_THD_TERM, [39](#)
 - CFG_SCHED_PRIO_LVL, [39](#)
 - CFG_SYSTMTR_MODE, [39](#)
- cprio
 - esThd, [66](#)
- cpu.c, [79](#)
- cpu.h, [79](#)
 - reg, [83](#)

cpu_cfg.h, [84](#)
 cthd
 esKernCtrl, [64](#)

 DLIST_ENTRY_INIT
 Internals, [33](#)
 DLIST_ENTRY_NEXT
 Internals, [33](#)
 DLIST_ENTRY_PREV
 Internals, [33](#)
 DLIST_ENTRY_RM
 Internals, [33](#)
 DLIST_IS_ENTRY_LAST
 Internals, [33](#)

 ES_KERN_INACTIVE
 Interface, [18](#)
 ES_KERN_INIT
 Interface, [18](#)
 ES_KERN_INTSRV_LOCK
 Interface, [18](#)
 ES_KERN_INTSRV_RUN
 Interface, [18](#)
 ES_KERN_LOCK
 Interface, [18](#)
 ES_KERN_RUN
 Interface, [18](#)
 ES_API_ENSURE
 Interface, [17](#)
 ES_API_OBLIGATION
 Interface, [17](#)
 ES_API_REQUIRE
 Interface, [17](#)
 ES_ASSERT
 Interface, [17](#)
 ES_CRITICAL_DECL
 Interface, [16](#)
 ES_CRITICAL_ENTER
 Interface, [17](#)
 ES_CRITICAL_EXIT
 Interface, [17](#)
 ES_KERNEL_ID
 Interface, [16](#)
 ES_KERNEL_VER
 Interface, [16](#)
 ES_STCK_SIZE
 Interface, [17](#)
 esKernCtrl, [63](#)
 cth, [64](#)
 pthd, [64](#)
 state, [64](#)
 esKernCtrl_T
 Interface, [18](#)
 esKernInit
 Interface, [18](#)
 esKernIsrEpiloguel
 Interface, [20](#)
 esKernIsrProloguel
 Interface, [19](#)

 esKernLockEnterl
 Interface, [20](#)
 esKernLockExitl
 Interface, [20](#)
 esKernStart
 Interface, [19](#)
 esKernState
 Interface, [18](#)
 esKernState_T
 Interface, [18](#)
 esKernSysTmrl
 Interface, [19](#)
 esSchedRdyAddl
 Interface, [25](#)
 esSchedRdyRml
 Interface, [26](#)
 esSchedYieldl
 Interface, [26](#)
 esSchedYieldIsrl
 Interface, [26](#)
 esStck_T
 Interface, [18](#)
 esSysTmrDisable
 Interface, [27](#)
 esSysTmrEnable
 Interface, [27](#)
 esThd, [64](#)
 cprio, [66](#)
 prio, [66](#)
 qCnt, [66](#)
 qRld, [66](#)
 signature, [66](#)
 stck, [66](#)
 thdL, [66](#)
 esThd::thdL, [73](#)
 next, [75](#)
 prev, [75](#)
 q, [74](#)
 esThd_T
 Interface, [18](#)
 esThdGetId
 Interface, [21](#)
 esThdGetPrio
 Interface, [22](#)
 esThdInit
 Interface, [20](#)
 esThdPost
 Interface, [23](#)
 esThdPostl
 Interface, [22](#)
 esThdQ, [67](#)
 grp, [68](#)
 prioOcc, [68](#)
 signature, [68](#)
 esThdQ::prioBM, [71](#)
 bit, [72](#)
 bitGrp, [72](#)
 esThdQ_T

- Interface, [18](#)
- esThdQAddI
 - Interface, [23](#)
- esThdQFetchI
 - Interface, [24](#)
- esThdQFetchRotatel
 - Interface, [25](#)
- esThdQInit
 - Interface, [23](#)
- esThdQIsEmpty
 - Interface, [25](#)
- esThdQRml
 - Interface, [24](#)
- esThdSetPriol
 - Interface, [22](#)
- esThdTerm
 - Interface, [21](#)
- esThdWait
 - Interface, [23](#)
- esThdWaitI
 - Interface, [23](#)
- esTmr, [68](#)
 - arg, [70](#)
 - fn, [70](#)
 - rtick, [69](#)
 - tmrL, [69](#)
- esTmr::tmrL, [75](#)
 - next, [76](#)
 - prev, [76](#)
- esTmr_T
 - Interface, [18](#)
- esTmrAddI
 - Interface, [27](#)
- FALSE
 - Compiler port, [45](#)
- fn
 - esTmr, [70](#)
- gKIdleId
 - kernel.c, [90](#)
- gKIdleStck
 - kernel.c, [90](#)
- gKTmrId
 - kernel.c, [90](#)
- gKTmrStck
 - kernel.c, [90](#)
- gKernCtrl
 - Interface, [29](#)
- gKernLockCnt
 - kernel.c, [90](#)
- gPortIsrNesting_
 - CPU port interface, [56](#)
 - CPU port internals, [58](#)
- gRdyQueue
 - Internals, [36](#)
- gSysTmr
 - kernel.c, [89](#)
- gTmrPend
 - kernel.c, [89](#)
- gTmrWait
 - kernel.c, [89](#)
- grp
 - esThdQ, [68](#)
- Interface, [13](#)
 - ES_KERN_INACTIVE, [18](#)
 - ES_KERN_INIT, [18](#)
 - ES_KERN_INTSRV_LOCK, [18](#)
 - ES_KERN_INTSRV_RUN, [18](#)
 - ES_KERN_LOCK, [18](#)
 - ES_KERN_RUN, [18](#)
 - ES_API_ENSURE, [17](#)
 - ES_API_OBLIGATION, [17](#)
 - ES_API_REQUIRE, [17](#)
 - ES_ASSERT, [17](#)
 - ES_CRITICAL_DECL, [16](#)
 - ES_CRITICAL_ENTER, [17](#)
 - ES_CRITICAL_EXIT, [17](#)
 - ES_KERNEL_ID, [16](#)
 - ES_KERNEL_VER, [16](#)
 - ES_STCK_SIZE, [17](#)
 - esKernCtrl_T, [18](#)
 - esKernInit, [18](#)
 - esKernIsrEpilogueI, [20](#)
 - esKernIsrPrologueI, [19](#)
 - esKernLockEnterI, [20](#)
 - esKernLockExitI, [20](#)
 - esKernStart, [19](#)
 - esKernState, [18](#)
 - esKernState_T, [18](#)
 - esKernSysTmrI, [19](#)
 - esSchedRdyAddI, [25](#)
 - esSchedRdyRml, [26](#)
 - esSchedYieldI, [26](#)
 - esSchedYieldIsrI, [26](#)
 - esStck_T, [18](#)
 - esSysTmrDisable, [27](#)
 - esSysTmrEnable, [27](#)
 - esThd_T, [18](#)
 - esThdGetId, [21](#)
 - esThdGetPrio, [22](#)
 - esThdInit, [20](#)
 - esThdPost, [23](#)
 - esThdPostI, [22](#)
 - esThdQ_T, [18](#)
 - esThdQAddI, [23](#)
 - esThdQFetchI, [24](#)
 - esThdQFetchRotatel, [25](#)
 - esThdQInit, [23](#)
 - esThdQIsEmpty, [25](#)
 - esThdQRml, [24](#)
 - esThdSetPriol, [22](#)
 - esThdTerm, [21](#)
 - esThdWait, [23](#)
 - esThdWaitI, [23](#)
 - esTmr_T, [18](#)
 - esTmrAddI, [27](#)

- gKernCtrl, 29
- PRIO_BM_GRP_INDX, 17
- userAssert, 27
- userCtxSw, 28
- userKernInit, 28
- userKernStart, 28
- userSysTmr, 27
- userThdInitEnd, 28
- userThdTerm, 28
- Internals, 30
 - DLIST_ENTRY_INIT, 33
 - DLIST_ENTRY_NEXT, 33
 - DLIST_ENTRY_PREV, 33
 - DLIST_ENTRY_RM, 33
 - DLIST_IS_ENTRY_LAST, 33
 - gRdyQueue, 36
 - KIDLE_STCK_SIZE, 34
 - kIdle, 36
 - kIdleInit, 36
 - KTMR_STCK_SIZE, 34
 - kTmr, 36
 - kTmrInit, 36
 - prioBM_T, 34
 - prioBMClear, 34
 - prioBMGet, 35
 - prioBMInit, 34
 - prioBMIsEmpty, 35
 - prioBMSet, 34
 - SYSTM_ACTIVE, 34
 - SYSTM_INACTIVE, 34
 - SCHED_POWER_SAVE, 33
 - SYSTM_SCHED_QM_MSK, 32
 - SYSTM_USR_QM_MSK, 32
 - schedInit, 35
 - schedQmActivate, 36
 - schedQmDeactivate, 36
 - schedQmEvaluatel, 36
 - schedQml, 35
 - schedRdyAddInitl, 35
 - schedStart, 35
 - sysTmr_T, 34
 - sysTmrInit, 36
 - sysTmrState, 34
 - sysTmrTryActivate, 36
 - sysTmrTryDeactivate, 36
- KIDLE_STCK_SIZE
 - Internals, 34
- kIdle
 - Internals, 36
- kIdleInit
 - Internals, 36
- KTMR_STCK_SIZE
 - Internals, 34
- kTmr
 - Internals, 36
- kTmrInit
 - Internals, 36
- Kernel, 11
 - Kernel configuration, 60
 - CFG_API_VALIDATION, 61
 - CFG_HOOK_CTX_SW, 62
 - CFG_HOOK_KERN_INIT, 61
 - CFG_HOOK_THD_TERM, 62
 - CFG_SCHED_PRIO_LVL, 61
 - kernel-example.dox, 84
 - kernel.c, 85
 - gKIdleId, 90
 - gKIdleStck, 90
 - gKTmrId, 90
 - gKTmrStck, 90
 - gKernLockCnt, 90
 - gSysTmr, 89
 - gTmrPend, 89
 - gTmrWait, 89
 - kernel.dox, 90
 - kernel.h, 90
 - kernel_cfg.h, 95, 96
 - next
 - esThd::thdL, 75
 - esTmr::tmrL, 76
 - PORT_C_ALIGNED
 - Compiler port, 44
 - PORT_C_FUNC
 - Compiler port, 44
 - PORT_C_INLINE
 - Compiler port, 43
 - PORT_C_NAKED
 - Compiler port, 44
 - PORT_C_WEAK
 - Compiler port, 44
 - PORT_CRITICAL_DECL
 - CPU port interface, 51
 - PORT_CRITICAL_ENTER
 - CPU port interface, 51
 - PORT_CRITICAL_EXIT
 - CPU port interface, 51
 - PORT_CTX_INIT
 - CPU port interface, 52
 - PORT_CTX_SW
 - CPU port interface, 53
 - PORT_CTX_SW_ISR
 - CPU port interface, 53
 - PORT_FIND_LAST_SET
 - CPU port interface, 51
 - PORT_HWREG_SET
 - Compiler port, 44
 - PORT_INIT
 - CPU port interface, 53
 - PORT_INIT_EARLY
 - CPU port interface, 53
 - PORT_INIT_LATE
 - CPU port interface, 53
 - PORT_INT_DISABLE
 - CPU port interface, 50
 - PORT_ISR_ENTER

- CPU port interface, [50](#)
- PORT_ISR_EXIT
 - CPU port interface, [50](#)
- PORT_ISR_IS_LAST
 - CPU port interface, [50](#)
- PORT_PWR2
 - CPU port interface, [51](#)
- PORT_STCK_SIZE
 - CPU port interface, [53](#)
- PORT_SYSTMR_DISABLE
 - CPU port interface, [52](#)
- PORT_SYSTMR_ENABLE
 - CPU port interface, [52](#)
- PORT_SYSTMR_INIT
 - CPU port interface, [51](#)
- PORT_SYSTMR_RELOAD
 - CPU port interface, [52](#)
- PORT_SYSTMR_TERM
 - CPU port interface, [51](#)
- PORT_THD_START
 - CPU port interface, [53](#)
- PRIO_BM_GRP_INDX
 - Interface, [17](#)
- Port template, [41](#)
- portCtx, [70](#)
 - r0, [70](#)
- portCtxInit_
 - CPU port interface, [55](#)
 - CPU port internals, [57](#)
- portCtxSw_
 - CPU port interface, [55](#)
- portCtxSwlsr_
 - CPU port interface, [55](#)
- portFindLastSet_
 - CPU port interface, [54](#)
 - CPU port internals, [57](#)
- portInit_
 - CPU port interface, [56](#)
- portInitEarly_
 - CPU port interface, [55](#)
- portInitLate_
 - CPU port interface, [56](#)
- portIntDisable_
 - CPU port interface, [54](#)
- portIntGet_
 - CPU port interface, [54](#)
- portIntSet_
 - CPU port interface, [54](#)
- portReg_T
 - CPU port interface, [53](#)
- portStck, [71](#)
 - reg, [71](#)
- portStck_T
 - CPU port interface, [53](#)
- portSysTmrDisable_
 - CPU port interface, [55](#)
- portSysTmrEnable_
 - CPU port interface, [55](#)
- portSysTmrInit_
 - CPU port interface, [54](#)
- portSysTmrIsrDisable_
 - CPU port interface, [55](#)
- portSysTmrIsrEnable_
 - CPU port interface, [55](#)
- portSysTmrReload_
 - CPU port interface, [55](#)
- portSysTmrTerm_
 - CPU port interface, [54](#)
- portThdStart_
 - CPU port interface, [55](#)
- prev
 - esThd::thdL, [75](#)
 - esTmr::tmrL, [76](#)
- prio
 - esThd, [66](#)
- prioBM_T
 - Internals, [34](#)
- prioBMClear
 - Internals, [34](#)
- prioBMGet
 - Internals, [35](#)
- prioBMInit
 - Internals, [34](#)
- prioBMIsEmpty
 - Internals, [35](#)
- prioBMSet
 - Internals, [34](#)
- prioOcc
 - esThdQ, [68](#)
- pthd
 - esKernCtrl, [64](#)
- pwr2LKP
 - CPU port interface, [56](#)
 - CPU port internals, [58](#)
- q
 - esThd::thdL, [74](#)
- qCnt
 - esThd, [66](#)
- qRld
 - esThd, [66](#)
- r0
 - portCtx, [70](#)
- reg
 - cpu.h, [83](#)
 - portStck, [71](#)
- rtick
 - esTmr, [69](#)
- SYSTMR_ACTIVE
 - Internals, [34](#)
- SYSTMR_INACTIVE
 - Internals, [34](#)
- SCHED_POWER_SAVE
 - Internals, [33](#)
- SYSTMR_SCHED_QM_MSK

- Internals, [32](#)
- SYSTMR_USR_QM_MSK
 - Internals, [32](#)
- schedInit
 - Internals, [35](#)
- schedQmActivate
 - Internals, [36](#)
- schedQmDeactivate
 - Internals, [36](#)
- schedQmEvaluateI
 - Internals, [36](#)
- schedQmI
 - Internals, [35](#)
- schedRdyAddInitI
 - Internals, [35](#)
- schedStart
 - Internals, [35](#)
- signature
 - esThd, [66](#)
 - esThdQ, [68](#)
- state
 - esKernCtrl, [64](#)
- stck
 - esThd, [66](#)
- sysTmr, [72](#)
 - tick, [73](#)
 - tmr, [73](#)
- sysTmr_T
 - Internals, [34](#)
- sysTmrInit
 - Internals, [36](#)
- sysTmrState
 - Internals, [34](#)
- sysTmrTryActivate
 - Internals, [36](#)
- sysTmrTryDeactivate
 - Internals, [36](#)
- TRUE
 - Compiler port, [45](#)
- thdL
 - esThd, [66](#)
- tick
 - sysTmr, [73](#)
- tmr
 - sysTmr, [73](#)
- tmrL
 - esTmr, [69](#)
- userAssert
 - Interface, [27](#)
- userCtxSw
 - Interface, [28](#)
- userKernInit
 - Interface, [28](#)
- userKernStart
 - Interface, [28](#)
- userSysTmr
 - Interface, [27](#)
- userThdInitEnd
 - Interface, [28](#)
- userThdTerm
 - Interface, [28](#)