[Configuration] **Archivo** FreeRTOSConfig.h

```
#ifndef FREERTOS_CONFIG_H
#define FREERTOS_CONFIG_H
#define configUSE_PREEMPTION                1
#define configUSE_IDLE_HOOK                 0
#define configUSE_TICK_HOOK                 0
#define configCPU_CLOCK_HZ                  58982400
#define configTICK_RATE_HZ                  250
#define configMAX_PRIORITIES                5
#define configMINIMAL_STACK_SIZE            128
#define configTOTAL_HEAP_SIZE               10240
#define configMAX_TASK_NAME_LEN             16
#define configUSE_TRACE_FACILITY            0
#define configUSE_16_BIT_TICKS              0
#define configIDLE_SHOULD_YIELD             1
#define configUSE_MUTEXES                   0
#define configUSE_RECURSIVE_MUTEXES         0
#define configUSE_COUNTING_SEMAPHORES       0
#define configUSE_ALTERNATIVE_API           0
#define configCHECK_FOR_STACK_OVERFLOW      0
#define configQUEUE_REGISTRY_SIZE           10
#define configGENERATE_RUN_TIME_STATS       0
#define configUSE_CO_ROUTINES               0
#define configMAX_CO_ROUTINE_PRIORITIES     1
#define configUSE_TIMERS                    1
#define configTIMER_TASK_PRIORITY           3
#define configTIMER_QUEUE_LENGTH            10
#define configTIMER_TASK_STACK_DEPTH        configMINIMAL_STACK_SIZE
#define configKERNEL_INTERRUPT_PRIORITY       [dependent of processor]
#define configMAX_SYSCALL_INTERRUPT_PRIORITY [dependent on processor and application]
#define configASSERT( ( x ) )               if( ( x ) == 0 ) vCallAssert(
__FILE__, __LINE__ )
#define INCLUDE_vTaskPrioritySet               1
#define INCLUDE_uxTaskPriorityGet              1
#define INCLUDE_vTaskDelete                    1
#define INCLUDE_vTaskCleanUpResources          0
#define INCLUDE_vTaskSuspend                   1
#define INCLUDE_vResumeFromISR                 1
#define INCLUDE_vTaskDelayUntil                1
#define INCLUDE_vTaskDelay                     1
#define INCLUDE_xTaskGetSchedulerState         1
#define INCLUDE_xTaskGetCurrentTaskHandle      1
#define INCLUDE_uxTaskGetStackHighWaterMark    0
#define INCLUDE_xTaskGetIdleTaskHandle         0
#define INCLUDE_xTimerGetTimerDaemonTaskHandle 0
#define INCLUDE_pcTaskGetTaskName              0
#endif /* FREERTOS_CONFIG_H */
```

Hook Functions

The Idle Task Hook

```
void vApplicationIdleHook( void );
```

The Tick Task Hook

```
void vApplicationTickHook (void);
```

Malloc Failed Hook Function

```
void vApplicationMallocFailedHook( void );
```

Stack Overflow Hook Function

```
void vApplicationStackOverflowHook( xTaskHandle xTask,
                                    signed portCHAR *pcTaskName );
```

FreeRTOS Constants

tskIDLE_PRIORITY	(0)
tskMAX_TASK_NAME_LEN	(16)
portTICK_RATE_MS	
portMAX_DELAY	
pdTRUE	(1)
pdFALSE	(0)
pdPASS	(1)
pdFAIL	(0)
errQUEUE_EMPTY	(0)
errQUEUE_FULL	(0)
errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY	(-1)
errNO_TASK_TO_RUN	(-2)
errQUEUE_BLOCKED	(-4)
errQUEUE_YIELD	(-5)

Port Data Types

portTickType
portBASE_TYPE

Handlers

xTaskHandle
xQueueHandle
xSemaphoreHandle
xTimerHandle
xCoRoutineHandle

Task Creation [API]

xTaskCreate [Task Creation] task. h

```
portBASE_TYPE xTaskCreate( pdTASK_CODE pvTaskCode,
                           const portCHAR * const pcName,
                           unsigned portSHORT usStackDepth,
                           void *pvParameters,
                           unsigned portBASE_TYPE uxPriority,
                           xTaskHandle *pvCreatedTask);
```

vTaskDelete [Task Creation] task. h

```
void vTaskDelete( xTaskHandle pxTask );
```

Task Control [API]

vTaskDelay [Task Control] task.h

```
void vTaskDelay( portTickType xTicksToDelay );
```

vTaskDelayUntil [Task Control] task.h

```
void vTaskDelayUntil( portTickType *pxPreviousWakeTime, portTickType
                      xTimeIncrement );
```

uxTaskPriorityGet [Task Control] task.h
```
unsigned portBASE_TYPE uxTaskPriorityGet( xTaskHandle pxTask );
```

vTaskPrioritySet [Task Control] task.h
```
void vTaskPrioritySet(xTaskHandle pxTask, unsigned portBASE_TYPE uxNewPriority );
```

vTaskSuspend [Task Control] task.h
```
void vTaskSuspend( xTaskHandle pxTaskToSuspend );
```

vTaskResume [Task Control] task.h
```
void vTaskResume( xTaskHandle pxTaskToResume );
```

xTaskResumeFromISR [Task Control] task.h
```
portBASE_TYPE xTaskResumeFromISR( xTaskHandle pxTaskToResume );
```

uxTaskGetStackHighWaterMark [Task Control] task. h
```
unsigned portBASE_TYPE uxTaskGetStackHighWaterMark( xTaskHandle xTask );
```

vTaskSetApplicationTaskTag [Task Control] task. h
```
void vTaskSetApplicationTaskTag(
                        xTaskHandle xTask,
                        pdTASK_HOOK_CODE pxTagValue );
```

xTaskGetApplicationTaskTag [Task Control] task. h
```
pdTASK_HOOK_CODE xTaskGetApplicationTaskTag( xTaskHandle xTask );
```

xTaskCallApplicationTaskHook [Task Control] task. h
```
portBASE_TYPE xTaskCallApplicationTaskHook(
                                    xTaskHandle xTask,
                                     void *pvParameter );
```

Kernel Control [API] task.h

taskYIELD

taskENTER_CRITICAL

taskEXIT_CRITICAL

taskDISABLE_INTERRUPTS

taskENABLE_INTERRUPTS

vTaskStartScheduler [Kernel Control] task.h
```
void vTaskStartScheduler( void );
```

vTaskEndScheduler [Kernel Control] task.h
```
void vTaskEndScheduler( void );
```

vTaskSuspendAll [Kernel Control] task.h
```
void vTaskSuspendAll( void );
```

xTaskResumeAll [Kernel Control] task.h
```
portBASE_TYPE xTaskResumeAll( void );
```

Task Utilities [API]

xTaskGetTickCount task.h
```
volatile portTickType xTaskGetTickCount( void );
```

xTaskGetTickCountFromISR task.h
```
volatile portTickType xTaskGetTickCountFromISR( void );
```

uxTaskGetNumberOfTasks task.h
```
unsigned portBASE_TYPE uxTaskGetNumberOfTasks( void );
```

vTaskList task.h
```
void vTaskList( portCHAR *pcWriteBuffer );
```

vTaskStartTrace task.h
```
void vTaskStartTrace( portCHAR * pcBuffer, unsigned portLONG ulBufferSize );
```

ulTaskEndTrace task.h
```
unsigned portLONG ulTaskEndTrace( void );
```

xTaskGetCurrentTaskHandle task.h
```
xTaskHandle xTaskGetCurrentTaskHandle( void );
```

xTaskGetIdleTaskHandle task.h
```
xTaskHandle xTaskGetIdleTaskHandle( void );
```

pcTaskGetTaskName task.h
```
signed char * pcTaskGetTaskName( xTaskHandle xTaskToQuery );
```

xTaskGetSchedulerState task.h
```
portBASE_TYPE xTaskGetSchedulerState( void );
```

vTaskGetRunTimeStats task.h
```
void vTaskGetRunTimeStats( portCHAR *pcWriteBuffer );
```

Queue Management [API]

uxQueueMessagesWaiting queue.h
```
unsigned portBASE_TYPE uxQueueMessagesWaiting( xQueueHandle xQueue );
```

uxQueueMessagesWaitingFromISR queue.h
```
unsigned portBASE_TYPE uxQueueMessagesWaiting( xQueueHandle xQueue );
```

vQueueDelete queue.h
```
void vQueueDelete( xQueueHandle xQueue );
```

xQueueCreate [Queue Management] queue.h
```
xQueueHandle xQueueCreate(
                          unsigned portBASE_TYPE uxQueueLength,
                          unsigned portBASE_TYPE uxItemSize
                          );
```

xQueueSend [Queue Management] queue.h
```
portBASE_TYPE xQueueSend( xQueueHandle xQueue,
                          const void * pvItemToQueue,
                          portTickType xTicksToWait );
```

xQueueReceive [Queue Management] queue.h

```
portBASE_TYPE xQueueReceive(
                                xQueueHandle xQueue,
                                void *pcBuffer,
                                portTickType xTicksToWait
                            );
```

xQueueSendFromISR [Queue Management] queue.h

```
portBASE_TYPE xQueueSendFromISR(
                                xQueueHandle pxQueue,
                                const void *pvItemToQueue,
                                portBASE_TYPE xTaskPreviouslyWoken
                                );
```

xQueueReceiveFromISR [Queue Management] queue.h

```
portBASE_TYPE xQueueReceiveFromISR(
                                    xQueueHandle pxQueue,
                                    void *pcBuffer,
                                    portBASE_TYPE *pxTaskWoken
                                    );
```

xQueueReset queue.h

```
portBASE_TYPE xQueueReset( xQueueHandle xQueue );
```

xQueueSendToBack [Queue Management] queue.h

```
portBASE_TYPE xQueueSendToBack(
                                  xQueueHandle xQueue,
                                  const void * pvItemToQueue,
                                  portTickType xTicksToWait
                                );
```

xQueueSendToFront [Queue Management] queue.h

```
portBASE_TYPE xQueueSendToToFront(
                                     xQueueHandle xQueue,
                                     const void * pvItemToQueue,
                                     portTickType xTicksToWait
                                   );
```

xQueuePeek [Queue Management] queue.h

```
portBASE_TYPE xQueuePeek(
                             xQueueHandle xQueue,
                             void *pvBuffer,
                             portTickType xTicksToWait
                           );
```

xQueueSendToBackFromISR [Queue Management] queue.h

```
portBASE_TYPE xQueueSendToBackFromISR
                    (
                        xQueueHandle pxQueue,
                        const void *pvItemToQueue,
                        portBASE_TYPE *pxHigherPriorityTaskWoken
                    );
```

xQueueSendToFrontFromISR [Queue Management] queue.h
```
portBASE_TYPE xQueueSendToFrontFromISR
                  (
                      xQueueHandle pxQueue,
                      const void *pvItemToQueue,
                      portBASE_TYPE *pxHigherPriorityTaskWoken
                  );
```

vQueueAddToRegistry [Queue Management] queue.h
```
void vQueueAddToRegistry(
                              xQueueHandle xQueue,
                              signed portCHAR *pcQueueName,
                          );
```

vQueueUnregisterQueue [Queue Management] queue.h
```
 void vQueueUnregisterQueue( xQueueHandle xQueue );
```

xQueueIsQueueEmptyFromISR queue.h
```
portBASE_TYPE xQueueIsQueueEmptyFromISR( const xQueueHandle pxQueue );
```

xQueueIsQueueFullFromISR queue.h
```
portBASE_TYPE xQueueIsQueueFullFromISR( const xQueueHandle pxQueue );
```

Semaphores [API]

vSemaphoreCreateBinary [Semaphores] semphr.h
```
vSemaphoreCreateBinary( xSemaphoreHandle xSemaphore )
```

xSemaphoreCreateCounting [Semaphores] semphr. h
```
xSemaphoreHandle xSemaphoreCreateCounting
               (
                  unsigned portBASE_TYPE uxMaxCount,
                  unsigned portBASE_TYPE uxInitialCount
               )
```

xSemaphoreCreateMutex [Semaphores] semphr. h
```
xSemaphoreHandle xSemaphoreCreateMutex( void )
```

xSemaphoreCreateRecursiveMutex [Semaphores] semphr. h
```
xSemaphoreHandle xSemaphoreCreateRecursiveMutex( void )
```

xSemaphoreTake [Semaphores] semphr.h
```
xSemaphoreTake(
                   xSemaphoreHandle xSemaphore,
                   portTickType xBlockTime
                   )
```

xSemaphoreTakeRecursive [Semaphores] semphr. h
```
xSemaphoreTakeRecursive( xSemaphoreHandle xMutex,
                         portTickType xBlockTime )
```

xSemaphoreGive [Semaphores] semphr.h
```
xSemaphoreGive( xSemaphoreHandle xSemaphore )
```

xSemaphoreGiveRecursive [Semaphores] semphr. h
```
xSemaphoreGiveRecursive( xSemaphoreHandle xMutex )
```

xSemaphoreGiveFromISR [Semaphores] semphr.h
```
xSemaphoreGiveFromISR(    xSemaphoreHandle xSemaphore,
                          portBASE_TYPE xTaskPreviouslyWoken
                          )
```

vSemaphoreDelete semphr.h
```
void vSemaphoreDelete( xSemaphoreHandle xSemaphore );
```

xSemaphoreGetMutexHolder [Semaphores] semphr. h
```
xSemaphoreGetMutexHolder( xSemaphoreHandle xMutex );
```

Software Timers [API]

xTimerCreate [Timer API] timers.h
```
xTimerHandle xTimerCreate
              ( const signed char *pcTimerName,
                portTickType xTimerPeriod,
                unsigned portBASE_TYPE uxAutoReload,
                void * pvTimerID,
                tmrTIMER_CALLBACK pxCallbackFunction );
```

xTimerIsTimerActive [Timer API] timers.h
```
portBASE_TYPE xTimerIsTimerActive( xTimerHandle xTimer );
```

pvTimerGetTimerID [Timer API] timers.h
```
void *pvTimerGetTimerID( xTimerHandle xTimer );
```

xTimerStart [Timer API] timers.h
```
portBASE_TYPE xTimerStart( xTimerHandle xTimer,
                           portTickType xBlockTime );
```

xTimerStop [Timer API] timers.h
```
portBASE_TYPE xTimerStop( xTimerHandle xTimer,
                          portTickType xBlockTime );
```

xTimerChangePeriod [Timer API] timers.h
```
portBASE_TYPE xTimerChangePeriod( xTimerHandle xTimer,
                                  portTickType xNewPeriod,
                                  portTickType xBlockTime );
```

xTimerDelete [Timer API] timers.h
```
portBASE_TYPE xTimerDelete( xTimerHandle xTimer,
                            portTickType xBlockTime );
```

xTimerReset [Timer API] timers.h
```
portBASE_TYPE xTimerReset( xTimerHandle xTimer,
                           portTickType xBlockTime );
```

xTimerStartFromISR [Timer API] timers.h
```
portBASE_TYPE xTimerStartFromISR
               (  xTimerHandle xTimer,
                  portBASE TYPE *pxHigherPriorityTaskWoken );
```
xTimerStopFromISR [Timer API] timers.h

```
portBASE_TYPE xTimerStopFromISR
             (   xTimerHandle xTimer,
                 portBASE_TYPE *pxHigherPriorityTaskWoken );
```

xTimerChangePeriodFromISR [Timer API] timers.h
```
portBASE_TYPE xTimerChangePeriodFromISR
           (  xTimerHandle xTimer,
              portTickType xNewPeriod,
              portBASE_TYPE *pxHigherPriorityTaskWoken );
```

xTimerResetFromISR [Timer API] timers.h
```
portBASE_TYPE xTimerResetFromISR
                (  xTimerHandle xTimer,
                   portBASE_TYPE *pxHigherPriorityTaskWoken );
```

xTimerGetTimerDaemonTaskHandle timers.h
```
xTaskHandle xTimerGetTimerDaemonTaskHandle( void );
```

FreeRTOS-MPU Specific Functions [API]

xTaskCreateRestricted [FreeRTOS-MPU Specific] task. h
```
portBASE_TYPE xTaskCreateRestricted(
                           xTaskParameters *pxTaskDefinition,
                           xTaskHandle *pxCreatedTask );
```

vTaskAllocateMPURegions [FreeRTOS-MPU Specific] task. h
```
void vTaskAllocateMPURegions(
               xTaskHandle xTaskToModify,
               const xMemoryRegion * const xRegions );
```

portSWITCH_TO_USER_MODE [FreeRTOS-MPU Specific] task. h
```
void portSWITCH_TO_USER_MODE( void );
```

Co-routine specific [API]

xCoRoutineCreate [Co-Routine Specific] croutine.h
```
portBASE_TYPE xCoRoutineCreate
                  (   crCOROUTINE_CODE pxCoRoutineCode,
                      unsigned portBASE_TYPE uxPriority,
                      unsigned portBASE_TYPE uxIndex
                  );
```

crDELAY [Co-Routine Specific] croutine.h
```
void crDELAY( xCoRoutineHandle xHandle,
              portTickType xTicksToDelay )
```

crQUEUE_SEND [Co-Routine Specific] croutine.h
```
crQUEUE_SEND(
                  xCoRoutineHandle xHandle,
                  xQueueHandle pxQueue,
                  void *pvItemToQueue,
                  portTickType xTicksToWait,
                  portBASE_TYPE *pxResult );
```

crQUEUE_RECEIVE [Co-Routine Specific] croutine.h

```
void crQUEUE_RECEIVE(
                     xCoRoutineHandle xHandle,
                     xQueueHandle pxQueue,
                     void *pvBuffer,
                     portTickType xTicksToWait,
                     portBASE_TYPE *pxResult
                 );
```

crQUEUE_SEND_FROM_ISR [Co-Routine Specific] croutine.h

```
portBASE_TYPE crQUEUE_SEND_FROM_ISR
                      (
                          xQueueHandle pxQueue,
                          void *pvItemToQueue,
                          portBASE_TYPE xCoRoutinePreviouslyWoken
                      );
```

crQUEUE_RECEIVE_FROM_ISR [Co-Routine Specific] croutine.h

```
portBASE_TYPE crQUEUE_SEND_FROM_ISR
                       (
                           xQueueHandle pxQueue,
                           void *pvBuffer,
                           portBASE_TYPE * pxCoRoutineWoken
                       );
```

vCoRoutineSchedule [Co-Routine Specific] croutine.h

```
void vCoRoutineSchedule( void );
```

Driver [API] (Application Programming Interface)

uint32_t xxx_Read_Driver (uint8_t nombre, void *data, uint32_t tamaño_data,
uint32_t tamaño_bloque, uint32_t tiempo_espera, uint8_t es_int);

uint32_t xxx_Write_Driver (…);

uint32_t xxx_Open_Driver (…);

uint32_t xxx_Close_Driver (…);

void xxx_Init_Driver (…);

GPIO Registers

```
LPC_PINCON->PINSELx        x = 0,1,2,3,4,7,9,10  00=Prim. 01=1Alt.  10=2Alt. 11=3Alt.
LPC_PINCON->PINMODEx       x = 0,1,2,3,4,7,9     00=P.up  01=Repeat 10=N/C   11=P.down
LPC_PINCON->PINMODE_ODx    x = 0,1,2,3,4           0=Normal  1=OD
LPC_GPIOx->FIODIR          x = 0,1,2,3,4           0=In      1=Out
LPC_GPIOx->FIOCLR          x = 0,1,2,3,4           0=s/c     1=CLR
LPC_GPIOx->FIOSET          x = 0,1,2,3,4           0=s/c     1=SET
LPC_GPIOx->FIOMASK         x = 0,1,2,3,4           0=Enable  1=Disable
LPC_GPIOx->FIOPIN          x = 0,1,2,3,4
```