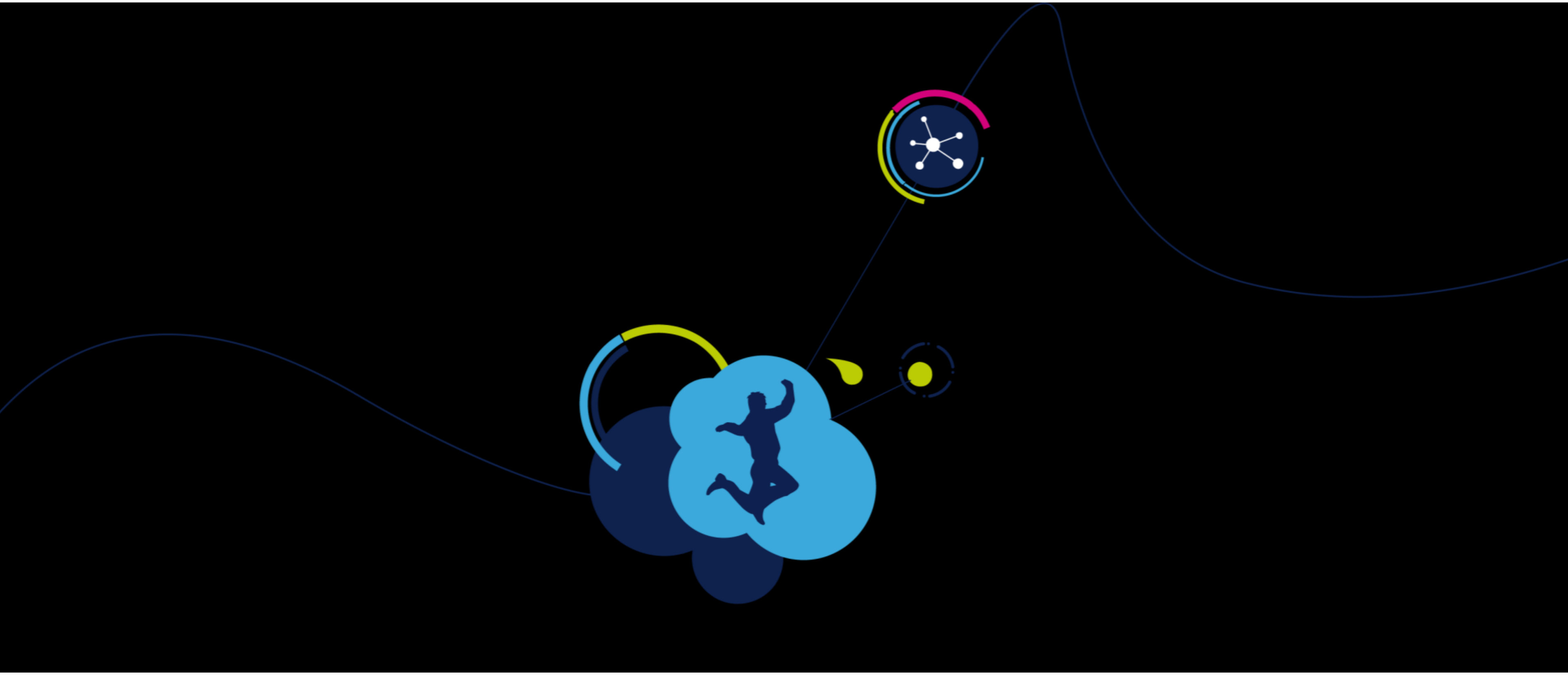
FreeRTOS with CubeMX

T.O.M.A.S – Technically Oriented Microcontroller Application Services
v0.02

life.augmented



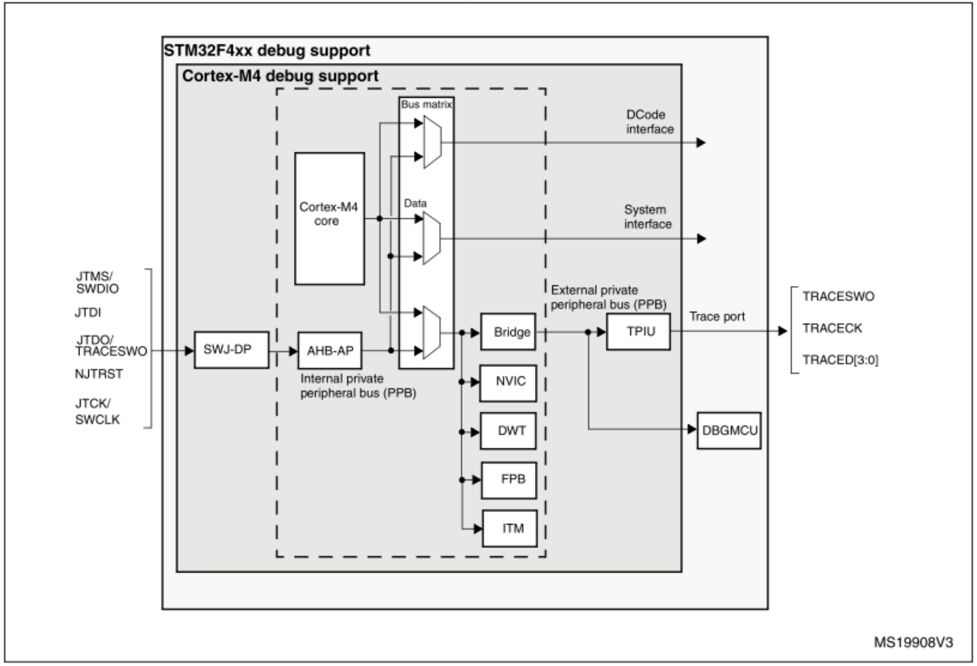
Using SWO to print information from STM32

life.augmented

Using SWO

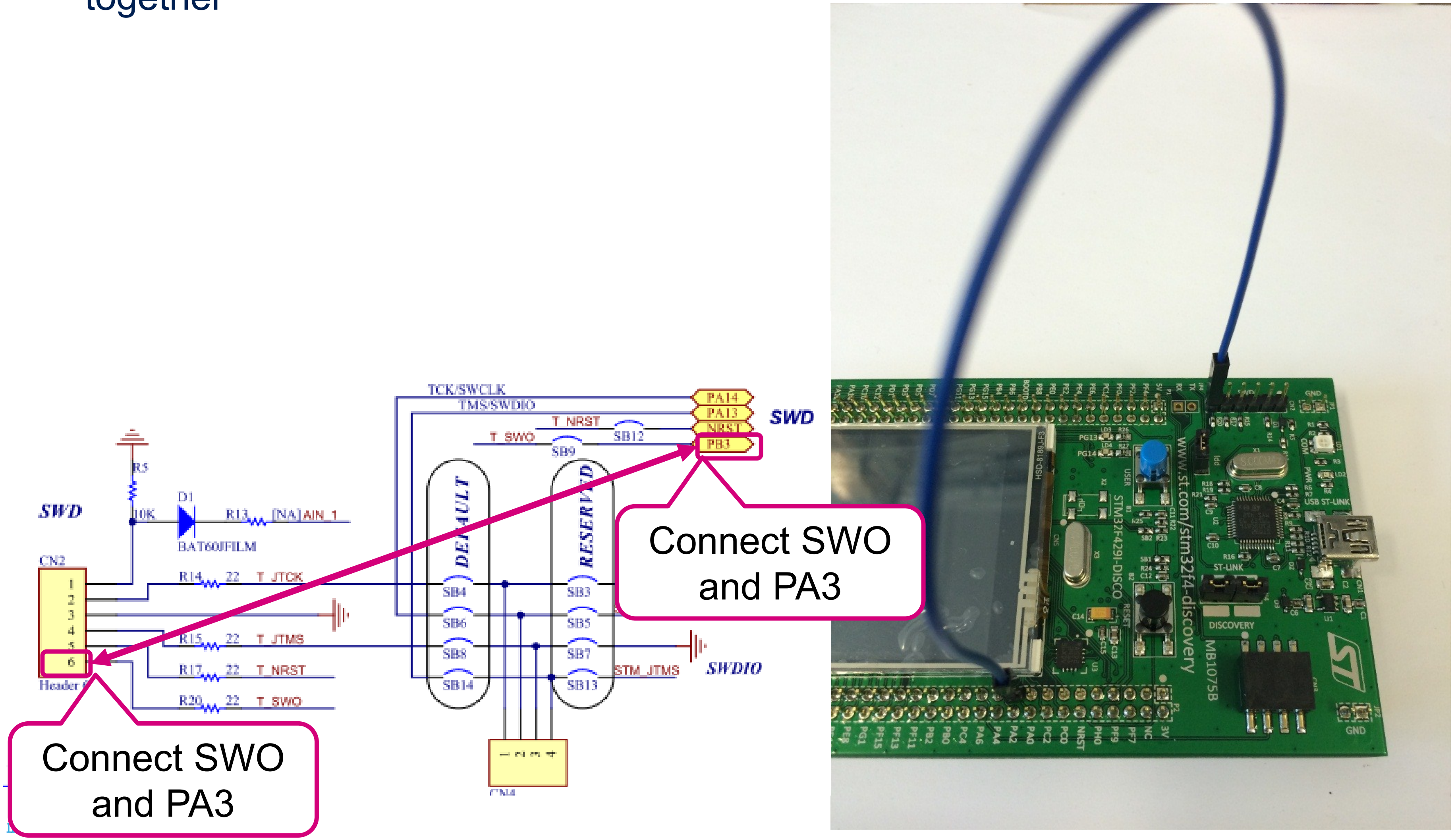
- On some stm32 is periphery called ITM, not mix with ETM(real trace)
- This periphery can be used to internal send data from MCU over SWO pin
- Is possible to redirect the printf into this periphery
- And also some IDEs can display this information during debug
- It is similar to USART but we don’t need any additional wires and PC terminal
- Video: Link

Figure 483. Block diagram of STM32 MCU and Cortex®-M4 with FPU-level debug support



Using SWO

- To make SWO working you musty connect the PA3 and Debugger SWO pin together

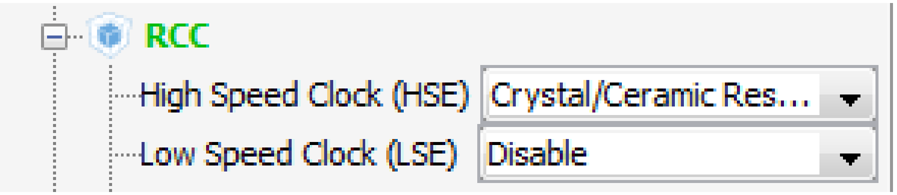


Connect SWO and PA3

Connect SWO and PA3

Using SWO for printf

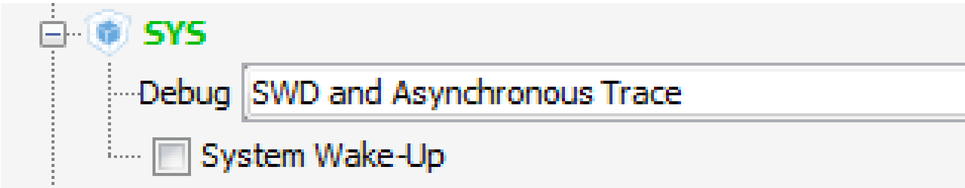
- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- We need only blank project with clock initialization
  - We set the RCC and configure the core to maximum speed and SWD with SWO



RCC

High Speed Clock (HSE) Crystal/Ceramic Res...

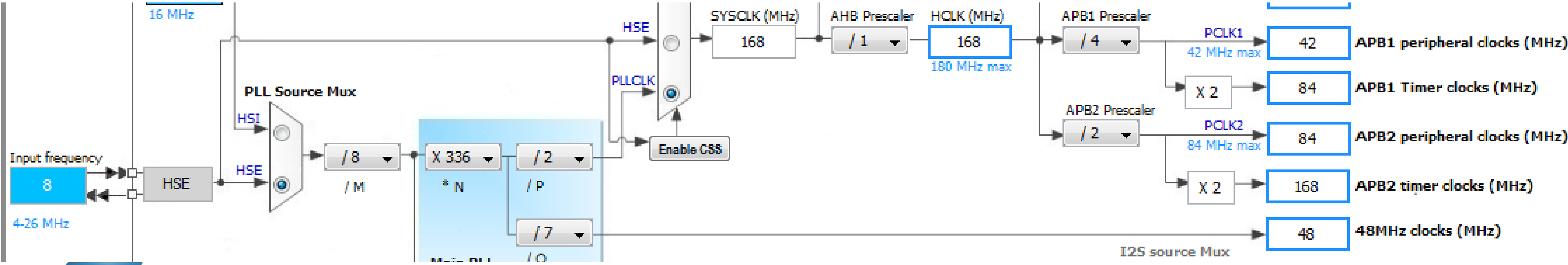
Low Speed Clock (LSE) Disable



SYS

Debug SWD and Asynchronous Trace

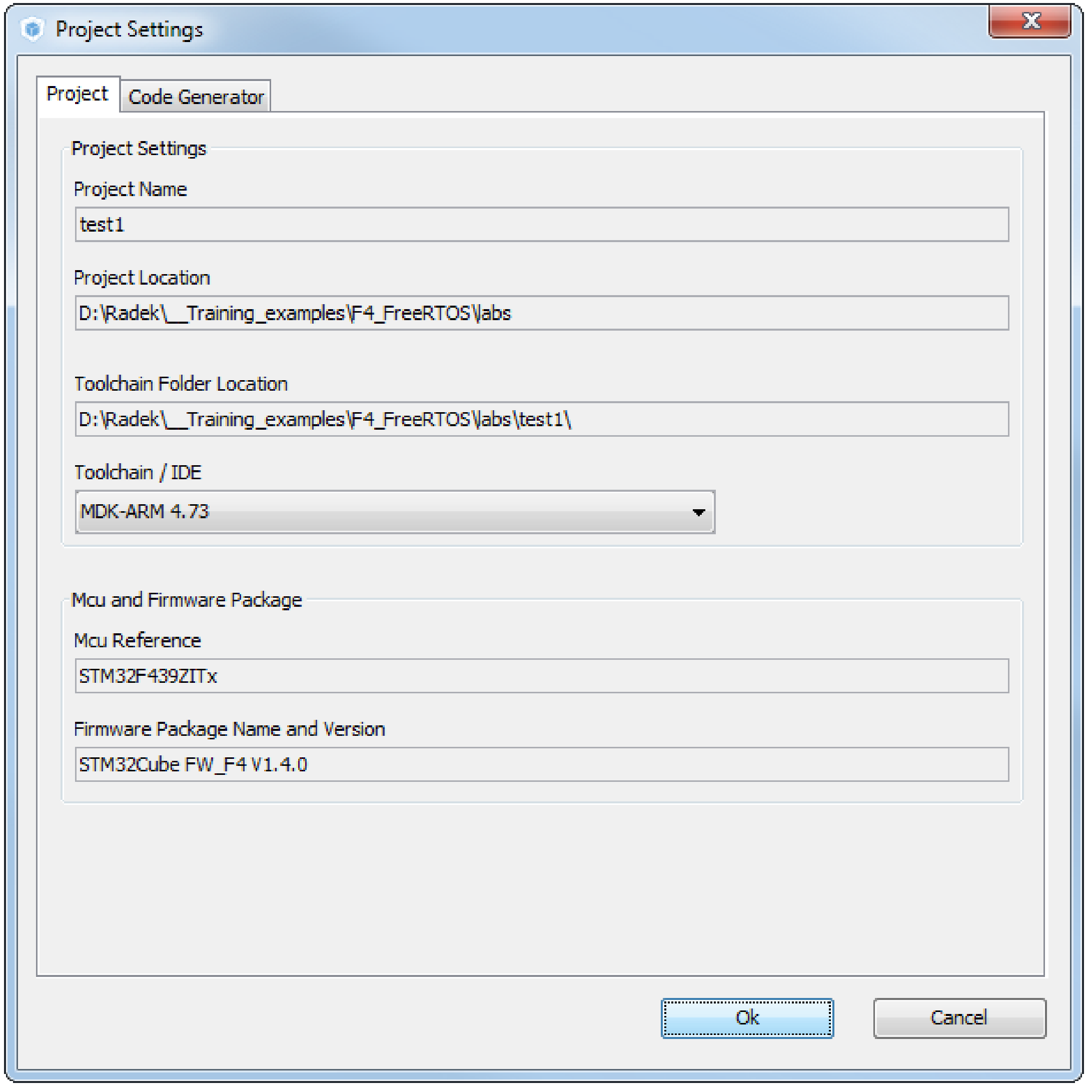
□ System Wake-Up



Using SWO for printf

- Now we set the project details for generation
  - Menu > Project > Project Settings
  - Set the project name
  - Project location
  - Type of toolchain

- Now we can Generate Code
  - Menu > Project > Generate Code



Project Settings

Project Name
test1

Project Location
D:\Radek\__Training_examples\F4_FreeRTOS\labs

Toolchain Folder Location
D:\Radek\__Training_examples\F4_FreeRTOS\labs\test1\

Toolchain / IDE
MDK-ARM 4.73

Mcu and Firmware Package

Mcu Reference
STM32F439ZITx

Firmware Package Name and Version
STM32Cube FW_F4 V1.4.0

Using SWO for printf in KEIL

- We need to include the stdio.h library to make printf working

```
/* USER CODE BEGIN Includes */
#include <stdio.h>
/* USER CODE END Includes */
```

- And define __FILE structure

```
/* USER CODE BEGIN PFP */
struct __FILE { int handle; /* Add whatever is needed */ };
/* USER CODE END PFP */
```

- fputc function must be defined to send byte over ITM

```
/* USER CODE BEGIN 4 */
/*send text over SWV*/
int fputc(int ch, FILE *f) {
ITM_SendChar(ch);//send method for SWV
  return(ch);
}
/* USER CODE END 4 */
```

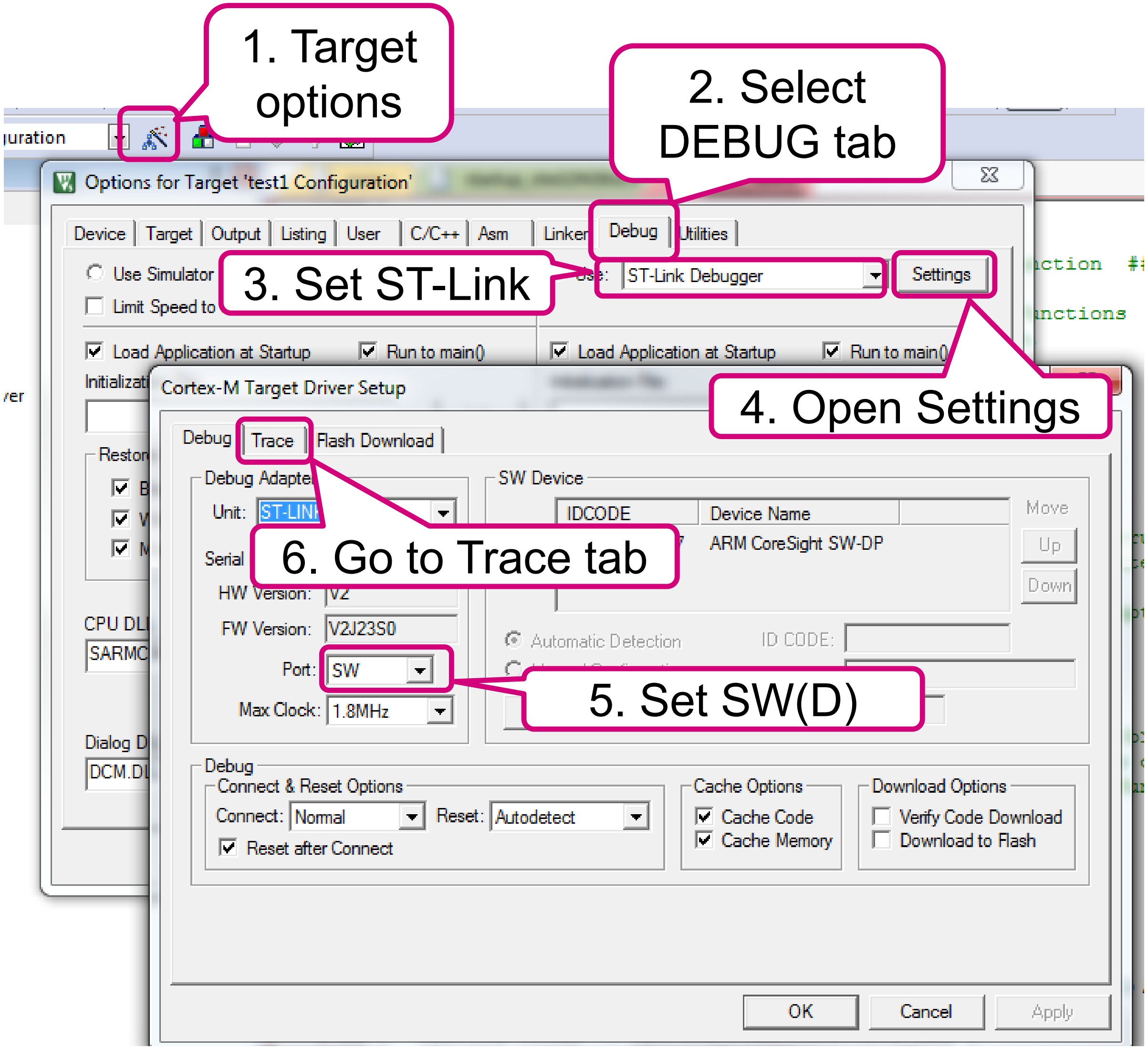
Using SWO for printf in KEIL

- If the MCU run on very high frequency and you not see print f output you may try put into ITM send delay loop

```
/* USER CODE BEGIN 4 */
/*send text over SWV*/
int fputc(int ch, FILE *f) {
  uint32_t i=0;
  for(i=0;i<0xFFFF;i++);//waiting method, lower value will stop the SWV
  ITM_SendChar(ch);//send method for SWV
  return(ch);
}
/* USER CODE END 4 */
```

Using SWO for printf in KEIL

- In KEIL
- Open target options (ALT+F7)
- Set ST-Link debugger
- Button Settings
- Set SW(D) in Debug TAB
- Got to Trace TAB



1. Target options

2. Select DEBUG tab

3. Set ST-Link

4. Open Settings

5. Set SW(D)

6. Go to Trace tab

Options for Target 'test1 Configuration'

○ Use Simulator

□ Limit Speed to

Use: ST-Link Debugger

☑ Load Application at Startup ☑ Run to main()

☑ Load Application at Startup ☑ Run to main()

Cortex-M Target Driver Setup

Debug Adapter

Unit: ST-LINK

Serial

HW Version: V2

FW Version: V2J23S0

Port: SW

Max Clock: 1.8MHz

SW Device

IDCODE	Device Name
	ARM CoreSight SW-DP

● Automatic Detection ID CODE:

Debug

Connect & Reset Options

Connect: Normal Reset: Autodetect

☑ Reset after Connect

Cache Options

☑ Cache Code

☑ Cache Memory

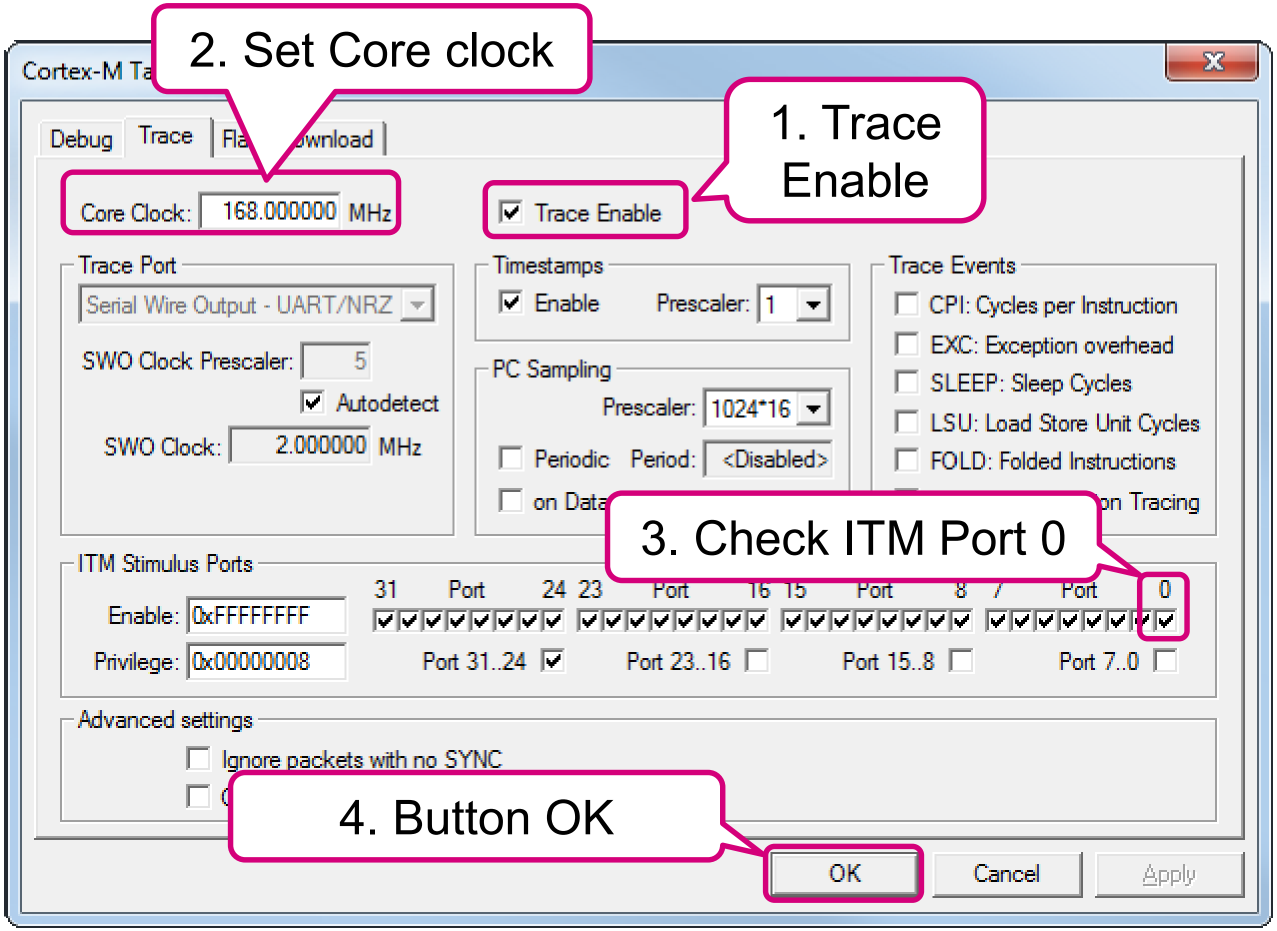
Download Options

□ Verify Code Download

□ Download to Flash

Using SWO for printf in KEIL

- Check Trace Enable
- Set core clock (168MHz must be same as in CubeMX clock tree)
- ITM Stimulus Port 0 must be checked
- Button OK



2. Set Core clock

1. Trace Enable

3. Check ITM Port 0

4. Button OK

Cortex-M Ta

Debug | Trace | Fla... ...wnload

Core Clock: 168.000000 MHz

☑ Trace Enable

Trace Port

Serial Wire Output - UART/NRZ

SWO Clock Prescaler: 5

☑ Autodetect

SWO Clock: 2.000000 MHz

Timestamps

☑ Enable Prescaler: 1

PC Sampling

Prescaler: 1024*16

□ Periodic Period: <Disabled>

□ on Data

Trace Events

□ CPI: Cycles per Instruction

□ EXC: Exception overhead

□ SLEEP: Sleep Cycles

□ LSU: Load Store Unit Cycles

□ FOLD: Folded Instructions

...on Tracing

ITM Stimulus Ports

	31 Port 24	23 Port 16	15 Port 8	7 Port 0
Enable: 0xFFFFFFFF	☑☑☑☑☑☑☑☑	☑☑☑☑☑☑☑☑	☑☑☑☑☑☑☑☑	☑☑☑☑☑☑☑☑
Privilege: 0x00000008	Port 31..24 ☑	Port 23..16 □	Port 15..8 □	Port 7..0 □

Advanced settings

□ Ignore packets with no SYNC

□ C...

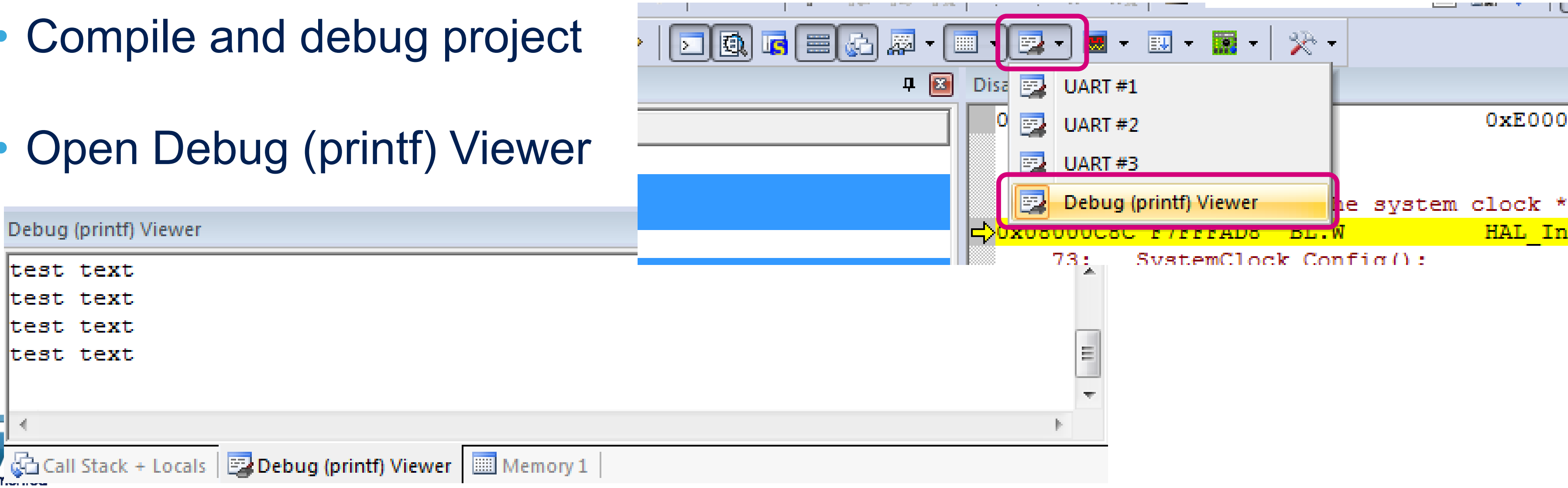
Using SWO for printf in KEIL

- Testing loop printf in main

```
/* USER CODE BEGIN 3 */
/* Infinite loop */
while (1)
{
  printf("test text\n");
  HAL_Delay(1000);
}
/* USER CODE END 3 */
```

- Compile and debug project

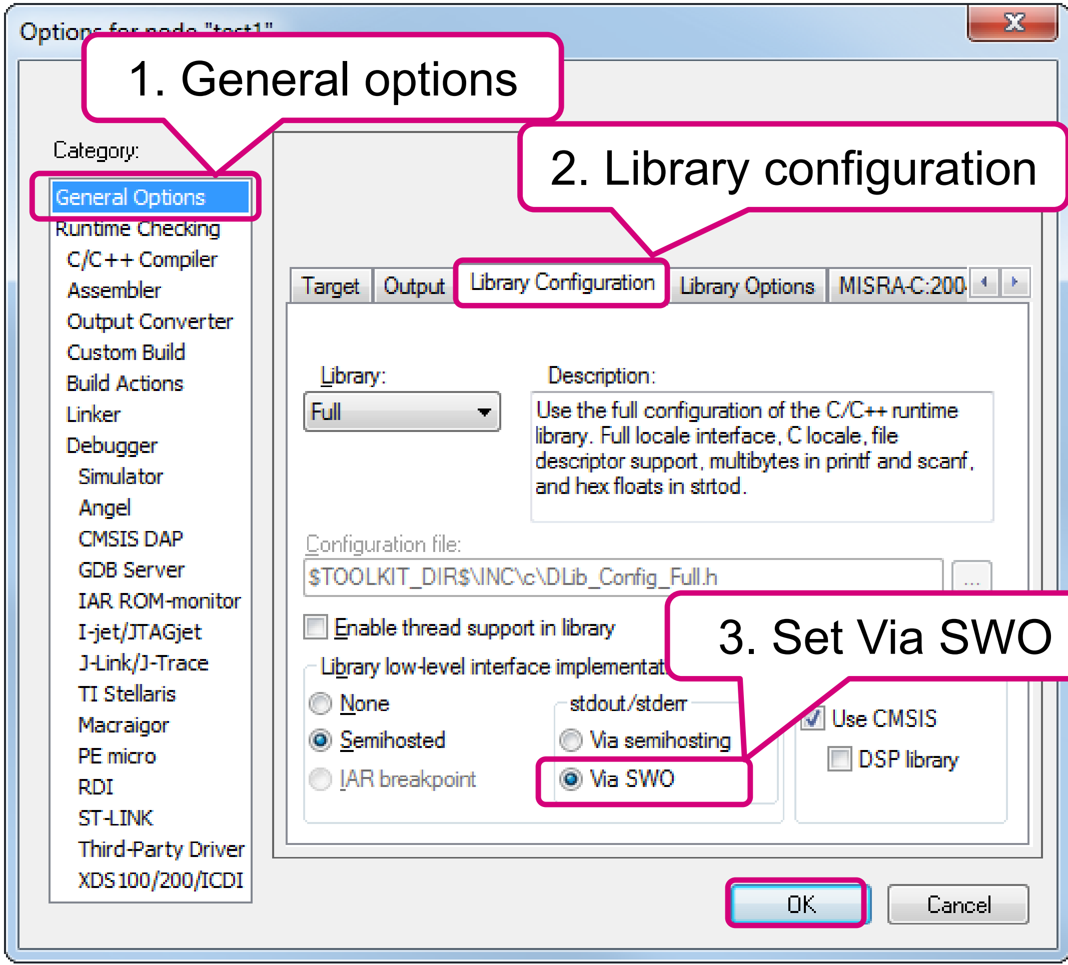
- Open Debug (printf) Viewer



```
test text
test text
test text
test text
```

Using SWO for printf in IAR

- Here is configuration more easier
- Open project options
- General Options
- TAB library and configuration
- Select stdout/stderr via SWO
- Button OK



1. General options

2. Library configuration

3. Set Via SWO

Library: Full

Description: Use the full configuration of the C/C++ runtime library. Full locale interface, C locale, file descriptor support, multibytes in printf and scanf, and hex floats in strtod.

Configuration file: $TOOLKIT_DIR$\INC\c\DLib_Config_Full.h

□ Enable thread support in library

Library low-level interface implementat

○ None
● Semihosted
○ IAR breakpoint

stdout/stderr
○ Via semihosting
● Via SWO

☑ Use CMSIS
□ DSP library

Using SWO for printf in IAR

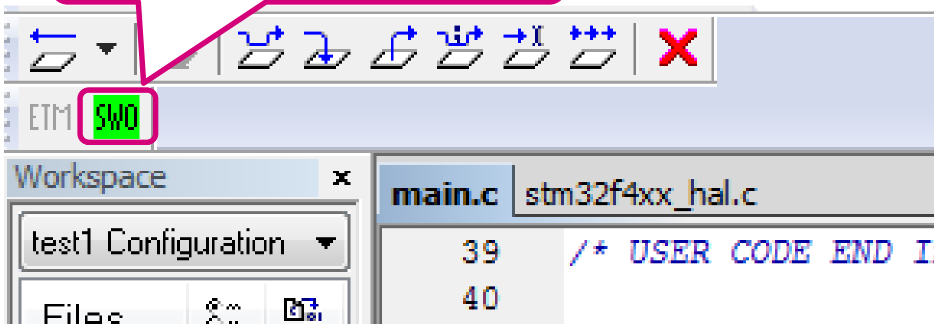
- Add testing printf into loop

```
/* USER CODE BEGIN 3 */
/* Infinite loop */
while (1)
{
  printf("test text\n");
  HAL_Delay(1000);
}
/* USER CODE END 3 */
```

- Compile project and go to debug

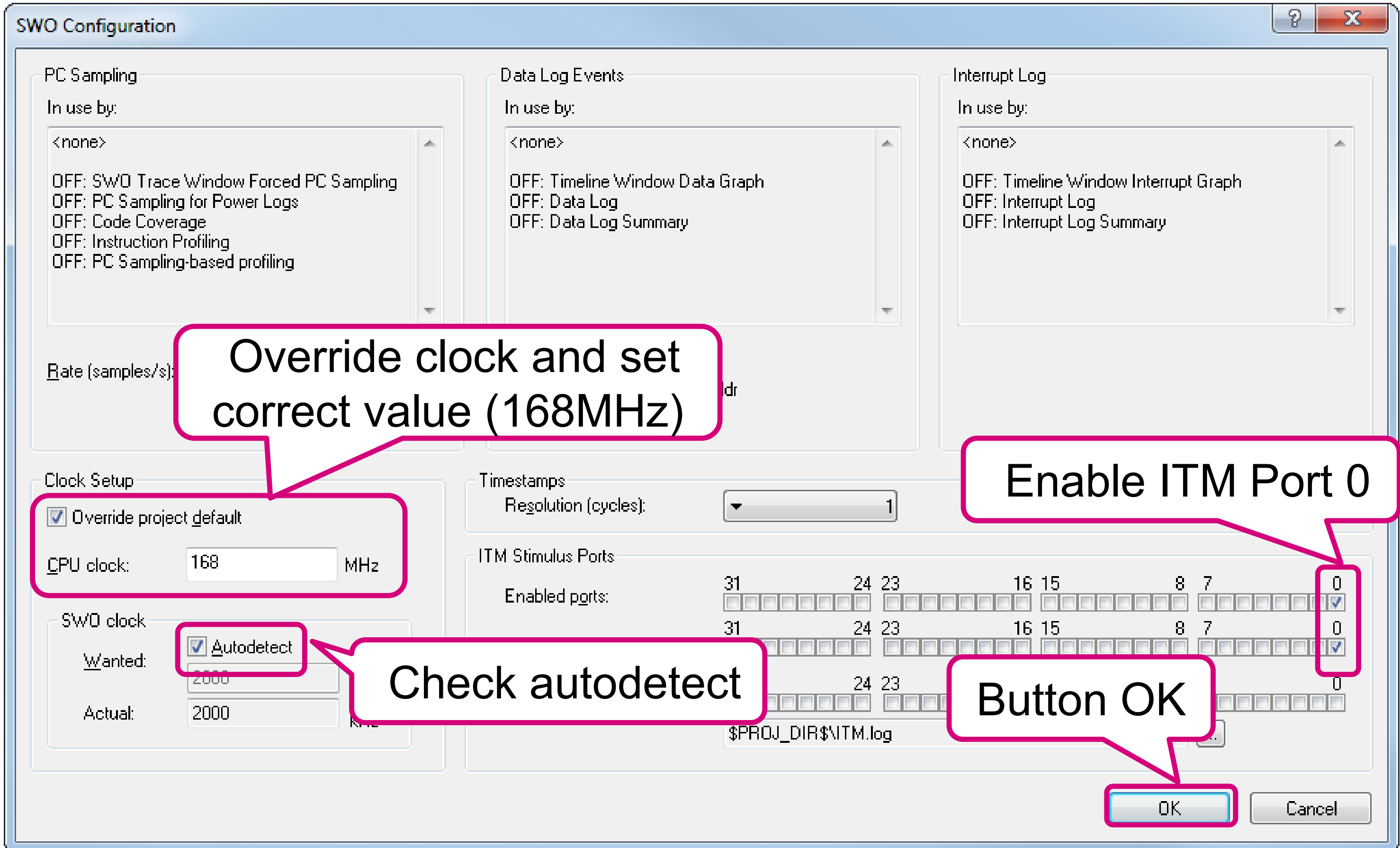
- Open SWO

Open SWO options



Using SWO for printf in IAR

- Override project default if clocks are different from core (168MHz in out case)
- Check autodetect SWO clock if possible
- Enable ITM Port 0 to terminal
- Button OK



SWO Configuration

PC Sampling
In use by:
<none>
OFF: SWO Trace Window Forced PC Sampling
OFF: PC Sampling for Power Logs
OFF: Code Coverage
OFF: Instruction Profiling
OFF: PC Sampling-based profiling

Rate (samples/s):

Data Log Events
In use by:
<none>
OFF: Timeline Window Data Graph
OFF: Data Log
OFF: Data Log Summary

Interrupt Log
In use by:
<none>
OFF: Timeline Window Interrupt Graph
OFF: Interrupt Log
OFF: Interrupt Log Summary

Clock Setup
☑ Override project default
CPU clock: 168 MHz

SWO clock
☑ Autodetect
Wanted: 2000
Actual: 2000

Timestamps
Resolution (cycles): 1

ITM Stimulus Ports
Enabled ports: 31 24 23 16 15 8 7 0
□□□□□□□□ □□□□□□□□ □□□□□□□□ □□□□□□□☑
31 24 23 16 15 8 7 0
□□□□□□□□ □□□□□□□□ □□□□□□□□ □□□□□□□☑
24 23 0

$PROJ_DIR$\ITM.log

Override clock and set correct value (168MHz)

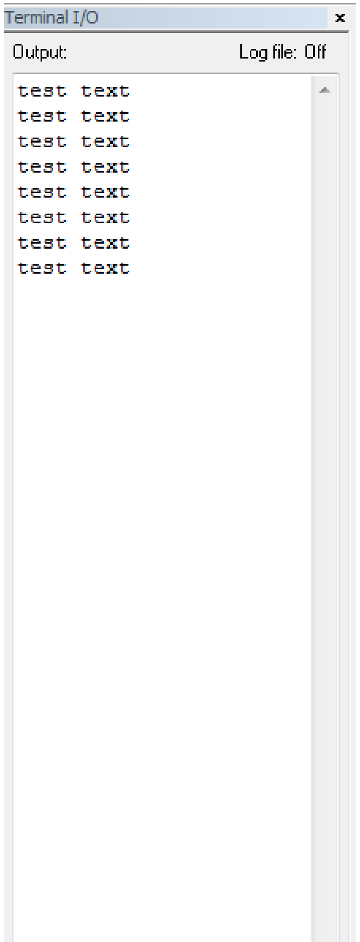
Enable ITM Port 0

Check autodetect

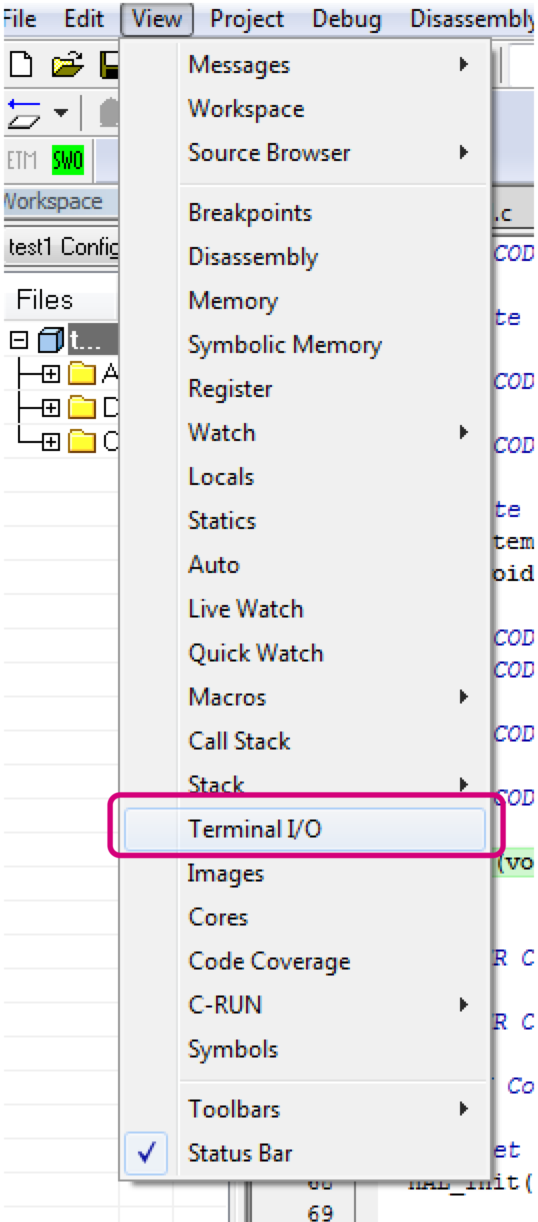
Button OK

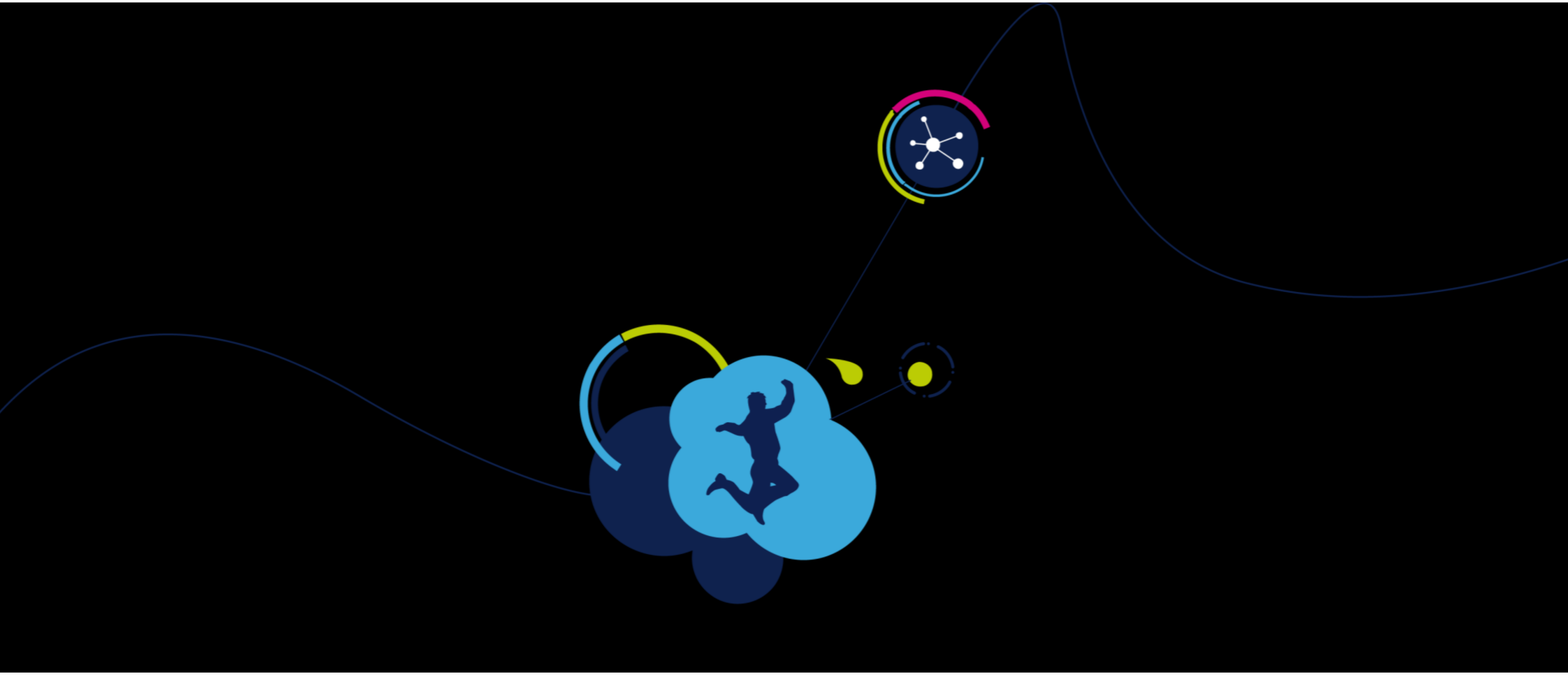
Using SWO for printf in IAR

- Menu View -> Terminal I/O
- Run program



```
test text
test text
test text
test text
test text
test text
test text
test text
```





FreeRTOS

FreeRTOS

About FreeRTOS

- Market leading RTOS by Real Time Engineers Ltd.
- Professionally developed with strict quality management
- Commercial versions available: OpenRTOS and SafeRTOS
- Documentation available on www.freertos.org
- Free support through forum (moderated by RTOS original author Richard Barry)

FreeRTOS
Main features

- Preemptive or cooperative real-time kernel
- Scalable RTOS with tiny footprint (less than 10KB ROM)
- Includes a tickless mode for low power applications
- Synchronization and inter-task communications using
  - message queues
  - binary and counting semaphores
  - mutexes
  - group events (flags)
- Software timers for tasks scheduling
- Execution trace functionality
- CMSIS-RTOS API port

FreeRTOS
APIs overview (1/2)

API category	FreeRTOS API	Description
Task creation	xTaskCreate	Creates a new task
	vTaskDelete	Deletes a task
Task control	vTaskDelay	Task delay
	vTaskPrioritySet	Sets task priority
	vTaskSuspend	Suspends a task
	vTaskResume	Resumes a task
Kernel control	vTaskStartScheduler	Starts kernel scheduler
	vTaskSuspendAll	Suspends all tasks
	xTaskResumeAll	Resumes all tasks
	taskYIELD	Forces a context switch
	taskENTER_CRITICAL	Enter a critical section (stops context switching)
	taskEXIT_CRITICAL	Exits from a critical section

FreeRTOS APIs overview (2/2)

API category	FreeRTOS API	Description
Message queues	xQueueCreate	Creates a queue
	xQueueSend	Sends data to queue
	xQueueReceive	Receive data from the queue
Semaphores	xSemaphoreCreateBinary	Creates a binary semaphore
	xSemaphoreCreateCounting	Creates a counting semaphore
	xSemaphoreCreateMutex	Creates a mutex semaphore
	xSemaphoreTake	Semaphore take
	xSemaphoreGive	Semaphore give
Timers	xTimerCreate	Creates a timer
	xTimerStart	Starts a timer
	xTimerStop	Stops a timer

FreeRTOS
CMSIS-RTOS FreeRTOS implementation

- Implementation in file cmsis-os.c (found in folder: “\Middlewares\Third_Party\FreeRTOS\Source\CMSIS_RTOS”)

- The following table lists examples of the CMSIS-RTOS APIs and the FreeRTOS APIs used to implement them

API category	CMSIS_RTOS API	FreeRTOS API
Kernel control	osKernelStart	vTaskStartScheduler
Thread management	osThreadCreate	xTaskCreate
Semaphore	osSemaphoreCreate	vSemaphoreCreateBinary xSemaphoreCreateCounting
Mutex	osMutexWait	xSemaphoreTake
Message queue	osMessagePut	xQueueSend xQueueSendFromISR
Timer	osTimerCreate	xTimerCreate

- Note: CMSIS-RTOS implements same model as FreeRTOS for task states

FreeRTOS
CMSIS-RTOS API

- CMSIS-RTOS API is a generic RTOS interface for Cortex-M processor based devices

- Middleware components using the CMSIS-RTOS API are RTOS agnostic, this allows an easy linking to any third-party RTOS

- The CMSIS-RTOS API defines a minimum feature set including
  - Thread Management
  - Kernel control
  - Semaphore management
  - Message queue and mail queue
  - Memory management
  - …

- For detailed documentation regarding CMSIS-RTOS refer to: http://www.keil.com/pack/doc/CMSIS/RTOS/html/index.html

FreeRTOS
Configuration options

- Configuration options are declared in file FreeRTOSConfig.h

- Important configuration options are:

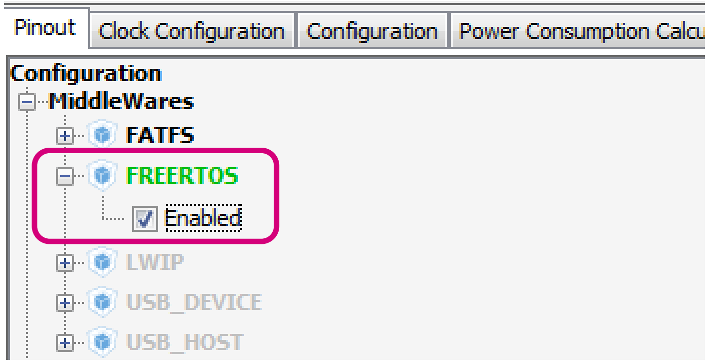
Config option	Description
configUSE_PREEMPTION	Enables Preemption
configCPU_CLOCK_HZ	CPU clock frequency in hertz
configTICK_RATE_HZ	Tick rate in hertz
configMAX_PRIORITIES	Maximum task priority
configTOTAL_HEAP_SIZE	Total heap size for dynamic allocation
configLIBRARY_LOWEST_INTERRUPT_PRIORITY	Lowest interrupt priority (0xF when using 4 cortex preemption bits)
configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY	Highest thread safe interrupt priority (higher priorities are lower numeric value)

FreeRTOS
Tickless idle mode operation

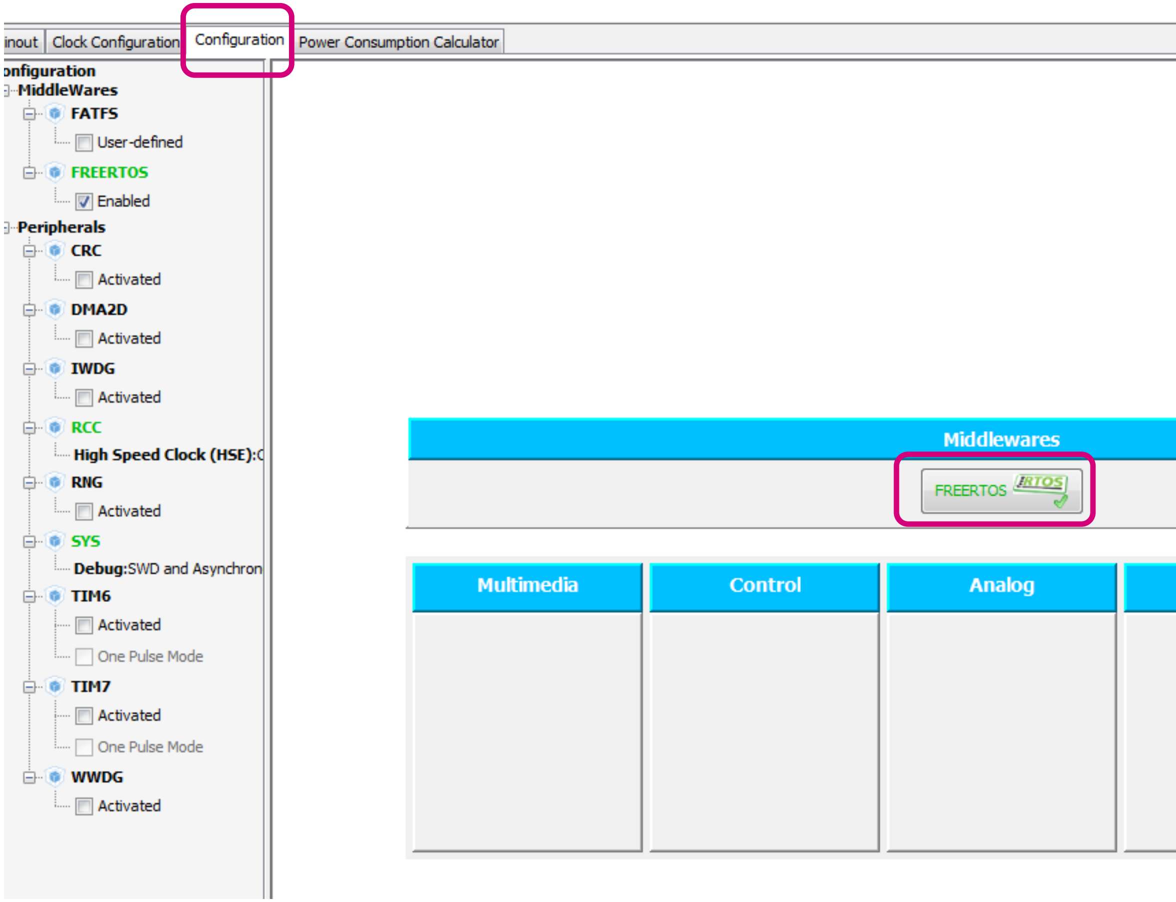
- Kernel can stop system tick interrupt and place MCU in low power mode, on exit from this mode systick counter is updated
- Enabled when setting configUSE_TICKLESS_IDLE as 1
- The kernel will call a macro portSUPPRESS_TICKS_AND_SLEEP() when the Idle task is the only task able to run (and no other task is scheduled to exit from blocked state after n ticks)
  - n value is defined in FreeRTOSconf.h file
- FreeRTOS implementation of portSUPRESS_TICKS_AND_SLEEP for cortexM3/M4 enters MCU in sleep low power mode
- Wakeup from sleep mode can be from a system interrupt/event

FreeRTOS in CubeMX

- Use CubeMX project from printf example
- In Pinout TAB select in MiddleWares FreeRTOS
- In Configuration TAB is now possible to configure FreeRTOS Parameters



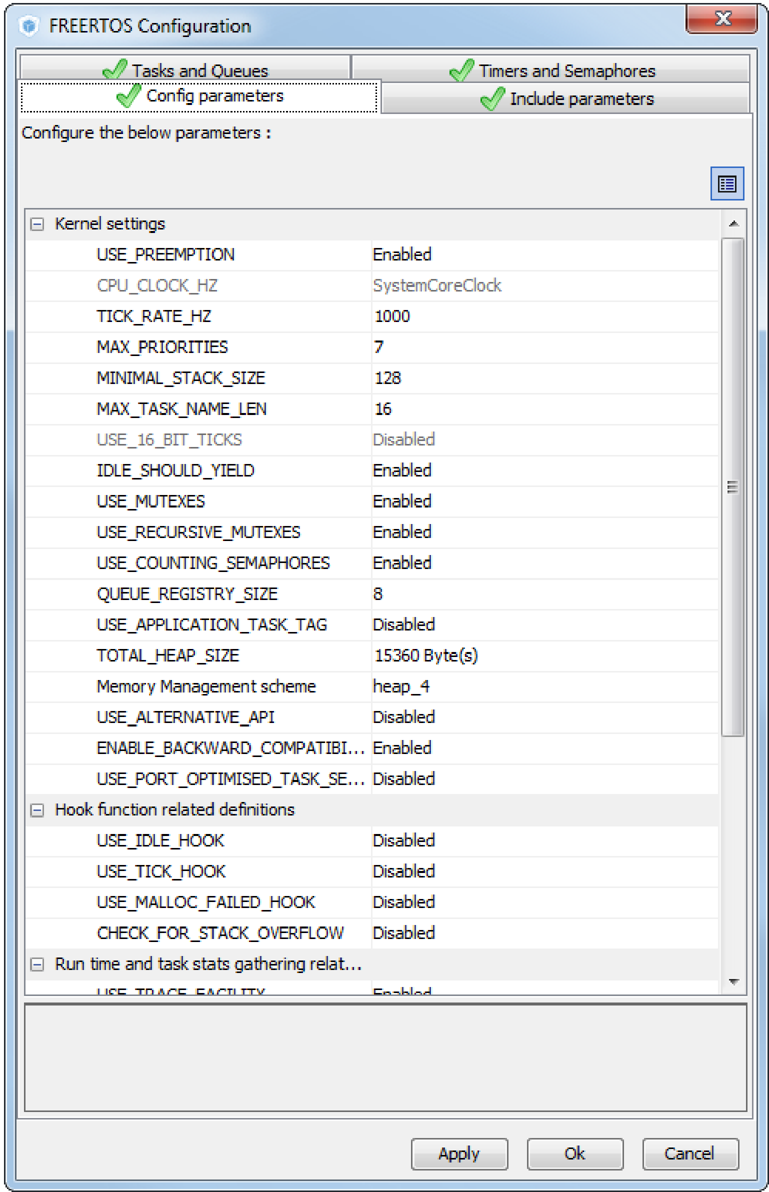
Configuration
- MiddleWares
  - FATFS
  - FREERTOS
    - ☑ Enabled
  - LWIP
  - USB_DEVICE
  - USB_HOST



onfiguration
- MiddleWares
  - FATFS
    - □ User-defined
  - FREERTOS
    - ☑ Enabled
- Peripherals
  - CRC
    - □ Activated
  - DMA2D
    - □ Activated
  - IWDG
    - □ Activated
  - RCC
    - High Speed Clock (HSE):C
  - RNG
    - □ Activated
  - SYS
    - Debug:SWD and Asynchron
  - TIM6
    - □ Activated
    - □ One Pulse Mode
  - TIM7
    - □ Activated
    - □ One Pulse Mode
  - WWDG
    - □ Activated

CubeMX FreeRTOS Configuration

- FreeRTOS configuration supported by CubeMX
- Config parameters
  - Set kernel
  - Mem setup
- Include parameters
  - Include some additional functions, not necessary for FreeRTOS run
- Tasks and Queues
  - We can easily create task or queue by CubeMX
- Timers and semaphores
  - CubeMX create semaphore and timer for us

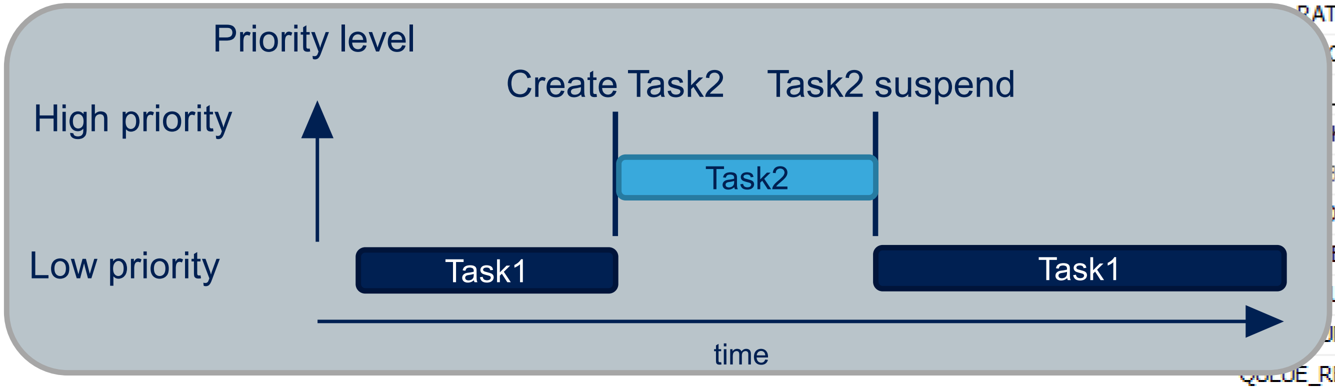


Configure the below parameters :

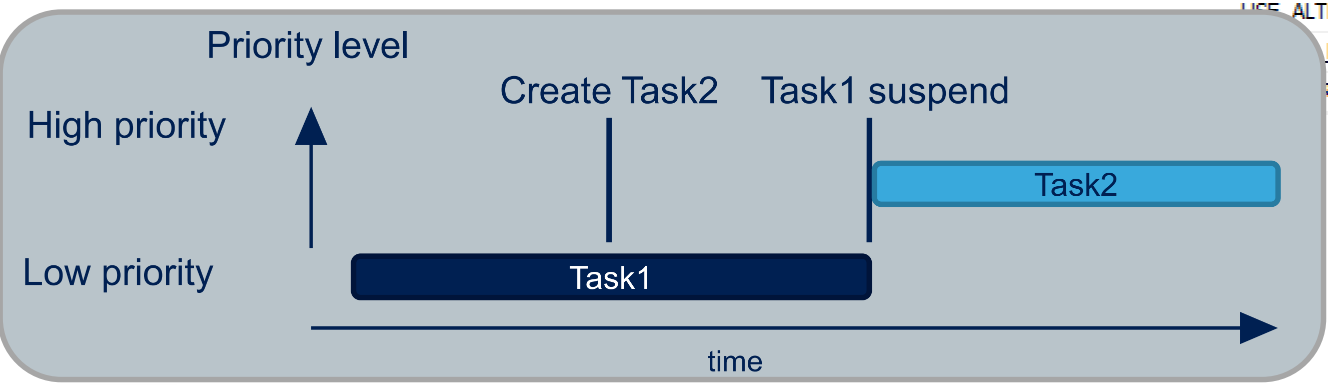
⊟ Kernel settings	
USE_PREEMPTION	Enabled
CPU_CLOCK_HZ	SystemCoreClock
TICK_RATE_HZ	1000
MAX_PRIORITIES	7
MINIMAL_STACK_SIZE	128
MAX_TASK_NAME_LEN	16
USE_16_BIT_TICKS	Disabled
IDLE_SHOULD_YIELD	Enabled
USE_MUTEXES	Enabled
USE_RECURSIVE_MUTEXES	Enabled
USE_COUNTING_SEMAPHORES	Enabled
QUEUE_REGISTRY_SIZE	8
USE_APPLICATION_TASK_TAG	Disabled
TOTAL_HEAP_SIZE	15360 Byte(s)
Memory Management scheme	heap_4
USE_ALTERNATIVE_API	Disabled
ENABLE_BACKWARD_COMPATIBI…	Enabled
USE_PORT_OPTIMISED_TASK_SE…	Disabled
⊟ Hook function related definitions	
USE_IDLE_HOOK	Disabled
USE_TICK_HOOK	Disabled
USE_MALLOC_FAILED_HOOK	Disabled
CHECK_FOR_STACK_OVERFLOW	Disabled
⊟ Run time and task stats gathering relat…	
USE_TRACE_FACILITY	Enabled

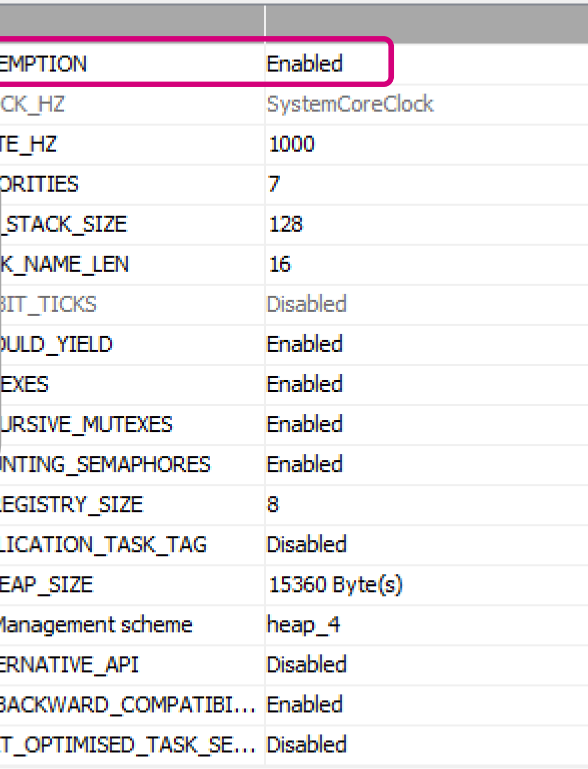
Kernel settings

- Use preemption
  - If enabled use pre-emptive scheduling

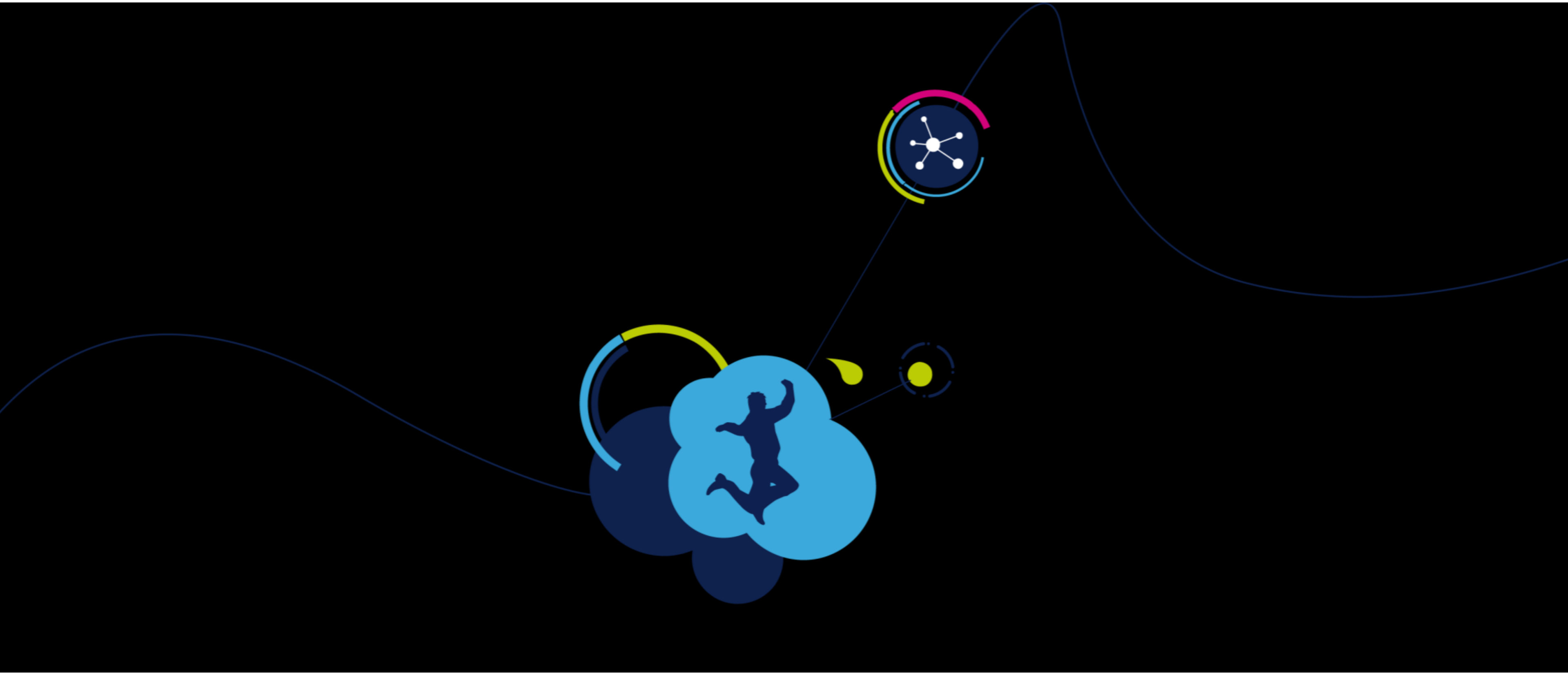


  - If disabled use co-operative scheduling





Kernel settings	
USE_PREEMPTION	Enabled
CPU_CLOCK_HZ	SystemCoreClock
RATE_HZ	1000
ORITIES	7
_STACK_SIZE	128
K_NAME_LEN	16
BIT_TICKS	Disabled
OULD_YIELD	Enabled
EXES	Enabled
URSIVE_MUTEXES	Enabled
UNTING_SEMAPHORES	Enabled
QUEUE_REGISTRY_SIZE	8
USE_APPLICATION_TASK_TAG	Disabled
TOTAL_HEAP_SIZE	15360 Byte(s)
Memory Management scheme	heap_4
USE_ALTERNATIVE_API	Disabled
_BACKWARD_COMPATIBI...	Enabled
RT_OPTIMISED_TASK_SE...	Disabled

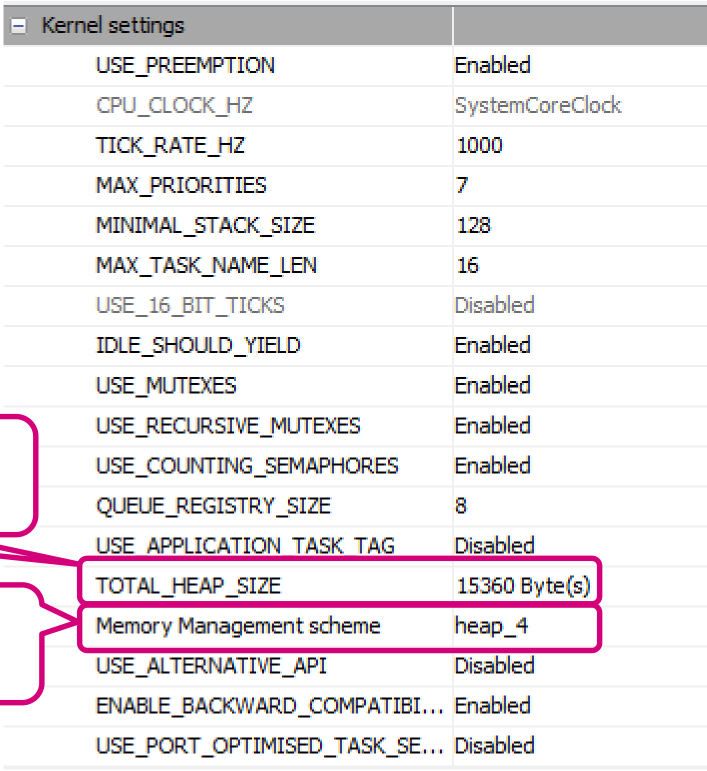


FreeRTOS
Memory allocations types

FreeRTOS
Dynamic memory management

- FreeRTOS have own heap which is use for components
  - Tasks
  - Queues
  - Semaphores
  - Mutexes
  - Dynamic memory allocation

- Is possible to select type of memory allocation



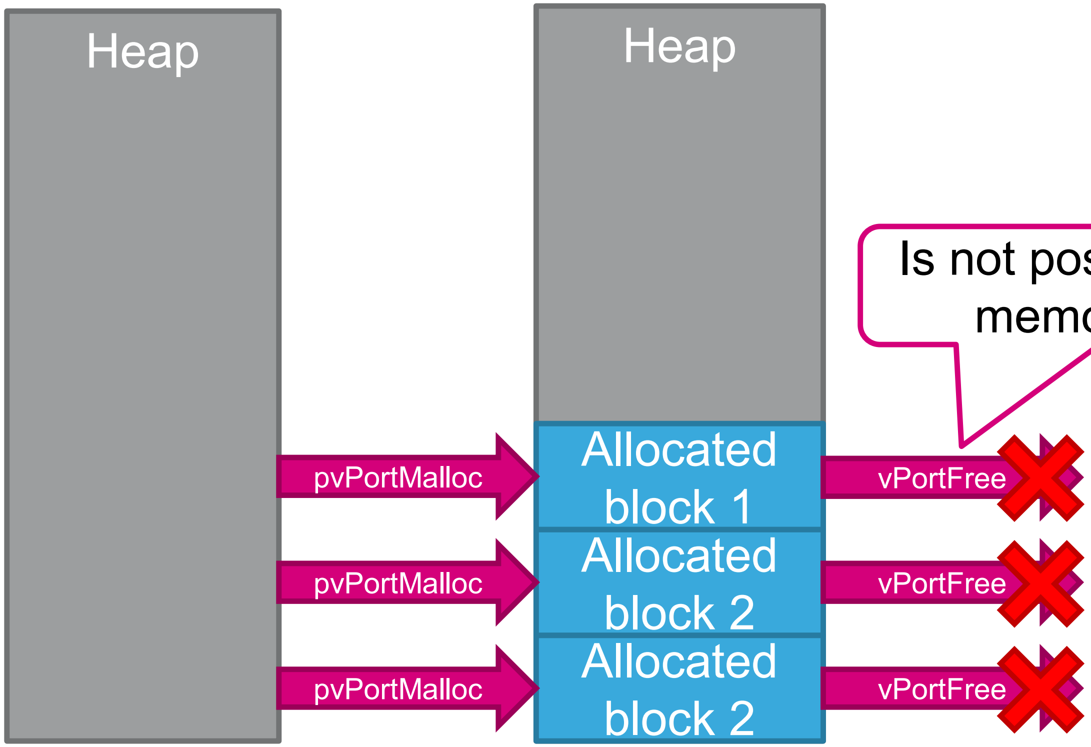
⊟ Kernel settings	
USE_PREEMPTION	Enabled
CPU_CLOCK_HZ	SystemCoreClock
TICK_RATE_HZ	1000
MAX_PRIORITIES	7
MINIMAL_STACK_SIZE	128
MAX_TASK_NAME_LEN	16
USE_16_BIT_TICKS	Disabled
IDLE_SHOULD_YIELD	Enabled
USE_MUTEXES	Enabled
USE_RECURSIVE_MUTEXES	Enabled
USE_COUNTING_SEMAPHORES	Enabled
QUEUE_REGISTRY_SIZE	8
USE_APPLICATION_TASK_TAG	Disabled
TOTAL_HEAP_SIZE	15360 Byte(s)
Memory Management scheme	heap_4
USE_ALTERNATIVE_API	Disabled
ENABLE_BACKWARD_COMPATIBI…	Enabled
USE_PORT_OPTIMISED_TASK_SE…	Disabled

Total heap size for FreeRTOS

How is memory allocated and dealocated

FreeRTOS
Dynamic memory management

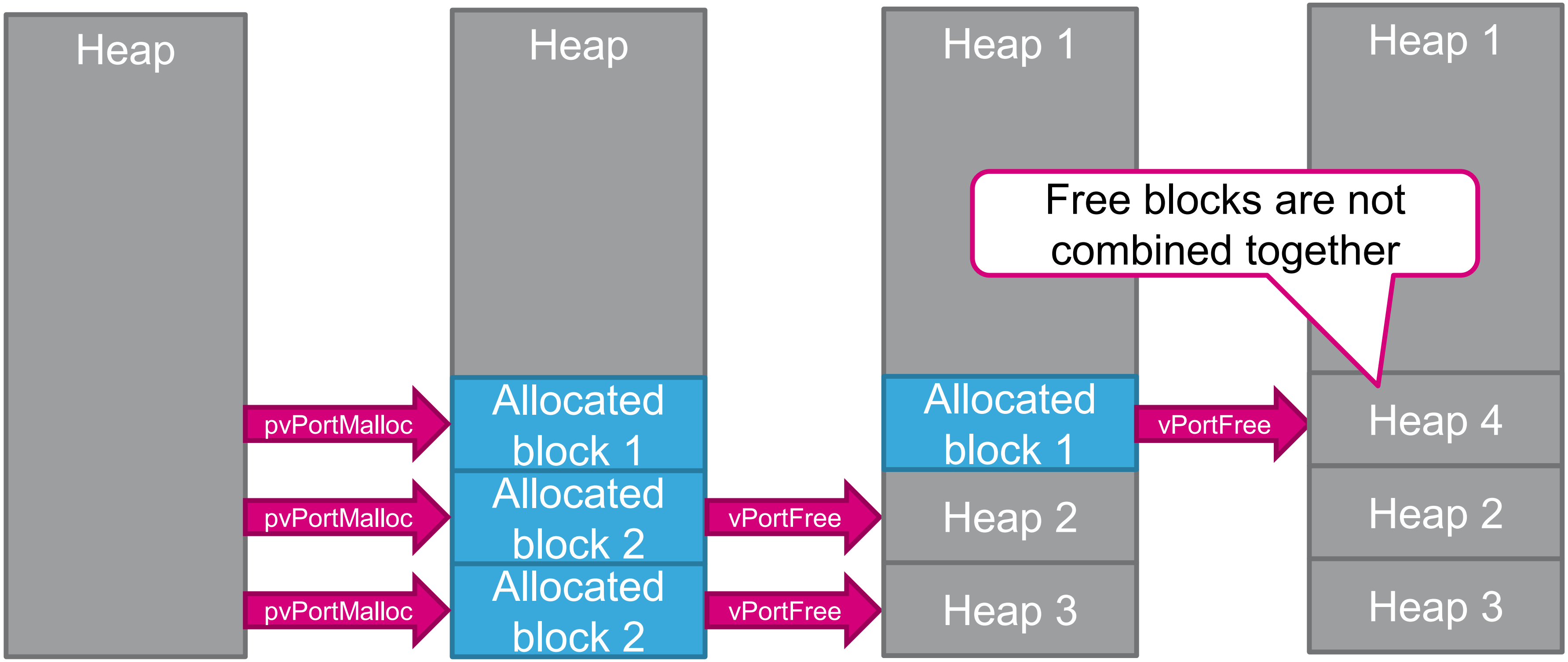
- Heap_1.c
  - Simplest allocation method (deterministic), but does not allow freeing of allocated memory => could be interesting when no memory freeing is necessary



Is not possible to return memory to heap

FreeRTOS
Dynamic memory management

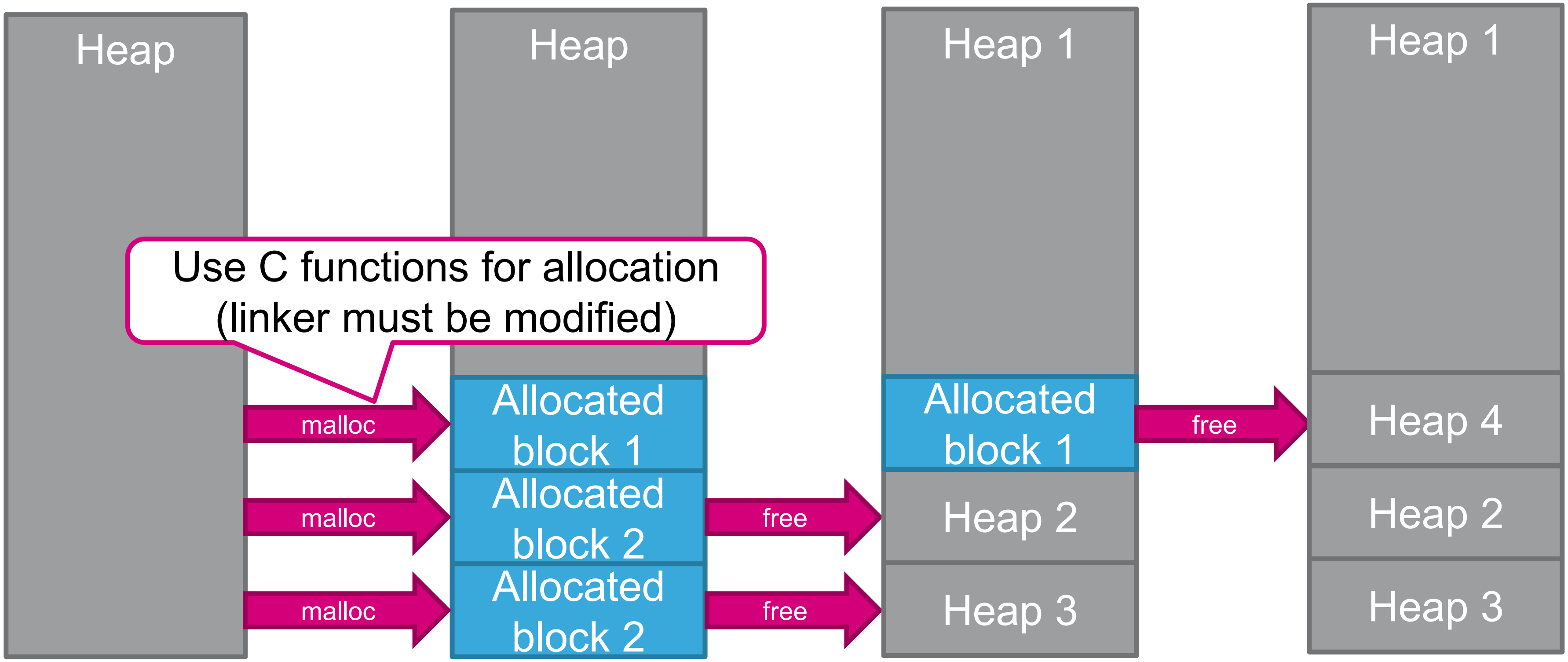
- Heap_2.c
  - Implements a best fit algorithm for allocation
  - Allows memory free operation but does not combine adjacent free blocks => risk of fragmentation



Free blocks are not combined together

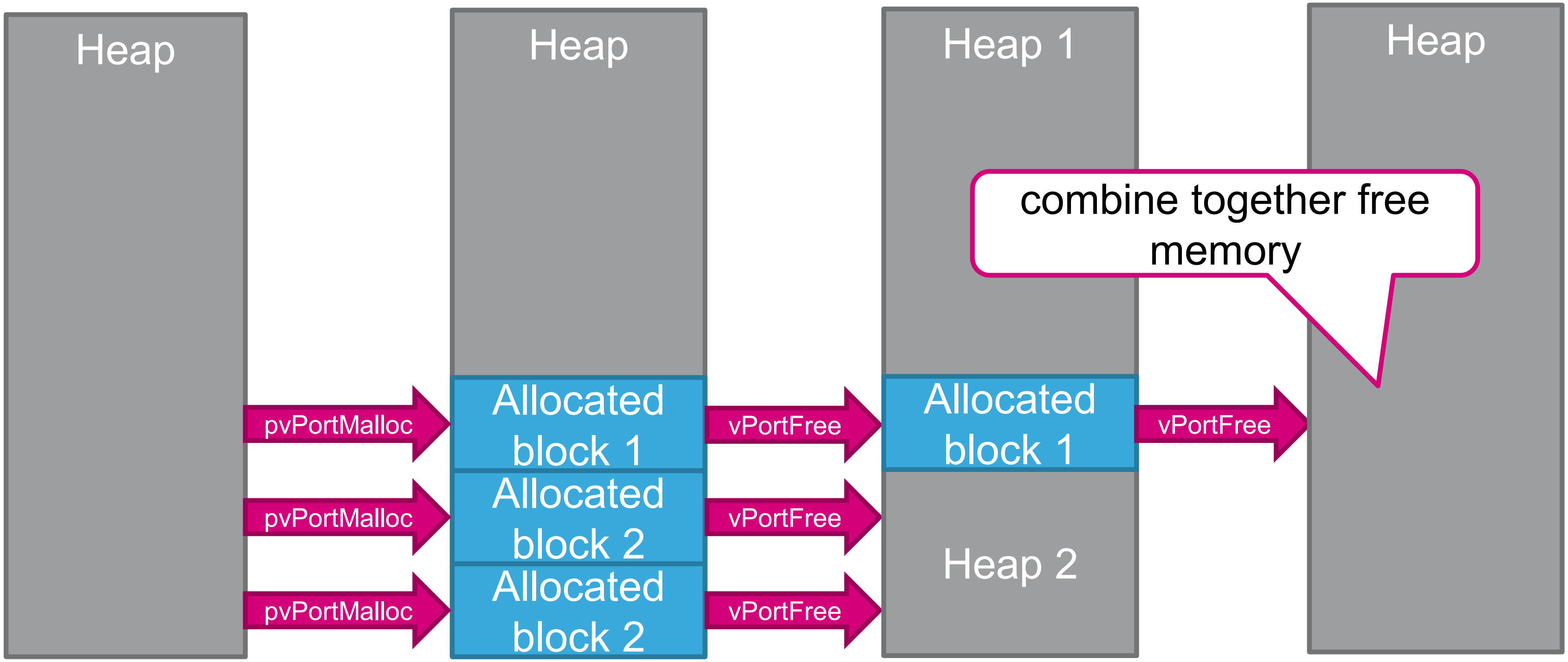
FreeRTOS
Dynamic memory management

- Heap_3.c
  - Implements a simple wrapper for the standard C library malloc() and free(), the wrapper makes these functions thread safe, but makes code increase and not deterministic



FreeRTOS
Dynamic memory management

- Heap_4.c
  - First fit algorithm and able to combine adjacent free memory blocks into a single block => this model is used in STM32Cube examples



Memory allocation

- Use heap_4.c

- Memory Handler definition

```
/* Private variables ---------------------------------------------------------*/
osThreadId Task1Handle;
osPoolId PoolHandle;
```

- Memory allocation

```
void StartTask1(void const * argument)
{
  /* USER CODE BEGIN 5 */
  osPoolDef(Memory,0x100,uint8_t);
  PoolHandle = osPoolCreate(osPool(Memory));
  uint8_t* buffer=osPoolAlloc(PoolHandle);
  /* Infinite loop */
  for(;;)
  {
    osDelay(5000);
  }
  /* USER CODE END 5 */
}
```

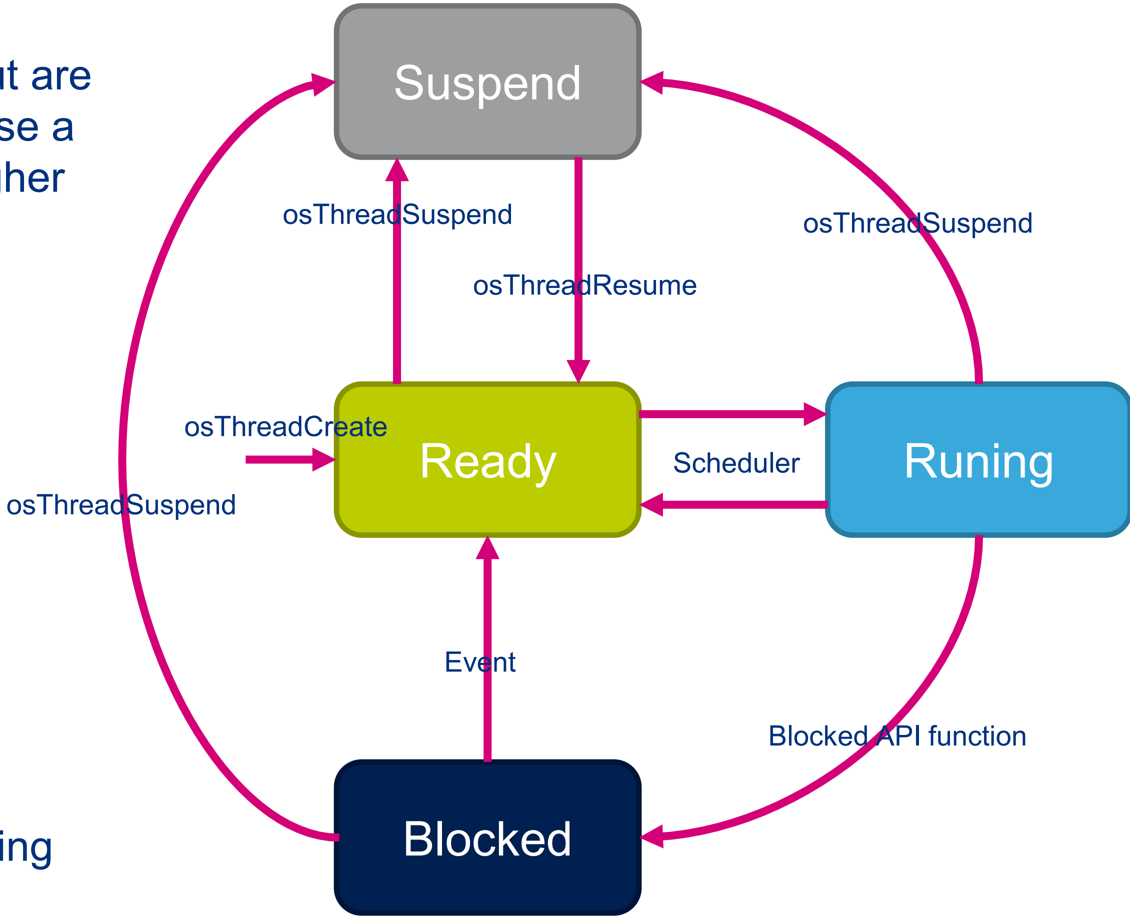
Create memory pool

Allocate memory from pool

FreeRTOS Tasks

FreeRTOS Task states

- **Ready**
  - Tasks are ready to execute but are not currently executing because a different task with equal or higher priority is running
- **Running**
  - when task is actually running
- **Blocked**
  - Task is waiting for a either a temporal or external event
- **Suspended**
  - Task not available for scheduling



FreeRTOS
Task switch

- Task switching on STM32?

- Cortex cores have implemented few features which directly support os systems

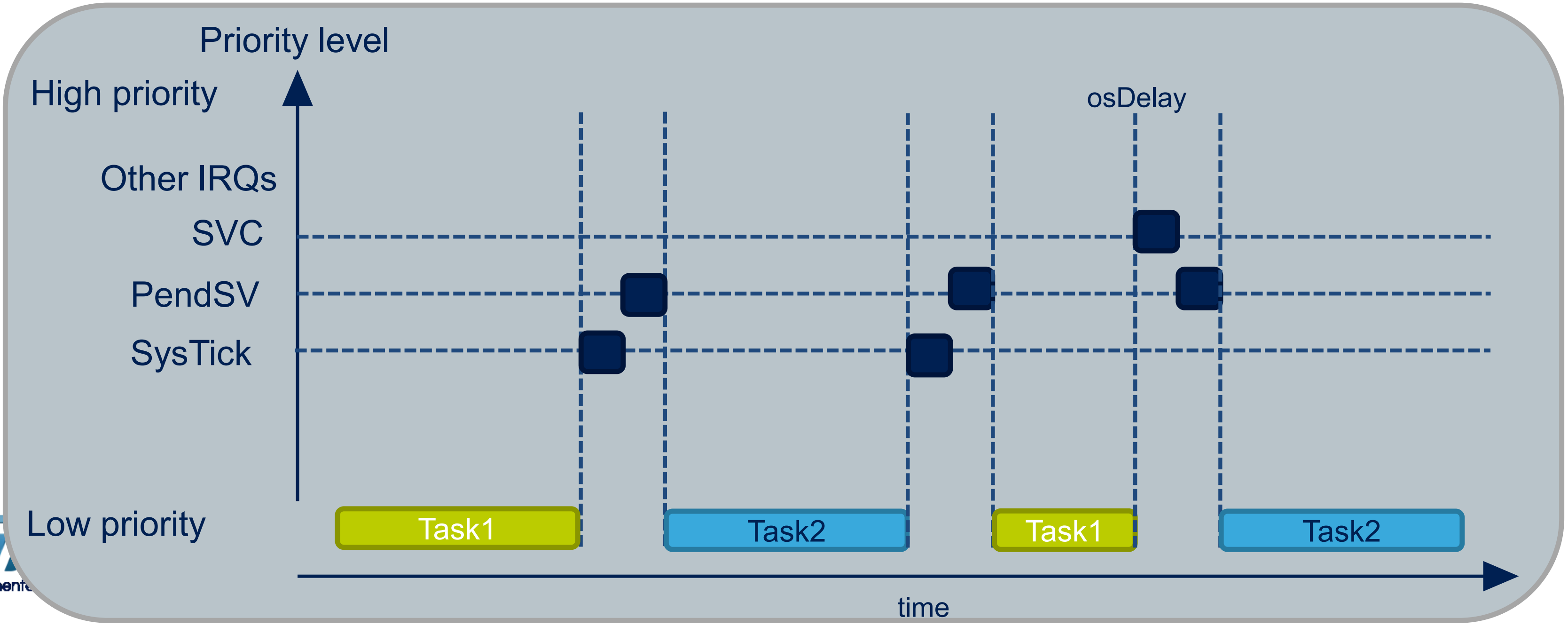
- Two interrupts dedicated for os
  - PendSV interrupt
  - SVC interrupt

- Two stack pointers
  - Process stack pointer
  - Main stack pointer

- SysTick timer
  - Used to periodically trigger scheduling

FreeRTOS OS interrupts

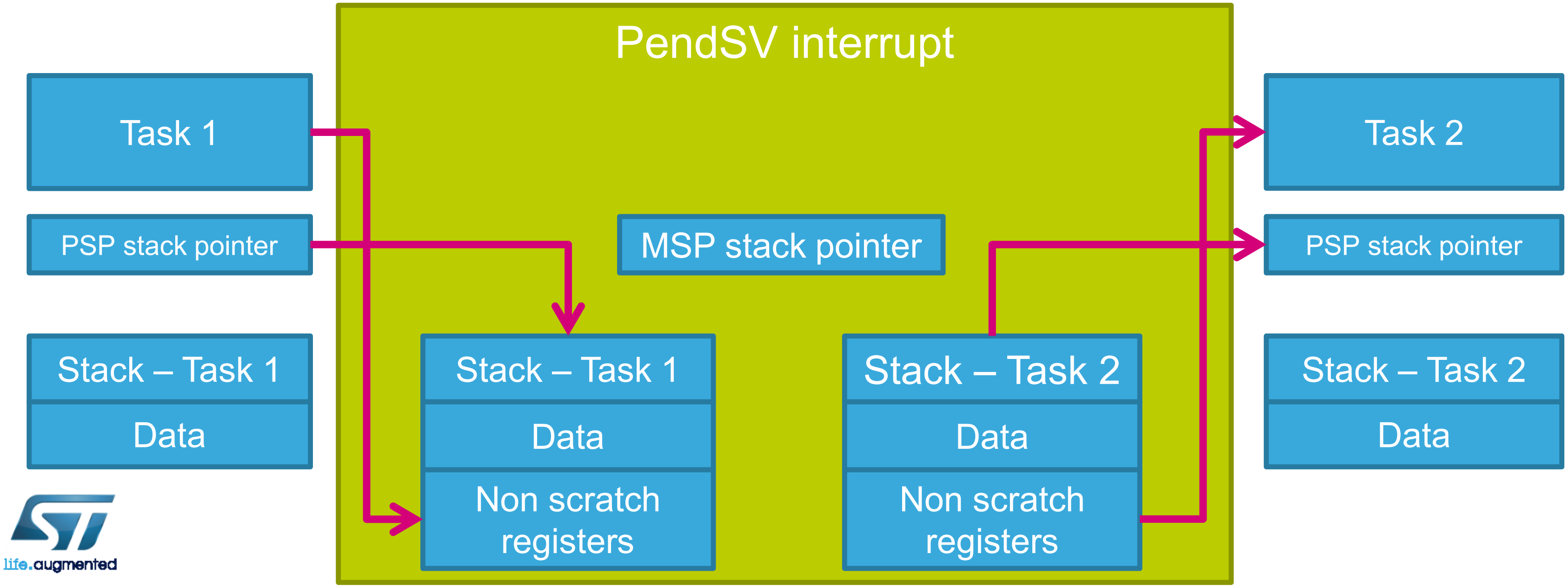
- PendSV interrupt
  - In this interrupt is the scheduler
  - Lowest NVIC interrupt priority
  - Not triggered by any periphery
  - Pending state set from other interrupts
  - Or from task which want end earlier (non MPU version)
- SVC interrupt
  - Interrupt risen by SVC instruction
  - Called if task want end earlier (MPU version)
  - In this interrupt set pending state PendSV (MPU version)
- SysTick timer
  - Set PendSV is context switch is



FreeRTOS
Stack pointer

- Main stack pointer
  - Used in interrupts
  - Allocated by linker during compiling

- Process stack pointer
  - Each task have own stack pointer
  - During context switch the stack pointer is initialized for correct task



Tasks API

- Create task

```
osThreadId osThreadCreate (const osThreadDef_t *thread_def, void *argument)
```

- Delete task

```
osStatus osThreadTerminate (osThreadId thread_id)
```

- Get task ID

```
osThreadId osThreadGetId (void)
```

- Task handle definition:

```
/* Private variables ---------------------------------------------------------*/
osThreadId Task1Handle;
```

- Create Task

```
 /* Create the thread(s) */
  /* definition and creation of Task1 */
  osThreadDef(Task1, StartTask1, osPriorityNormal, 0, 128);
  Task1Handle = osThreadCreate(osThread(Task1), NULL);
```

Tasks API

- Check if task is suspended

```
osStatus osThreadIsSuspended(osThreadId thread_id)
```

- Resume task

```
osStatus osThreadResume (osThreadId thread_id)
```

- Check state of task

```
osThreadState osThreadGetState(osThreadId thread_id)
```

- Suspend task

```
osStatus osThreadSuspend (osThreadId thread_id)
```

- Resume all tasks

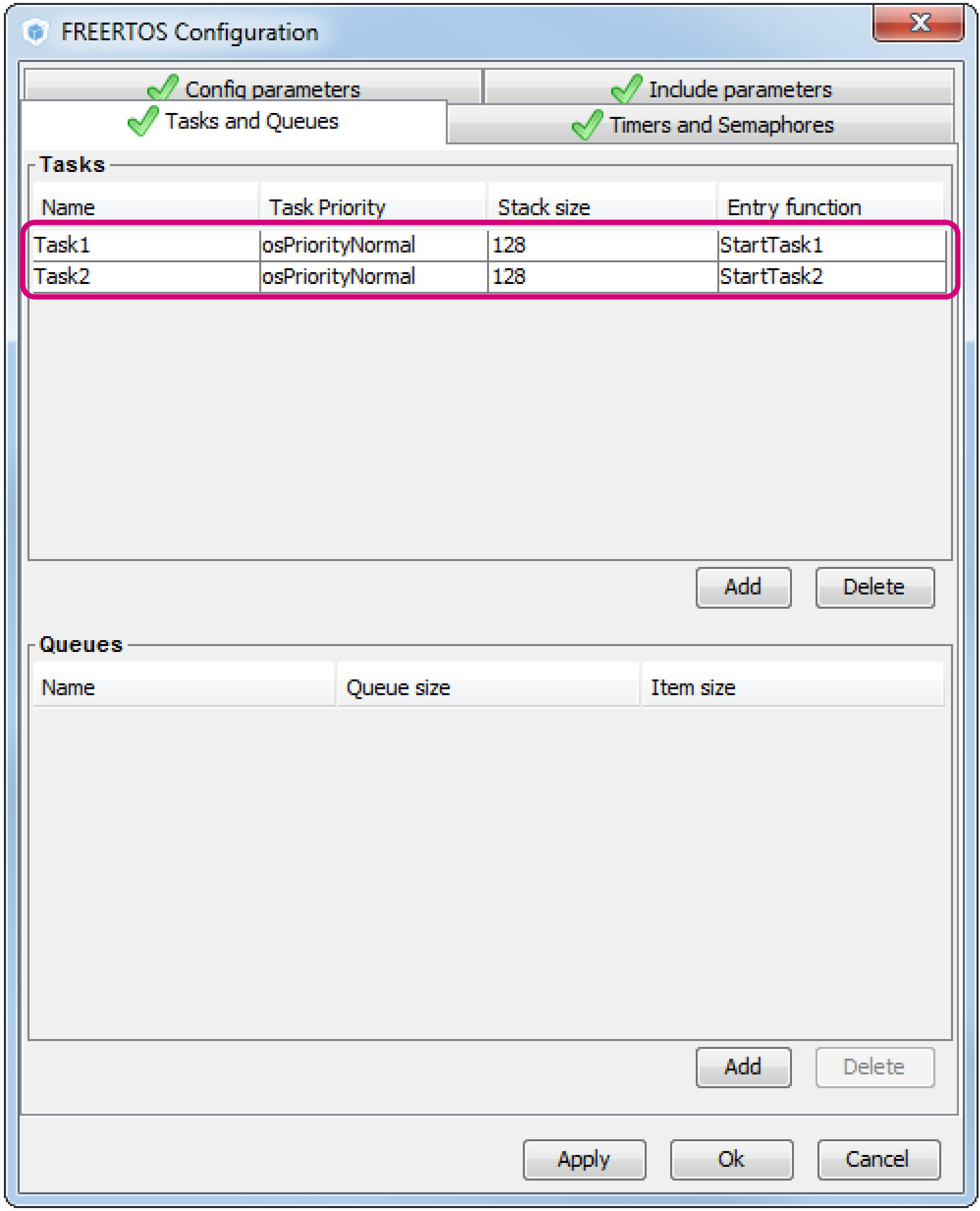
```
osStatus osThreadResumeAll (void)
```

- Suspend all tasks

```
osStatus osThreadSuspendAll (void)
```

Tasks lab

- By default defined one defaultTask
- Task is defined by
  - Name
  - Priority
  - Stack size
  - Name of entry function
- Define two tasks
  - Task1
  - Task2
- With same priority



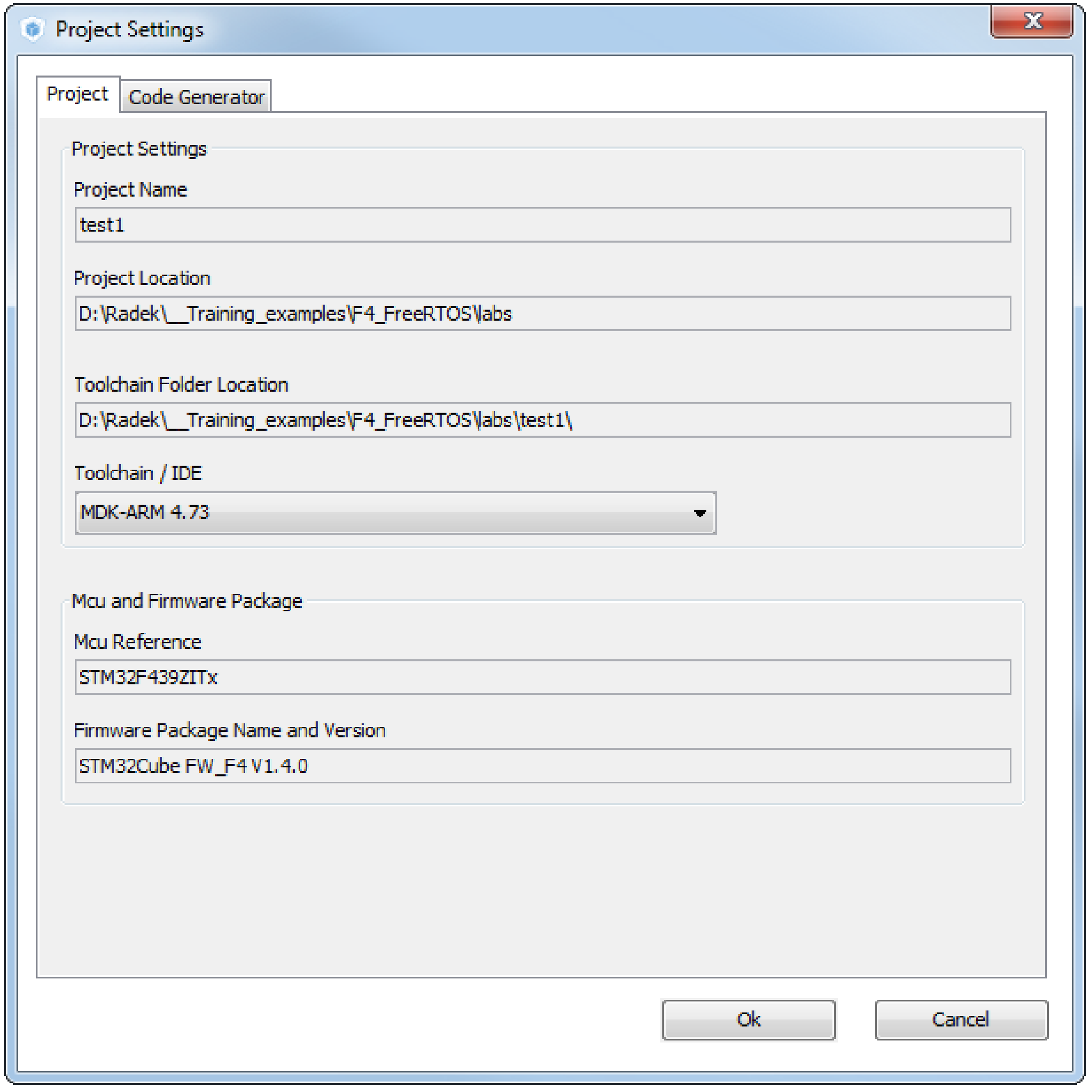
Name	Task Priority	Stack size	Entry function
Task1	osPriorityNormal	128	StartTask1
Task2	osPriorityNormal	128	StartTask2

Name	Queue size	Item size

Tasks lab

- Now we set the project details for generation
  - Menu > Project > Project Settings
  - Set the project name
  - Project location
  - Type of toolchain

- Now we can Generate Code
  - Menu > Project > Generate Code



Project Settings

Project Name: test1

Project Location: D:\Radek\__Training_examples\F4_FreeRTOS\labs

Toolchain Folder Location: D:\Radek\__Training_examples\F4_FreeRTOS\labs\test1\

Toolchain / IDE: MDK-ARM 4.73

Mcu and Firmware Package

Mcu Reference: STM32F439ZITx

Firmware Package Name and Version: STM32Cube FW_F4 V1.4.0

Tasks lab

- Any component in FreeRTOS need to have handle, very similar to CubeMX

```
/* Private variables ---------------------------------------------------------*/
osThreadId Task1Handle;
osThreadId Task2Handle;
```

- Task function prototypes, names was taken from CubeMX

```
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
void StartTask1(void const * argument);
void StartTask2(void const * argument);
```

- Before the scheduler is start we must create tasks

```
 /* Create the thread(s) */
  /* definition and creation of Task1 */
  osThreadDef(Task1, StartTask1, osPriorityNormal, 0, 128);
  Task1Handle = osThreadCreate(osThread(Task1), NULL);

  /* definition and creation of Task2 */
  osThreadDef(Task2, StartTask2, osPriorityNormal, 0, 128);
  Task2Handle = osThreadCreate(osThread(Task2), NULL);
```

Define task parameters

Create task, allocate memory

Tasks lab

- Start the scheduler, the scheduler function newer ends

```
 /* Start scheduler */
  osKernelStart();
  /* We should never get here as control is now taken by the scheduler */
```

- On first task run StartTask1 is called

- Task must have inside infinity loop in case we don’t want to end the task

```
void StartTask1(void const * argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
    printf("Task 1\n");
    osDelay(1000);
  }
  /* USER CODE END 5 */
}
```

Endless loop

osDealy will start context switch

Tasks lab

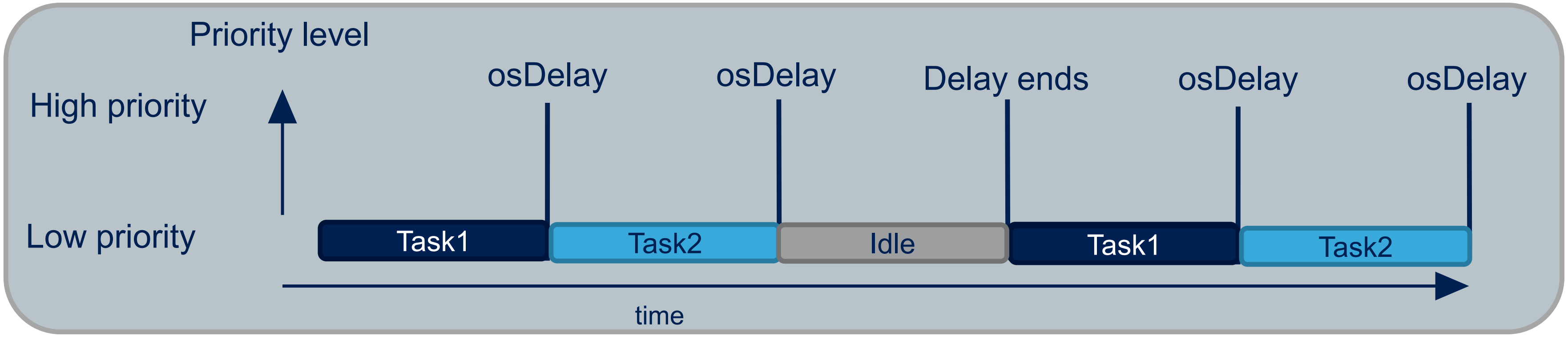
- Second loop is same as previous

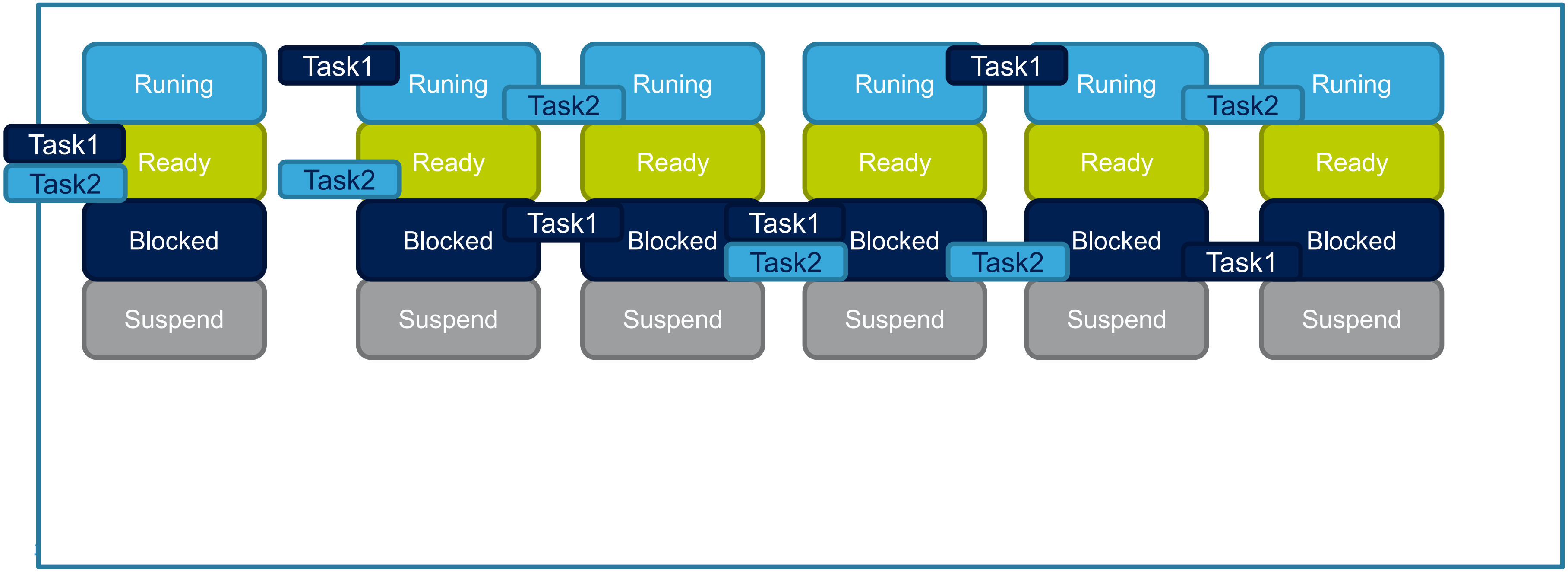
```
/* StartTask2 function */
void StartTask2(void const * argument)
{
  /* USER CODE BEGIN StartTask2 */
  /* Infinite loop */
  for(;;)
  {
    printf("Task 2\n");
    osDelay(1000);
  }
  /* USER CODE END StartTask2 */
}
```

- Compile and run project in debug and watch terminal window

Tasks lab

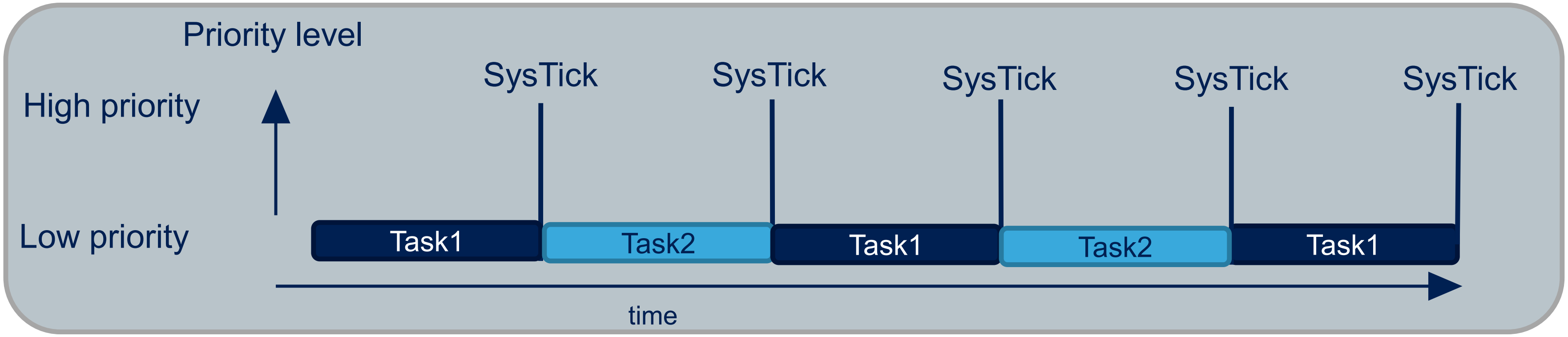
- If both Dealys are processed the FreeRTOS is in idle state

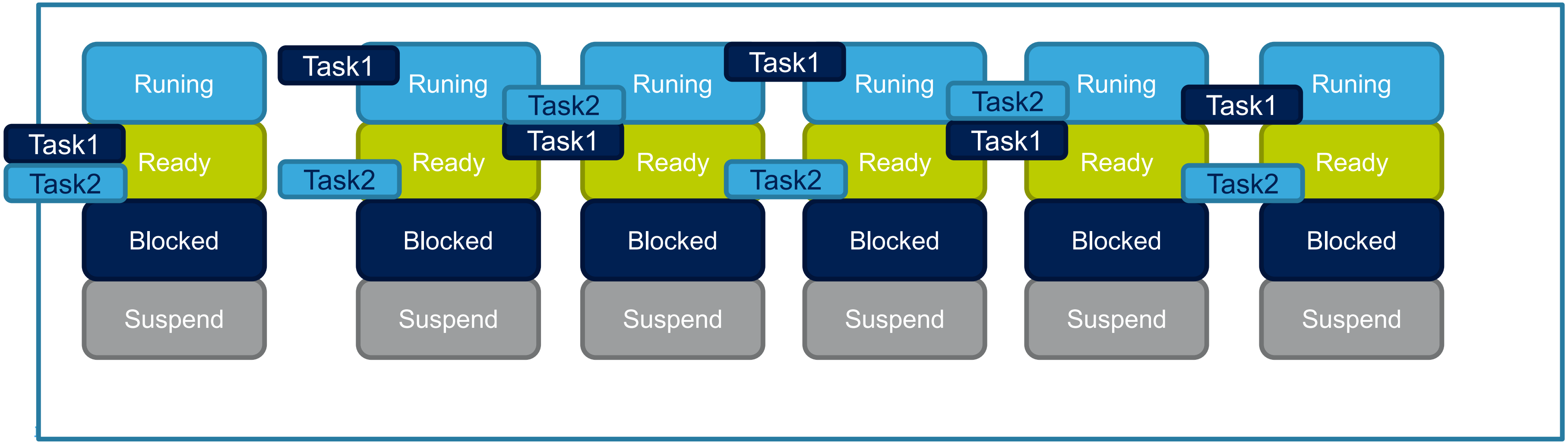




Tasks lab

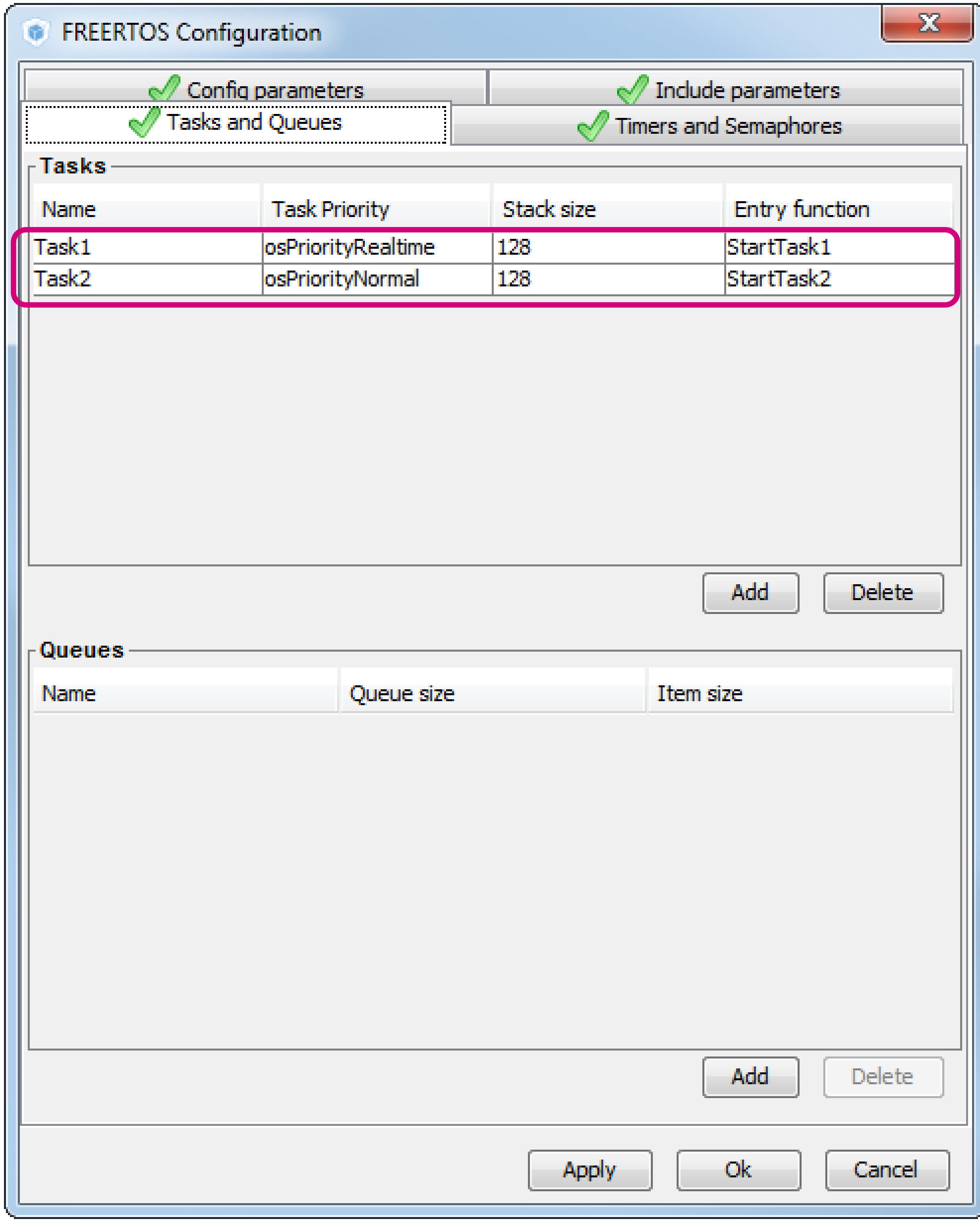
- Without Delays the threads will be in running state or in Ready state
- Use HAL_Delay





Tasks lab

- Set one task Higher priority
- Double click on task for change
- Button OK



FREERTOS Configuration

✓ Config parameters ✓ Include parameters ✓ Tasks and Queues ✓ Timers and Semaphores

Tasks

Name	Task Priority	Stack size	Entry function
Task1	osPriorityRealtime	128	StartTask1
Task2	osPriorityNormal	128	StartTask2

Queues

Name	Queue size	Item size

Tasks lab

- After we 5x times send text put task to block state
- Because task have high priority it allow to run lower priority task

```
/* USER CODE END 4 */
void StartTask1(void const * argument)
{
  /* USER CODE BEGIN 5 */
  uint32_t i = 0;
  /* Infinite loop */
  for(;;)
  {
    for (i = 0; i < 5; i++){
      printf("Task 1\n");
      HAL_Delay(50);
    }
    osDelay(1000);
  }
  /* USER CODE END 5 */
}
```

Helps not spam terminal

Block task

Tasks lab

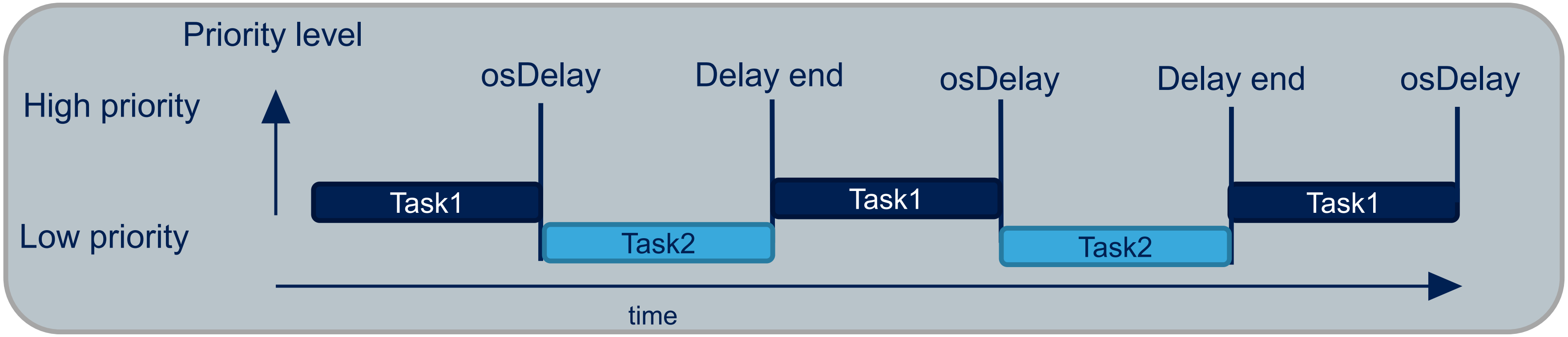
- If higher priority task is not running we can print text from this task

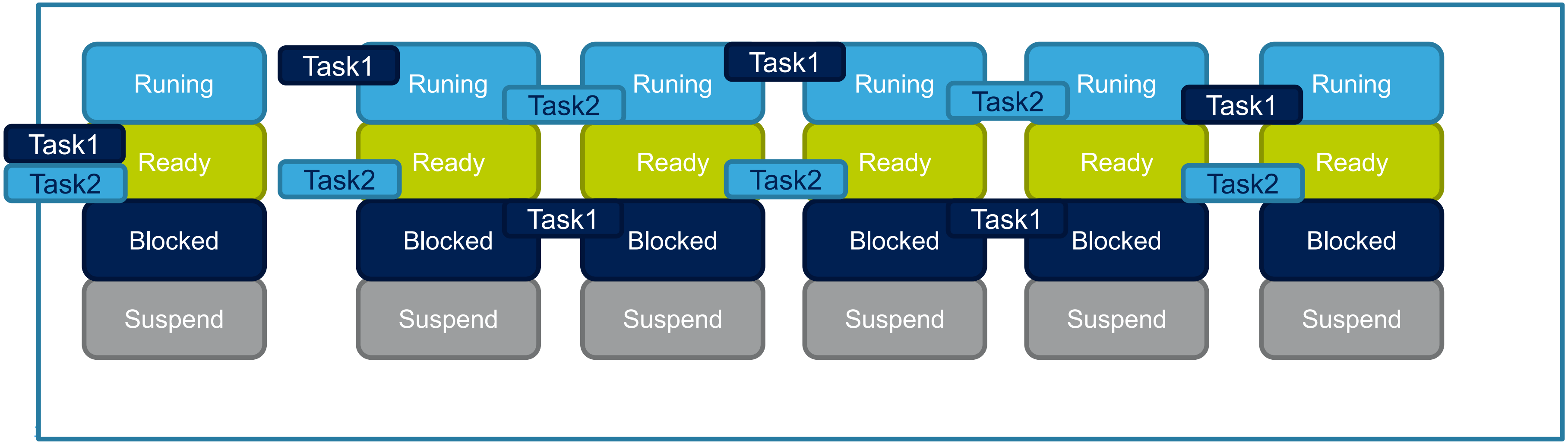
```
/* StartTask2 function */
void StartTask2(void const * argument)
{
  /* USER CODE BEGIN StartTask2 */
  /* Infinite loop */
  for(;;)
  {
    printf("Task 2\n");
    HAL_Delay(50);
  }
  /* USER CODE END StartTask2 */
}
```

Helps not spam terminal

Tasks lab

- What happen if Task1 not call osDelay?





osDelay API

- Delay function

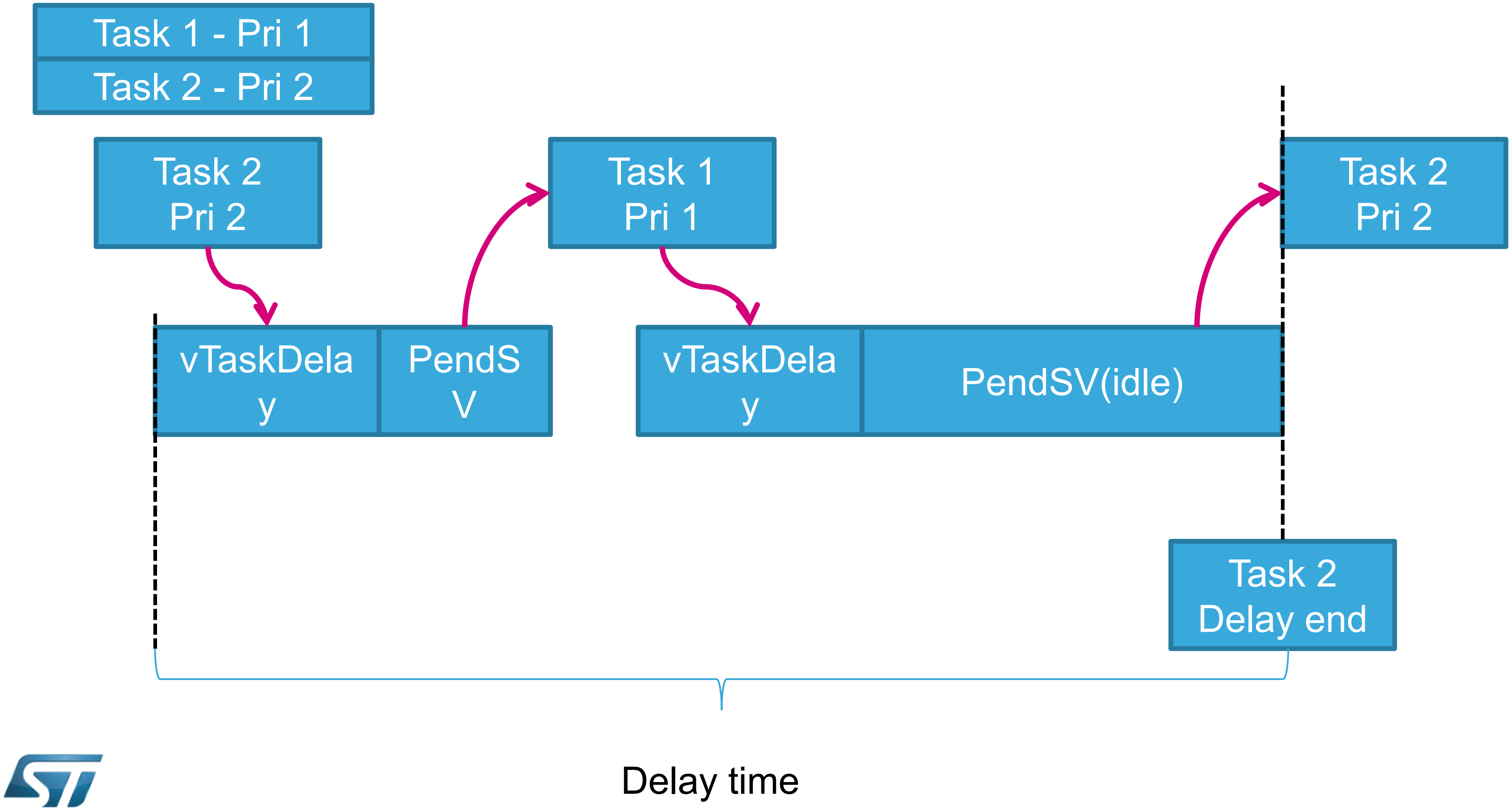
```
osStatus osDelay (uint32_t millisec)
```

- Delay function which measure time from which is delay measured

```
osStatus osDelayUntil (uint32_t PreviousWakeTime, uint32_t millisec)
```

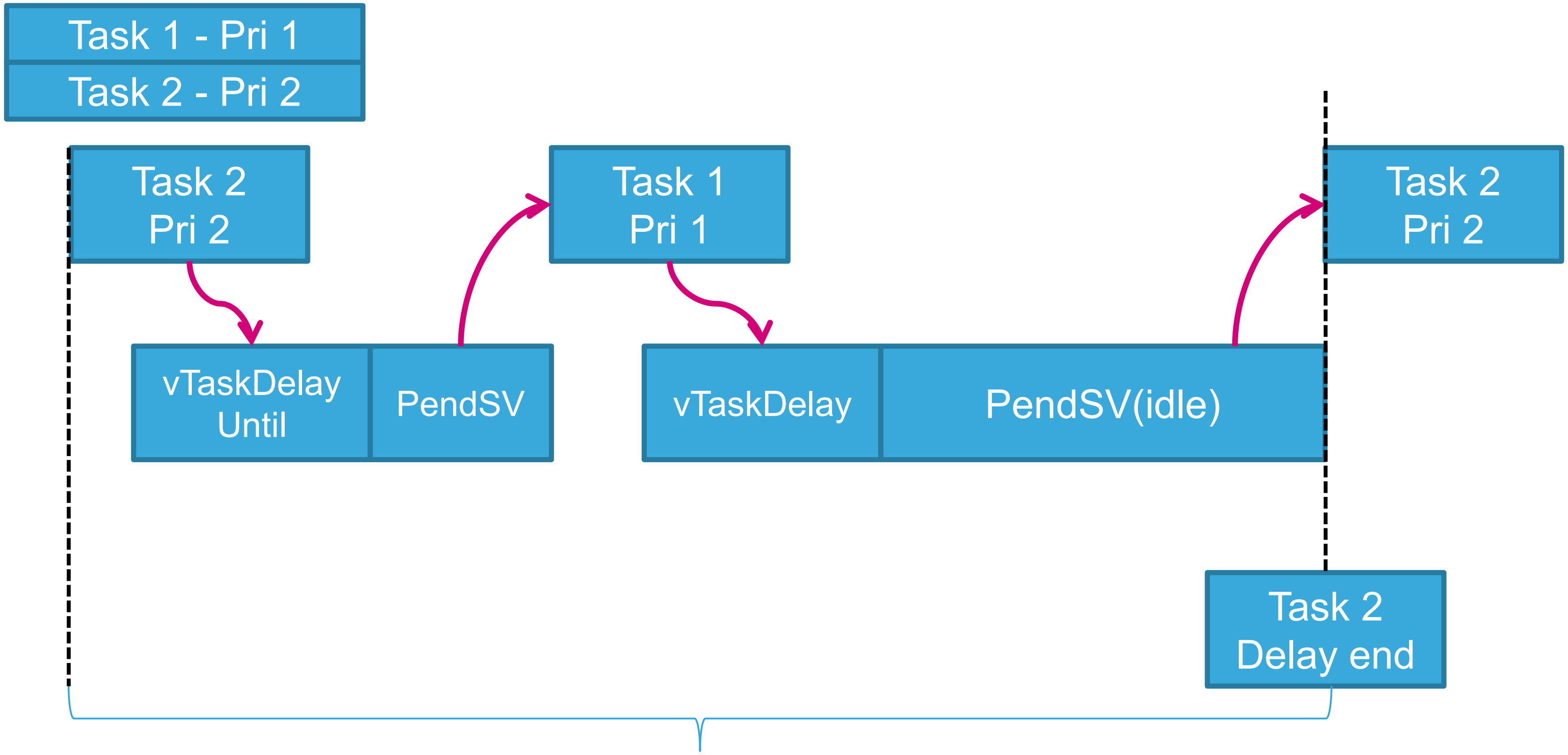
osDelay function

- osDelay start measure time from osDelay call



osDelayUntil

- osDelayUntil measure time from point which we selected
- This allow us to call task in regular intervals



DelayUntil time

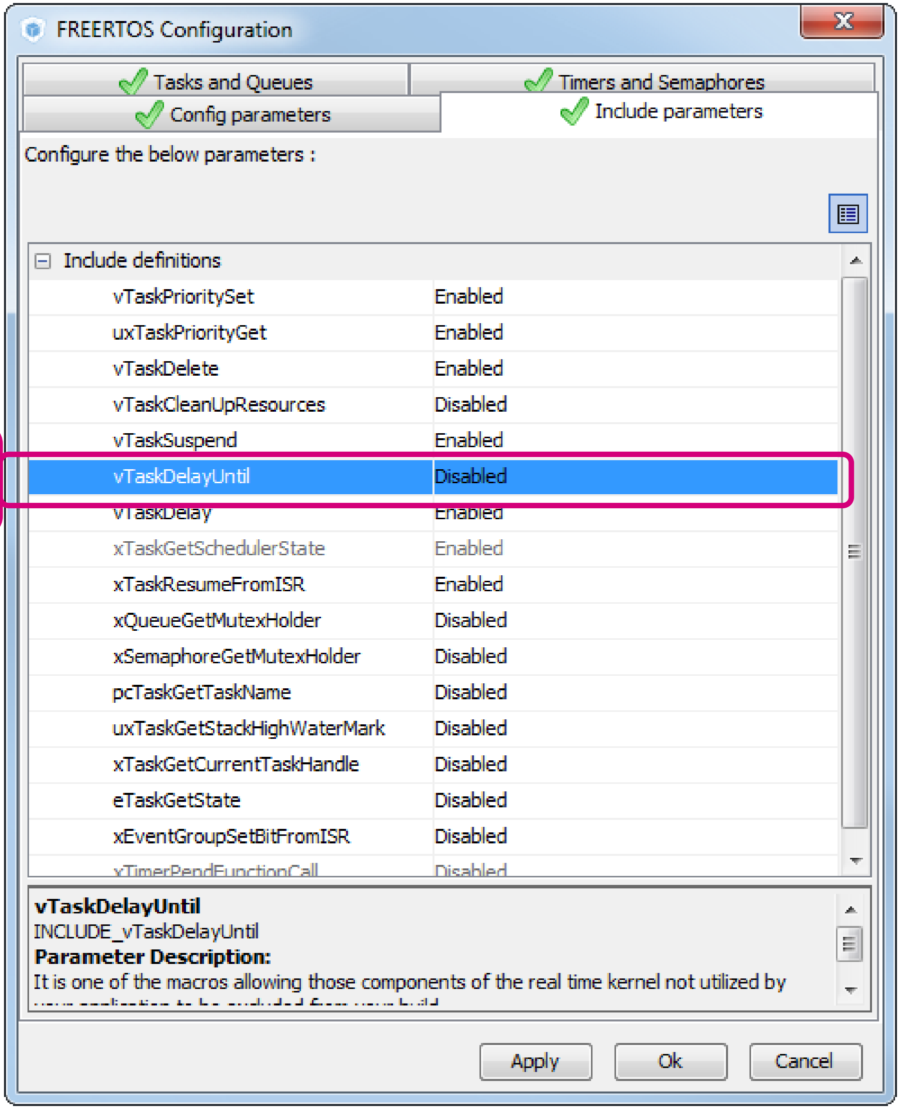
osDelay and osDelayUntil

- Enable vTaskDelayUntil in Include parameters

- Regenerate project, modify tasks to:

```c
void StartTask1(void const * argument)
{

  /* USER CODE BEGIN 5 */
  uint32_t i = 0;
  /* Infinite loop */
  for(;;)
  {
    printf("Task 1\n");
    HAL_Delay(1000);
    osDelay(2000);
  }
  /* USER CODE END 5 */
}

/* StartTask2 function */
void StartTask2(void const * argument)
{
  /* USER CODE BEGIN StartTask2 */
  /* Infinite loop */
  for(;;)
  {
    printf("Task 2\n");
    HAL_Delay(200);
  }
  /* USER CODE END StartTask2 */
}
```

Delay between two run is 2s



Configure the below parameters :

Include definitions	
vTaskPrioritySet	Enabled
uxTaskPriorityGet	Enabled
vTaskDelete	Enabled
vTaskCleanUpResources	Disabled
vTaskSuspend	Enabled
vTaskDelayUntil	Disabled
vTaskDelay	Enabled
xTaskGetSchedulerState	Enabled
xTaskResumeFromISR	Enabled
xQueueGetMutexHolder	Disabled
xSemaphoreGetMutexHolder	Disabled
pcTaskGetTaskName	Disabled
uxTaskGetStackHighWaterMark	Disabled
xTaskGetCurrentTaskHandle	Disabled
eTaskGetState	Disabled
xEventGroupSetBitFromISR	Disabled
xTimerPendFunctionCall	Disabled

vTaskDelayUntil
INCLUDE_vTaskDelayUntil
Parameter Description:
It is one of the macros allowing those components of the real time kernel not utilized by your application to be excluded from your build

osDelay and osDelayUntil

- Enable vTaskDelayUntil in Include parameters
- Regenerate project, modify tasks to:

```
void StartTask1(void const * argument)
{

  /* USER CODE BEGIN 5 */
  uint32_t wakeuptime;
  /* Infinite loop */
  for(;;)
  {
    wakeuptime=osKernelSysTick();
    printf("Task 1\n");
    HAL_Delay(1000);
    osDelayUntil(wakeuptime,2000);
  }
  /* USER CODE END 5 */

}
```

For osDelayUntil function we need mark wakeup time

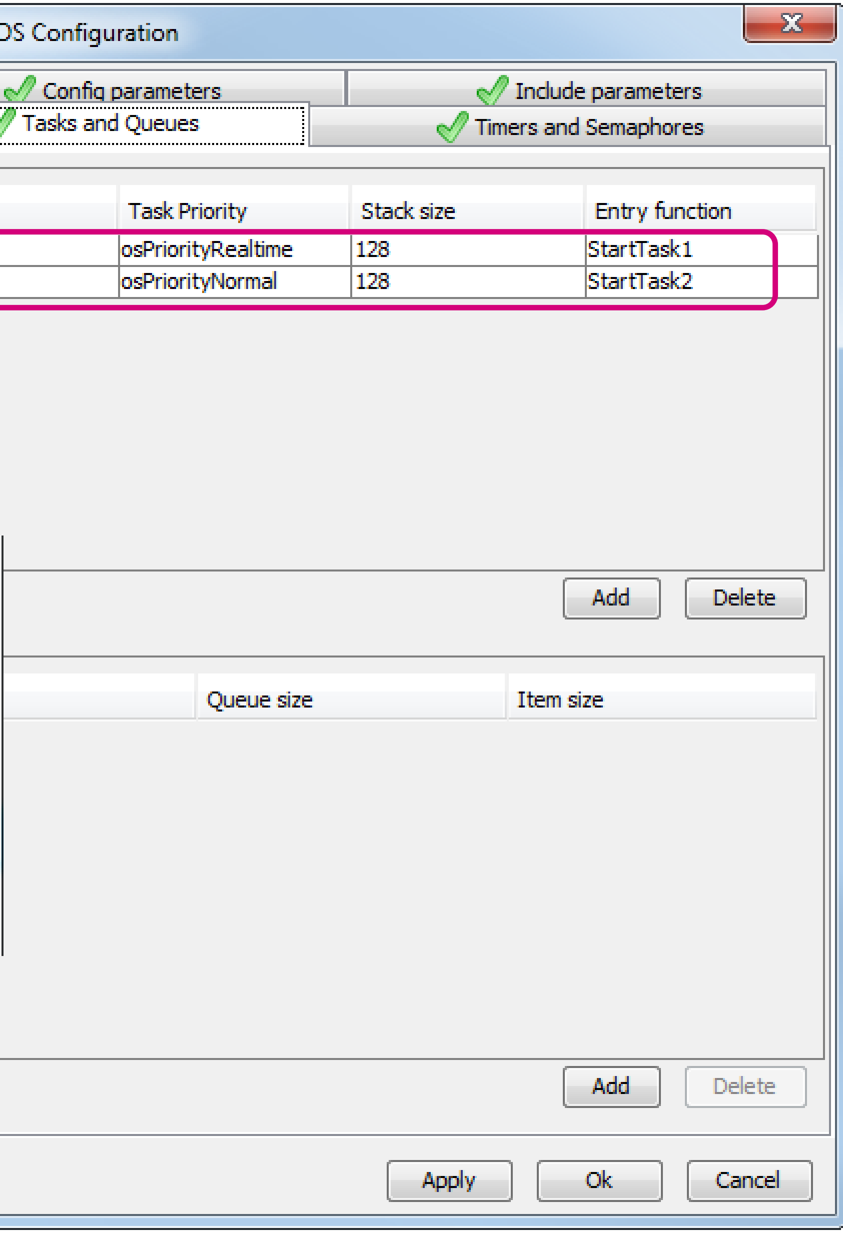
Function will be executed every 2s

Time from which is delay measured

Real delay time

Priority change lab

- Task1 have higher priority than Task2
- If not enable vTaskPriorityGet and uxTaskPrioritySet in IncludeParameters

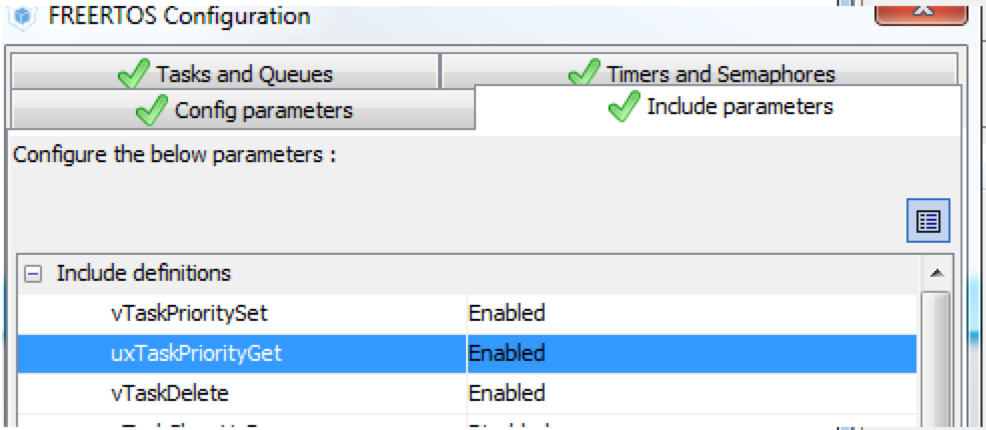


FREERTOS Configuration

Tasks

Name	Task Priority	Stack size	Entry function
Task1	osPriorityRealtime	128	StartTask1
Task2	osPriorityNormal	128	StartTask2

Queue size	Item size



FREERTOS Configuration

Configure the below parameters :

⊟ Include definitions	
vTaskPrioritySet	Enabled
uxTaskPriorityGet	Enabled
vTaskDelete	Enabled

Priority change lab

- Modify Task1 to:

```
void StartTask1(void const * argument)
{
  /* USER CODE BEGIN 5 */
  osPriority priority;
  /* Infinite loop */
  for(;;)
  {
    priority=osThreadGetPriority(Task2Handle);
    printf("Task 1\n");
    osThreadSetPriority(Task2Handle,priority+1);
    HAL_Delay(1000);
  }
  /* USER CODE END 5 */
}
```

Reads Task2 priority

Increase Task2 priority

Priority change lab

- Modify Task2 to:

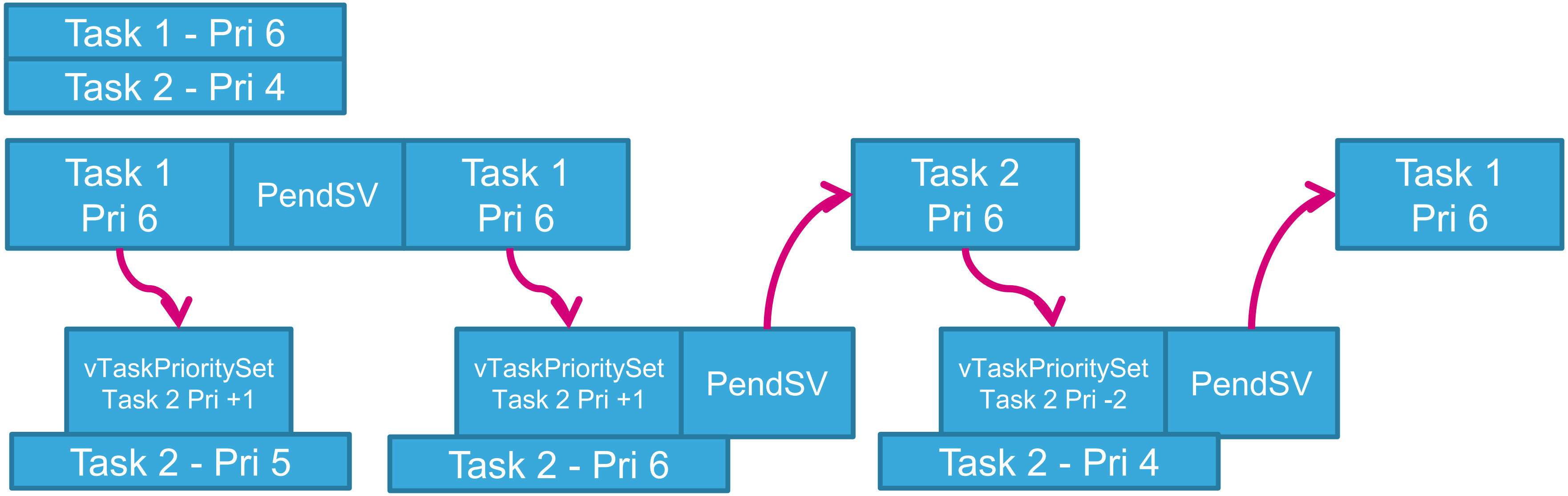
```c
/* StartTask2 function */
void StartTask2(void const * argument)
{
  /* USER CODE BEGIN StartTask2 */
  osPriority priority;
  /* Infinite loop */
  for(;;)
  {
    priority=osThreadGetPriority(NULL);
    printf("Task 2\n");
    osThreadSetPriority(NULL,priority-2);
  }
  /* USER CODE END StartTask2 */
}
```

Read priority of current task

Decrease task priority

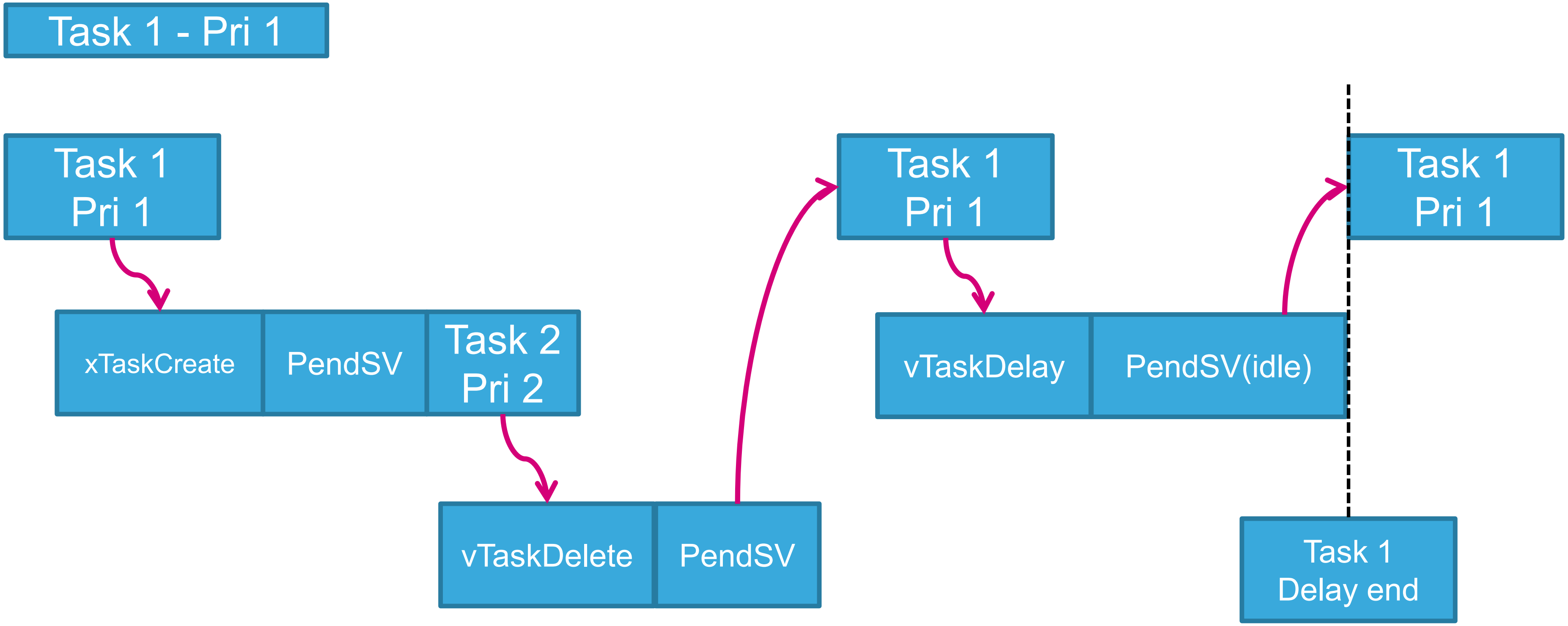
Priority change lab

- How priorities are changed?



Creating and deleting tasks lab

- Example how to create and delete tasks



Creating and deleting tasks lab

- Example how to create and delete tasks

- Comment Task2 creation part in main.c

```
 /* definition and creation of Task2 */
//  osThreadDef(Task2, StartTask2, osPriorityNormal, 0, 128);
//  Task2Handle = osThreadCreate(osThread(Task2), NULL);
```

- Modify Task1 to create task2

```
void StartTask1(void const * argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
    printf("Create task2");
    osThreadDef(Task2, StartTask2, osPriorityNormal, 0, 128);
    Task2Handle = osThreadCreate(osThread(Task2), NULL);
    osDelay(1000);
  }
  /* USER CODE END 5 */
}
```

Task 2 creation

Creating and deleting tasks lab

- Example how to create and delete tasks

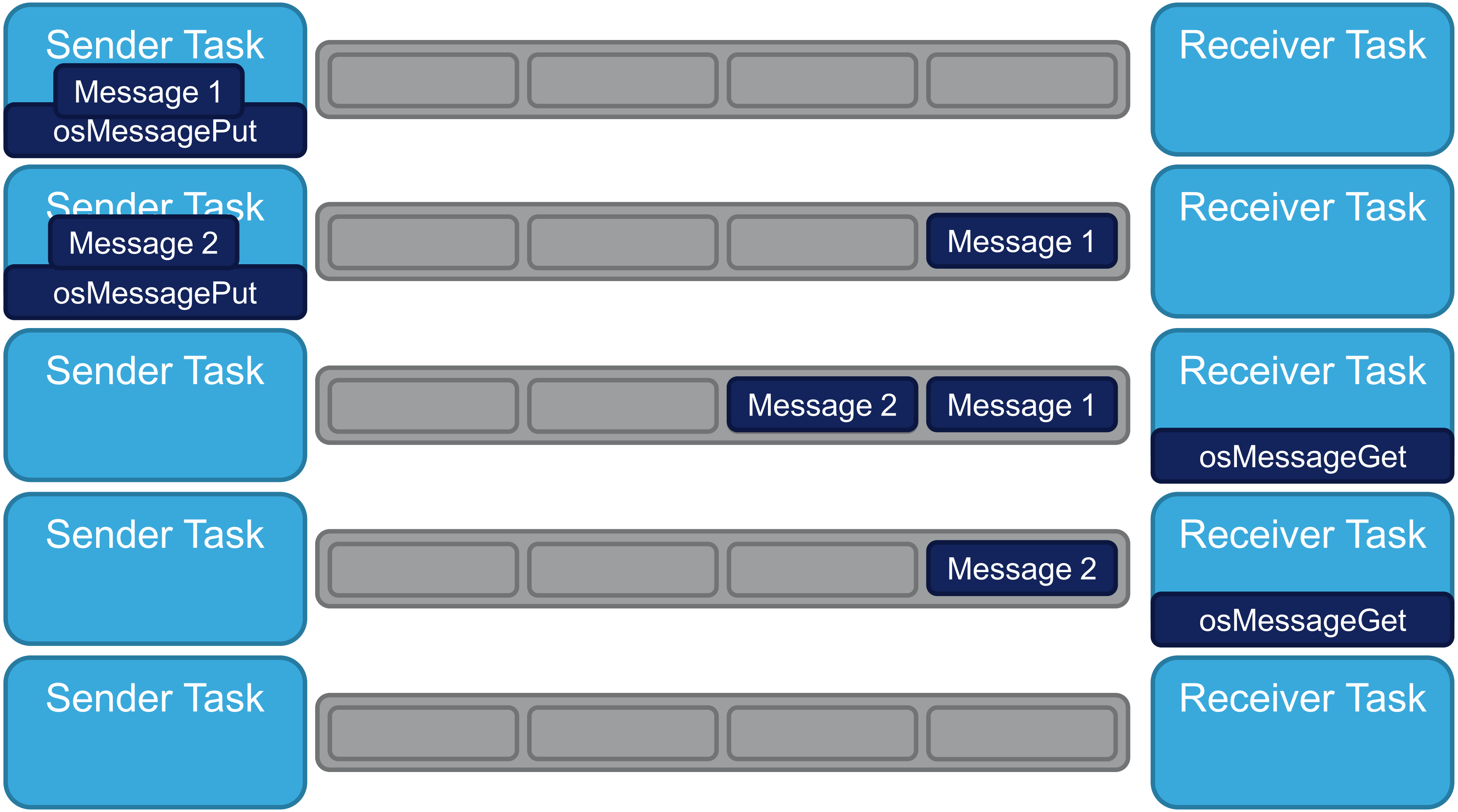
- Modift Task2 to delete him-self:

```
/* StartTask2 function */
void StartTask2(void const * argument)
{
  /* USER CODE BEGIN StartTask2 */
  /* Infinite loop */
  for(;;)
  {
    printf("Delete Task2\n");
    osThreadTerminate(Task2Handle);
  }
  /* USER CODE END StartTask2 */
}
```

Delete Task

FreeRTOS Queues

Queue


Sender Task
Message 1
osMessagePut
Receiver Task
Sender Task
Message 2
osMessagePut
Message 1
Receiver Task
Sender Task
Message 2
Message 1
Receiver Task
osMessageGet
Sender Task
Message 2
Receiver Task
osMessageGet
Sender Task
Receiver Task

Queue

- Create Queue:

```
osMessageQId osMessageCreate (const osMessageQDef_t *queue_def, osThreadId thread_id)
```

Queue Handle

Create Queue

- Put data into Queue

```
osStatus osMessagePut (osMessageQId queue_id, uint32_t info, uint32_t millisec)
```

Queue Handle

Item to send

Send timeout

- Receive data from Queue

```
osEvent osMessageGet (osMessageQId queue_id, uint32_t millisec)
```

Structure with status and with received item

Queue handle

Receiving timeout

Queue

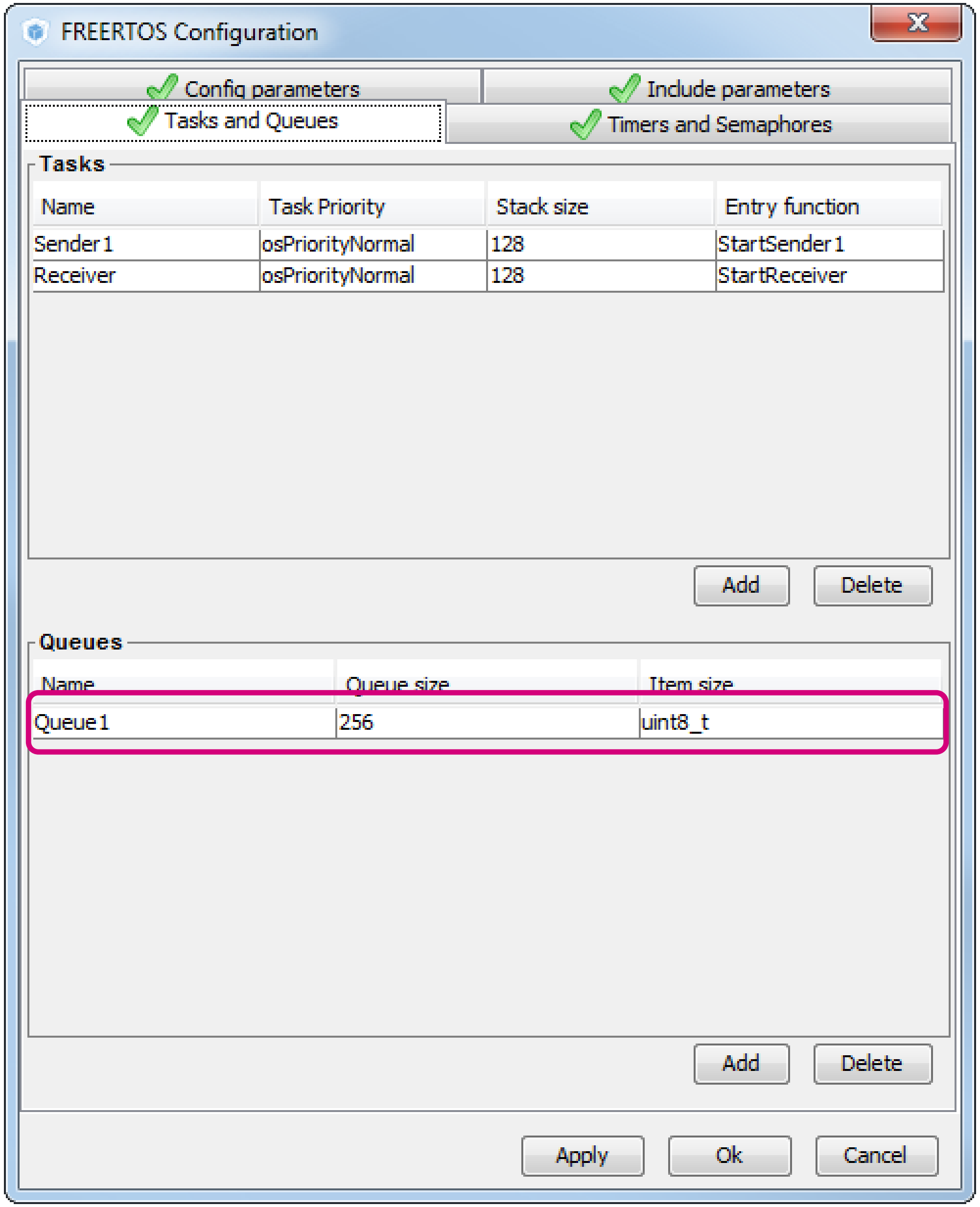
- osEvent structure

```
typedef struct  {
  osStatus                 status;     ///< status code: event or error information
  union  {
    uint32_t                    v;     ///< message as 32-bit value
    void                       *p;     ///< message or mail as void pointer
    int32_t               signals;     ///< signal flags
  } value;                             ///< event value
  union  {
    osMailQId             mail_id;     ///< mail id obtained by \ref osMailCreate
    osMessageQId       message_id;     ///< message id obtained by \ref osMessageCreate
  } def;                               ///< event definition
} osEvent;
```

- If we want to get data from osEvent we must use:
  - osEventName.v if the value is 32bit message(or 8/16bit)
  - osEventName.p and retype on selected datatype

Queue lab

- Set both tasks to normal priority
- Queue part
- Button Add
- Set queue size to 256
- Queue type to uint8_t
- Button OK



Tasks

Name	Task Priority	Stack size	Entry function
Sender1	osPriorityNormal	128	StartSender1
Receiver	osPriorityNormal	128	StartReceiver

Queues

Name	Queue size	Item size
Queue1	256	uint8_t

Queue lab

- Queue handler is now defined

```
/* Private variables ---------------------------------------------------------*/
osThreadId Sender1Handle;
osThreadId ReceiverHandle;
osMessageQId Queue1Handle;
```

- Queue item type initialization, length definition and create of queue and memory allocation

```
 /* Create the queue(s) */
  /* definition and creation of Queue1 */
  osMessageQDef(Queue1, 256, uint8_t);
  Queue1Handle = osMessageCreate(osMessageQ(Queue1), NULL);
```

Queue item definition

Queue size

Queue lab

- Sender1 task

```
void StartSender1(void const * argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
    printf("Task1\n");
    osMessagePut(Queue1Handle,0x1,200);
    printf("Task1 delay\n");
    osDelay(1000);
  }
  /* USER CODE END 5 */
}
```

Put value ‘1’ into queue

Item to send

Timeout for send

Queue handle

Queue lab

- Teceiver task

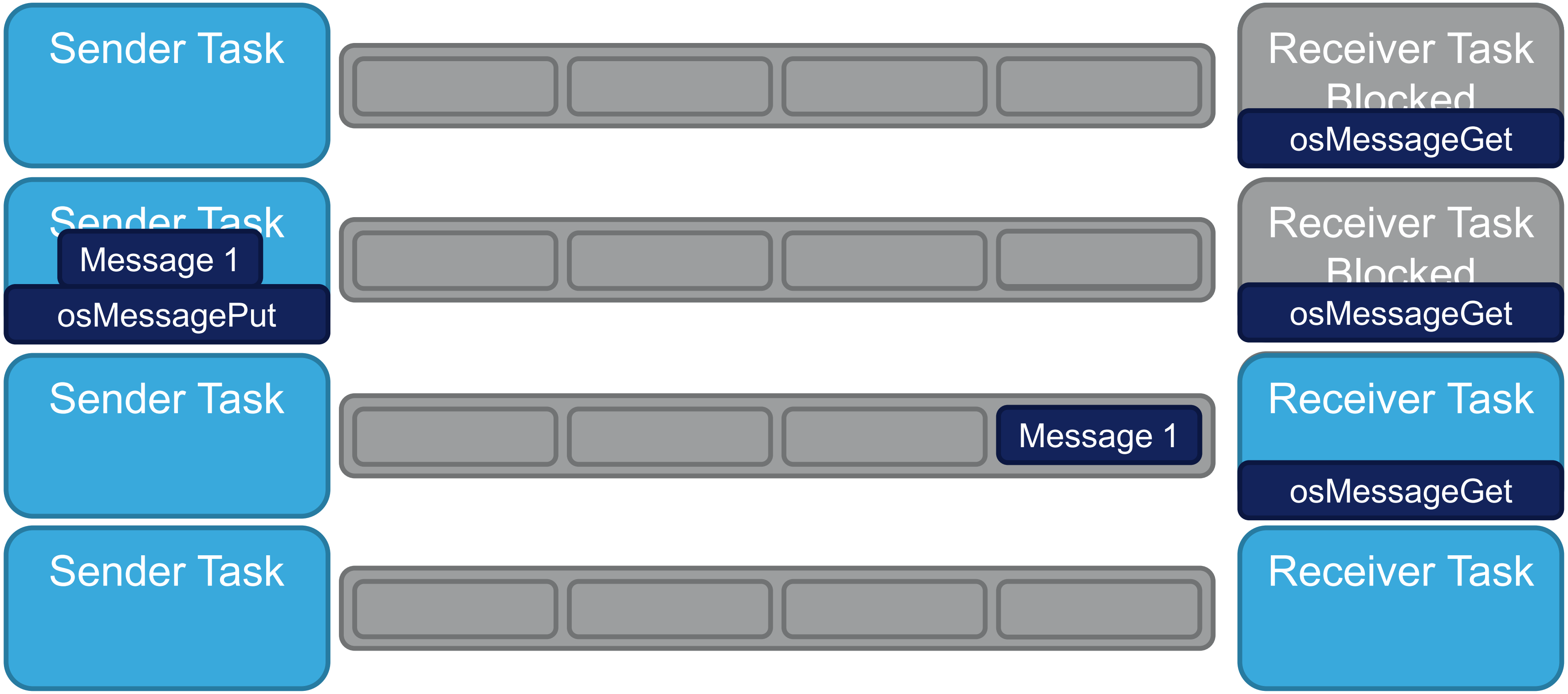
```
/* StartReceiver function */
void StartReceiver(void const * argument)
{
  /* USER CODE BEGIN StartReceiver */
  osEvent retvalue;
  /* Infinite loop */
  for(;;)
  {
    printf("Task2\n");
    retvalue=osMessageGet(Queue1Handle,4000);
    printf(“%d \n",retvalue.value.p);
  }
  /* USER CODE END StartReceiver */
}
```

Get item from queue

How long we wait on data in queue
It will Block task

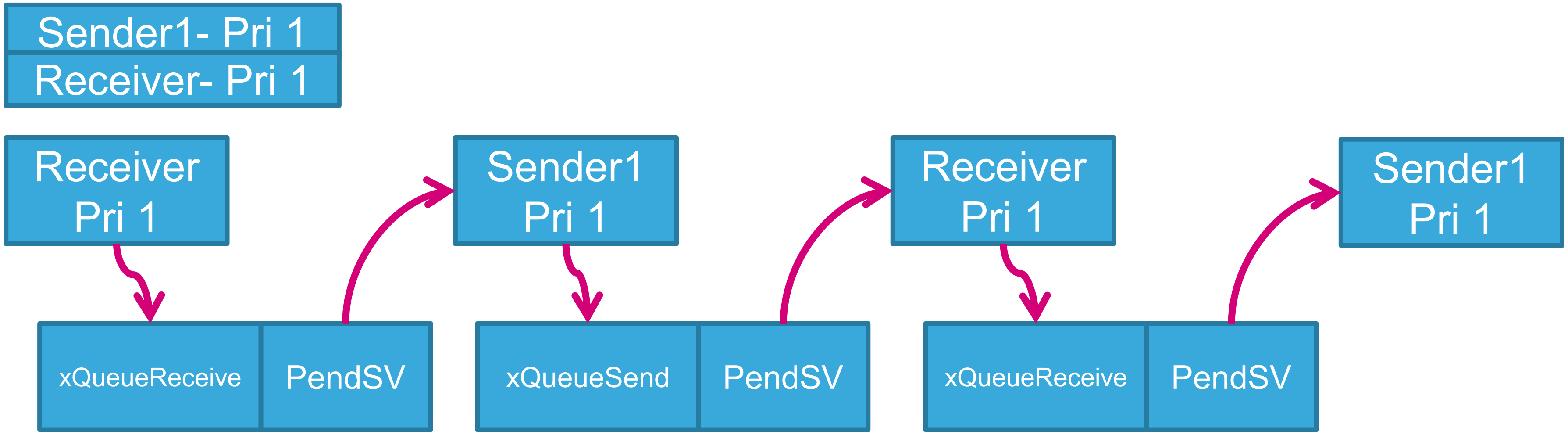
Queue handle

Queue Blocking



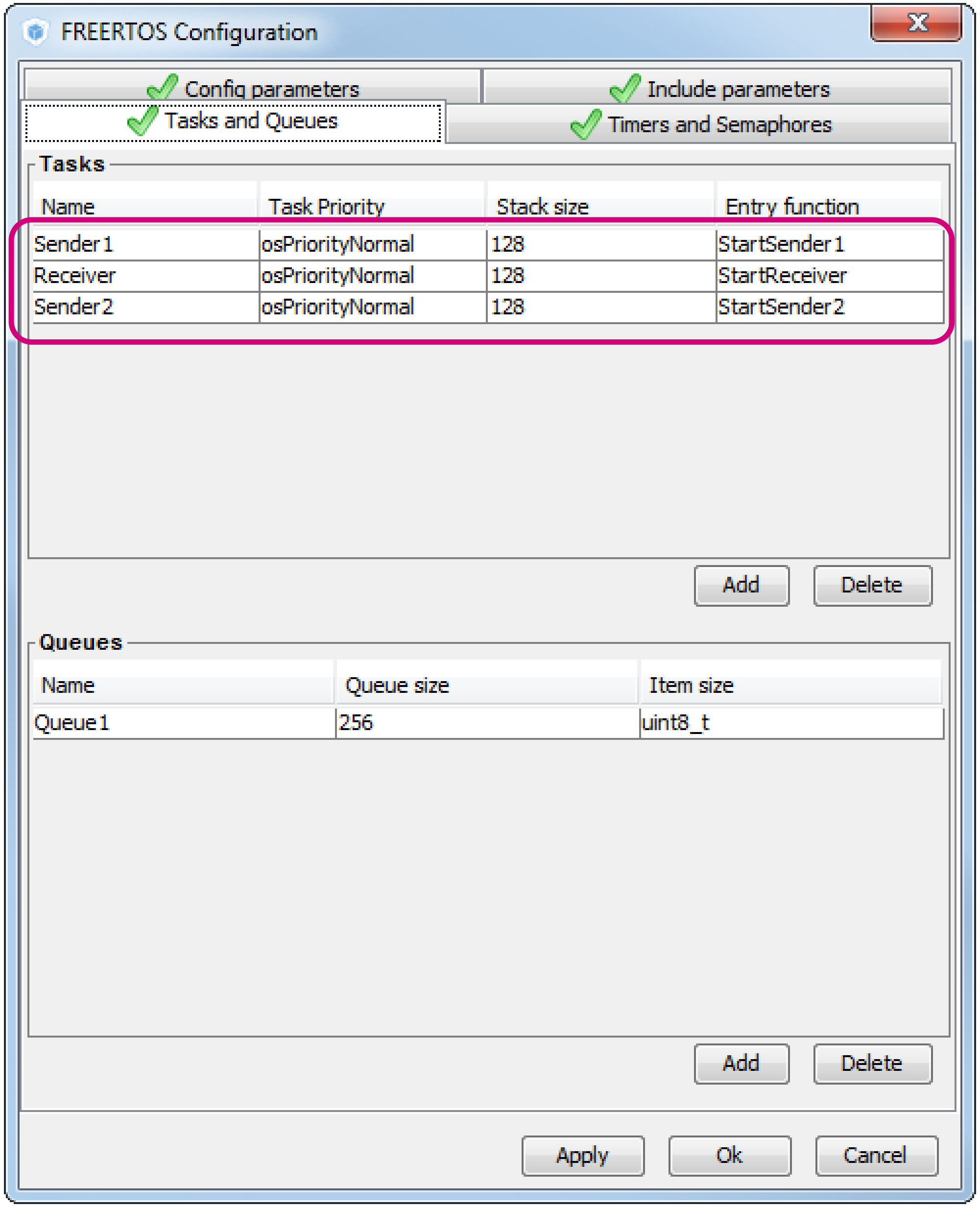
Queue Blocking

- After calling osMessageGet
- If any data are not in queue the task I blocked for settable time
- If the data are in queue the task will continue



Two senders lab

- Two sending tasks
- One receivers tasks
- Same priorities



Tasks

Name	Task Priority	Stack size	Entry function
Sender1	osPriorityNormal	128	StartSender1
Receiver	osPriorityNormal	128	StartReceiver
Sender2	osPriorityNormal	128	StartSender2

Queues

Name	Queue size	Item size
Queue1	256	uint8_t

Two senders lab

- Two sending tasks
- They are same no change necessary

```
void StartSender1(void const * argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
    printf("Task1\n");
    osMessagePut(Queue1Handle,0x1,200);
    printf("Task1 delay\n");
    osDelay(2000);
  }
  /* USER CODE END 5 */
}
```

```
void StartSender2(void const * argument)
{
  /* USER CODE BEGIN StartSender2 */
  /* Infinite loop */
  for(;;)
  {
    printf("Task2\n");
    osMessagePut(Queue1Handle,0x2,200);
    printf("Task2 delay\n");
    osDelay(2000);
  }
  /* USER CODE END StartSender2 */
}
```

life.augmented

Two senders lab

- Simple receiver

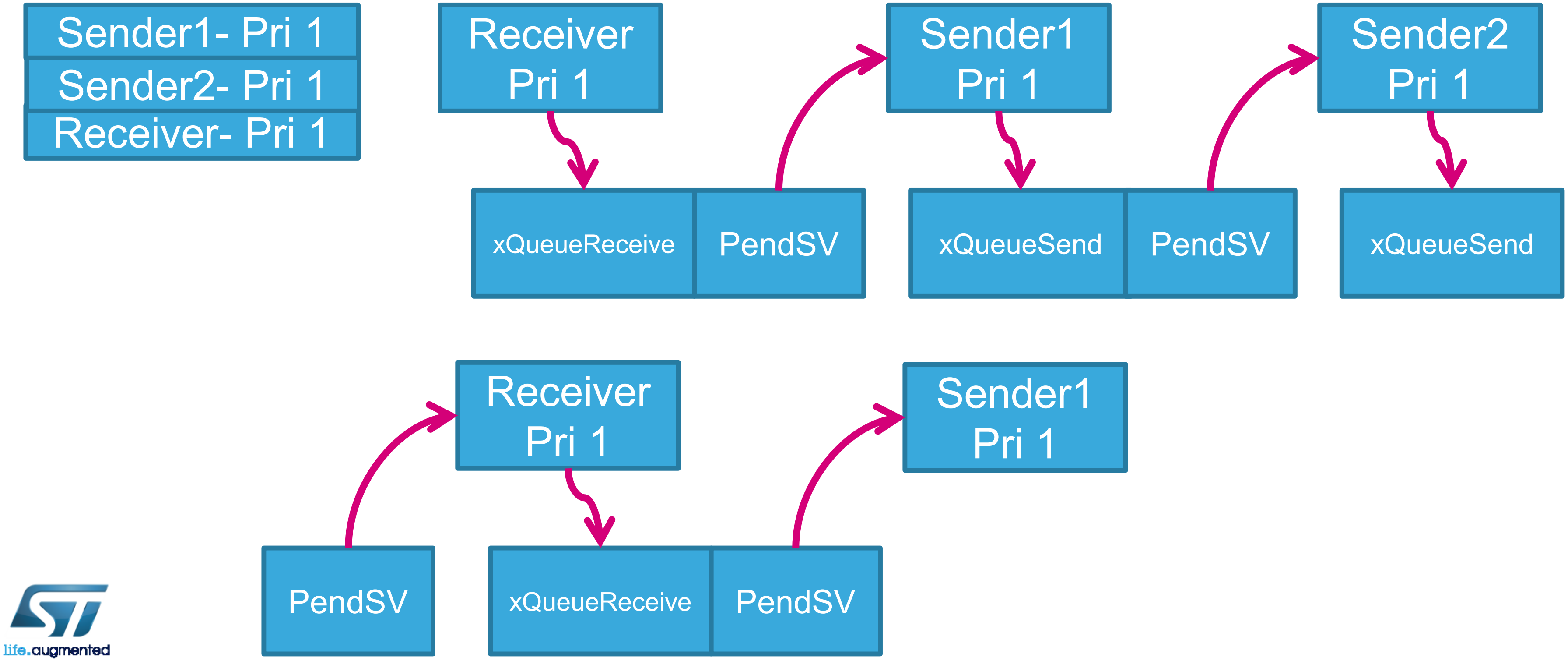
```
/* StartReceiver function */
void StartReceiver(void const * argument)
{
  /* USER CODE BEGIN StartReceiver */
  osEvent retvalue;
  /* Infinite loop */
  for(;;)
  {
    retvalue=osMessageGet(Queue1Handle,4000);
    printf("Receiver\n");
    printf("%d \n",retvalue.value.p);
  }
  /* USER CODE END StartReceiver */
}
```

Two senders lab

- What we can see in debug now?

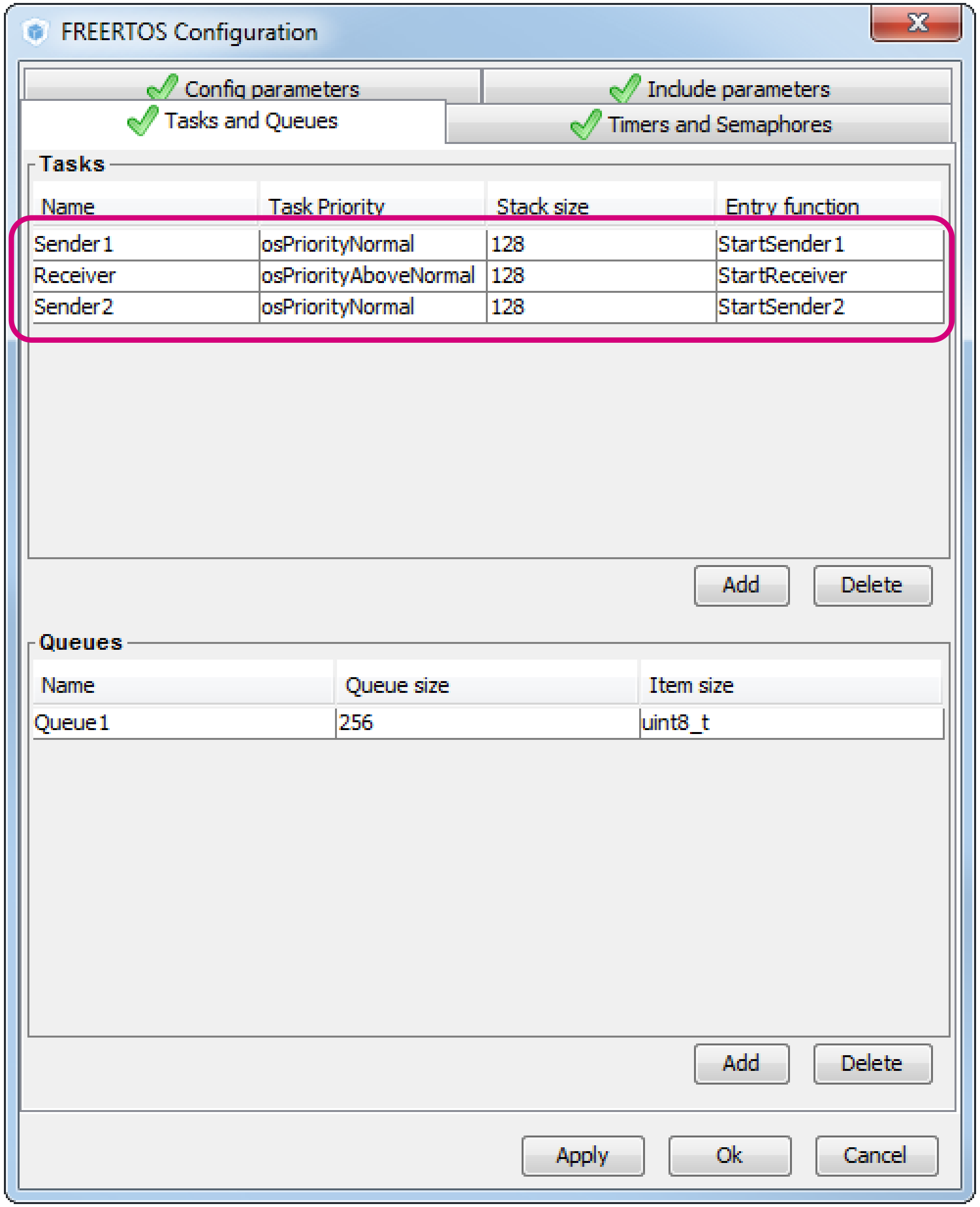
- Because tasks have same priority, receiver will get data from queue after both task put data into queue

- What happened if will be more tesks?



Receiver with higher priority lab

- Senders have same priority
- Receiver have higher priority than senders



FREERTOS Configuration

Tasks

Name	Task Priority	Stack size	Entry function
Sender1	osPriorityNormal	128	StartSender1
Receiver	osPriorityAboveNormal	128	StartReceiver
Sender2	osPriorityNormal	128	StartSender2

Queues

Name	Queue size	Item size
Queue1	256	uint8_t

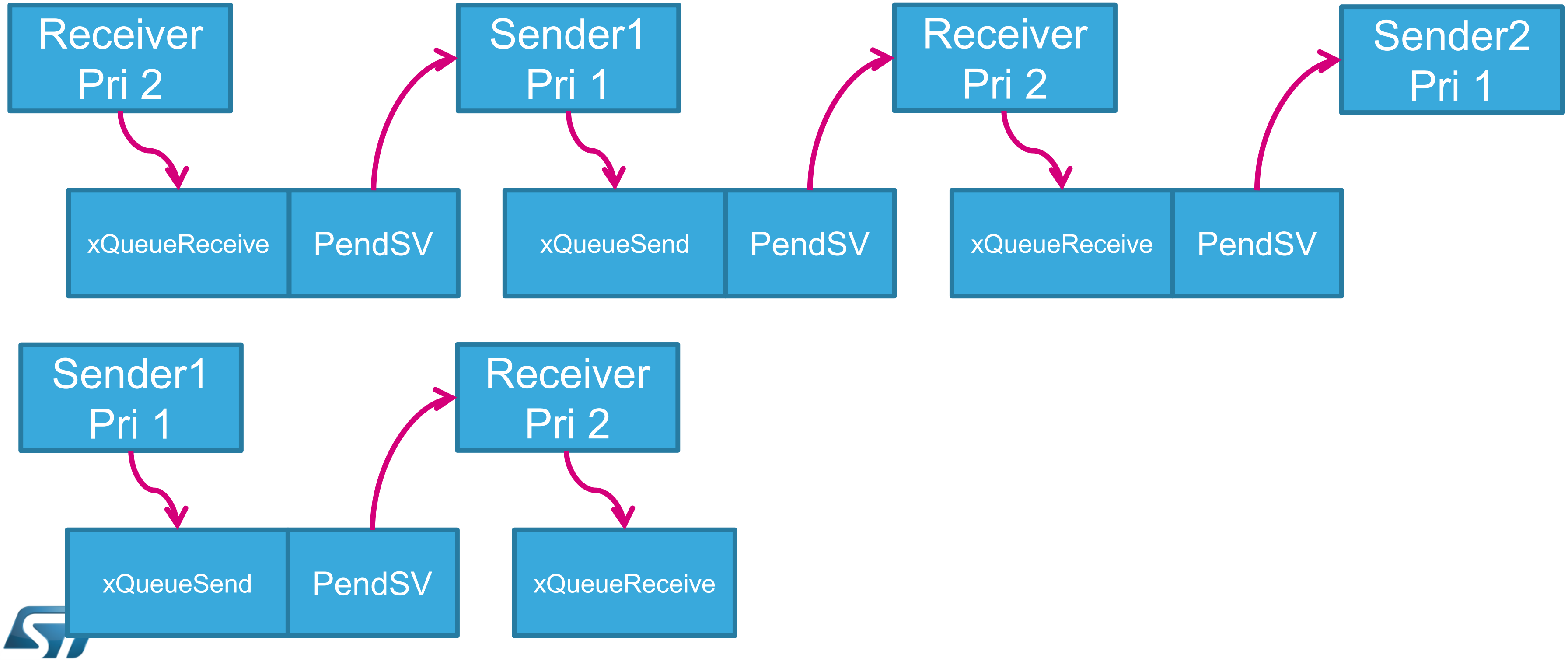
Receiver with higher priority lab

- Receiver is now unblocked every time when sender tasks put data into queue

Sender1- Pri 1

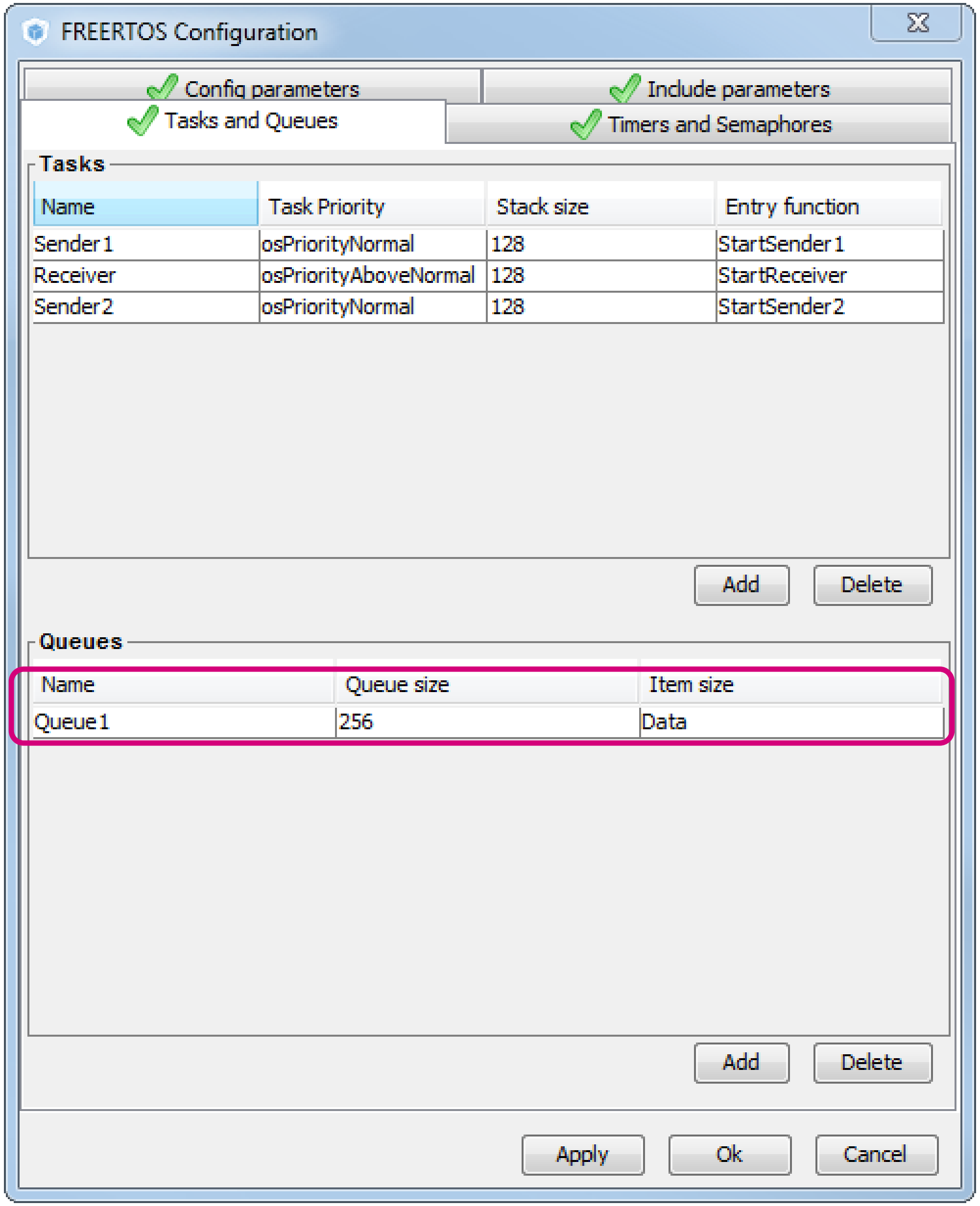
Sender2- Pri 1

Receiver- Pri 2



Queue items lab

- Queues allow to define type (different variables or structures) which the queue use.
- Regenerate project



Tasks

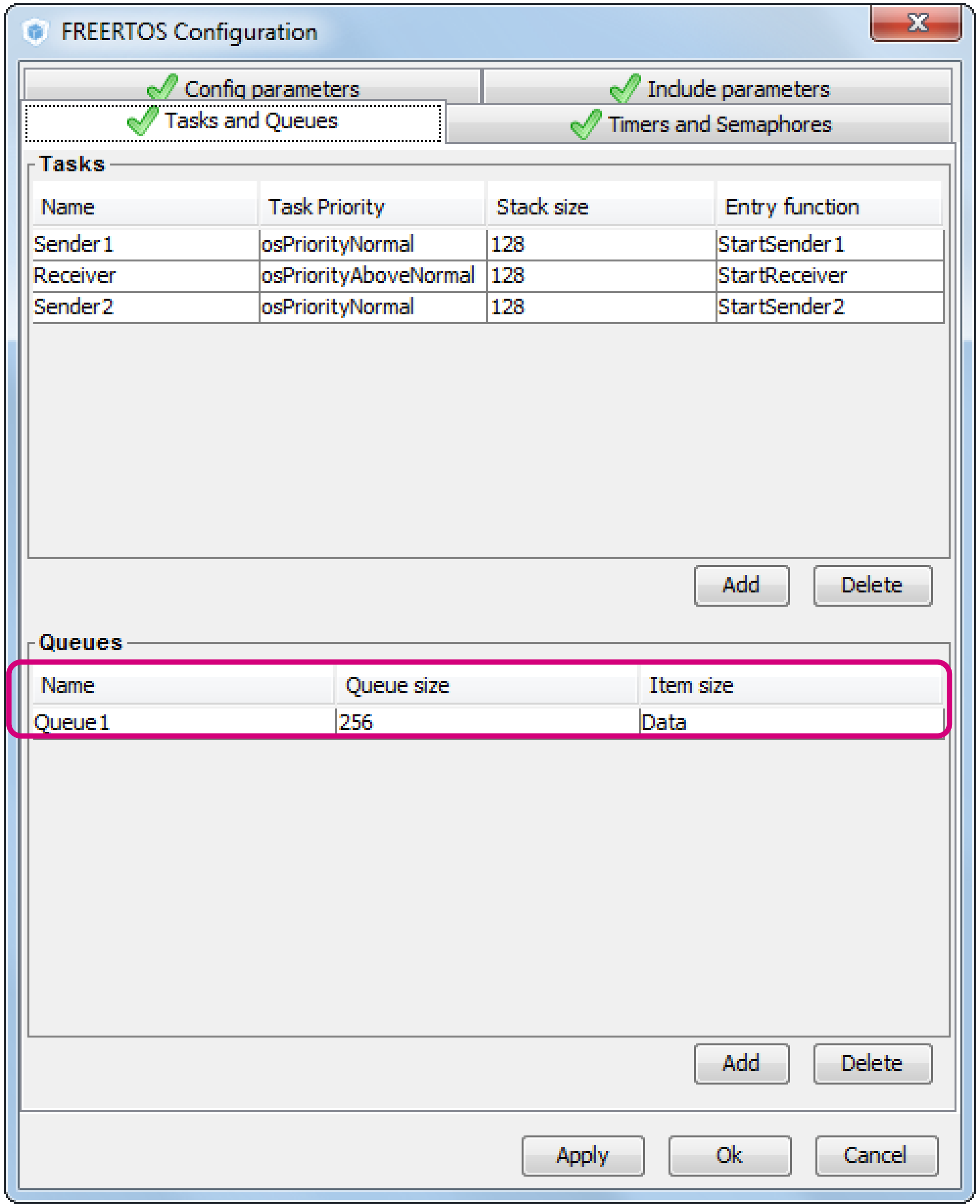
Name	Task Priority	Stack size	Entry function
Sender1	osPriorityNormal	128	StartSender1
Receiver	osPriorityAboveNormal	128	StartReceiver
Sender2	osPriorityNormal	128	StartSender2

Queues

Name	Queue size	Item size
Queue1	256	Data

Queue items lab

- Item size will be structure called Data



Tasks

Name	Task Priority	Stack size	Entry function
Sender1	osPriorityNormal	128	StartSender1
Receiver	osPriorityAboveNormal	128	StartReceiver
Sender2	osPriorityNormal	128	StartSender2

Queues

Name	Queue size	Item size
Queue1	256	Data

Queue items lab

- Create new structure type for data

```
/* Define the structure type that will be passed on the queue. */
typedef struct
{
  uint8_t Value;
  uint8_t Source;
} Data;
```

- Define Structures which will be sent from sender task

```
/* Declare two variables of type Data that will be passed on the queue. */
Data DataToSend1={10,1};
Data DataToSend2={20,2};
```

Queue items lab

- Sent data from Sender task

```
void StartSender1(void const * argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
    printf("Task1\n");
    osMessagePut(Queue1Handle,(uint32_t)&DataToSend1,200);
    printf("Task1 delay\n");
    osDelay(2000);
  }
  /* USER CODE END 5 */
}
```

Put data into queue

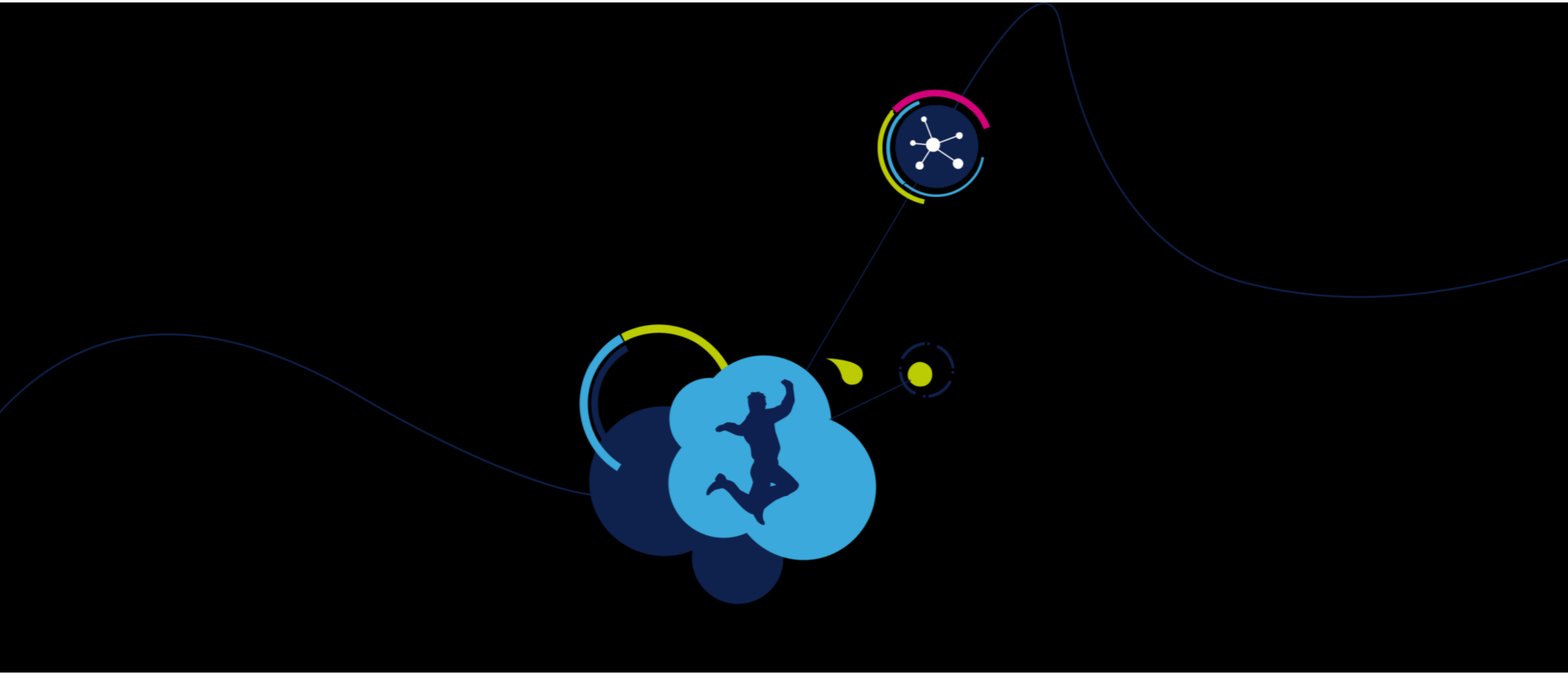
Queue items lab

- Receiver data from sender task

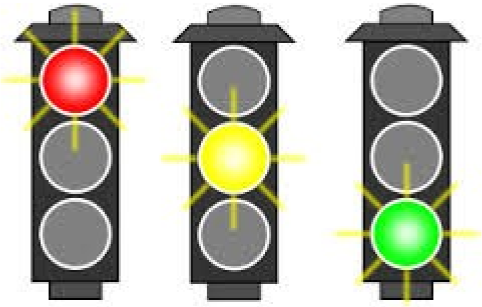
```
/* StartReceiver function */
void StartReceiver(void const * argument)
{
  /* USER CODE BEGIN StartReceiver */
  osEvent retvalue;
  /* Infinite loop */
  for(;;)
  {
    retvalue=osMessageGet(Queue1Handle,4000);
    if(((Data*)retvalue.value.p)->Source==1){
      printf("Receiver Receive message from Sender 1\n");
    }else{
      printf("Receiver Receive message from Sender 2\n");
    }
    printf("Data: %d \n",((Data*)retvalue.value.p)->Value);
  }
  /* USER CODE END StartReceiver */
}
```

Get data from queue

Decode data from osEvent structure



FreeRTOS Semaphores

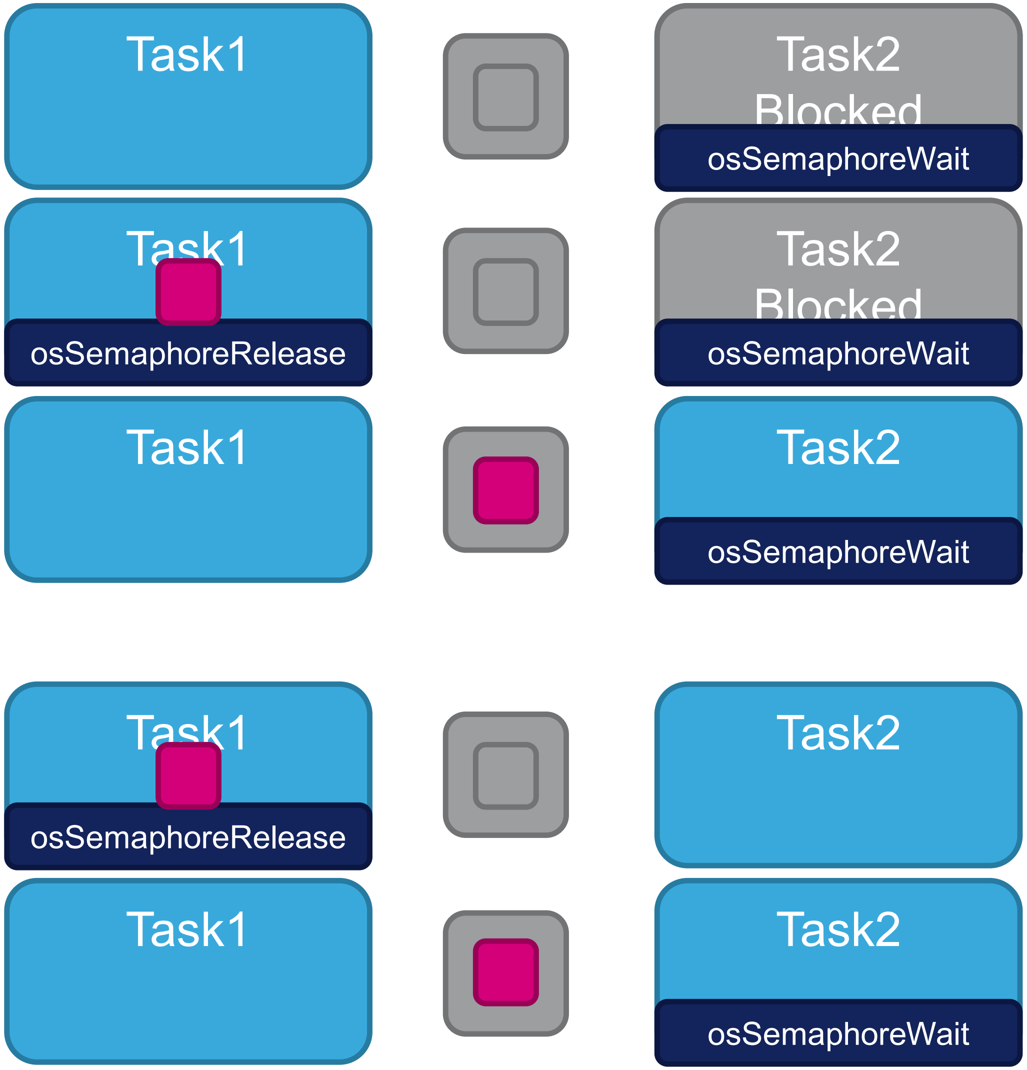


life.augmented

Semaphores

- Used for synchronization between
  - Tasks
  - Interrupt and task
- Two types
  - Binary semaphores
  - Counting semaphores
- Binary semaphore
  - Have only one ‘token’
  - Using to synchronize one action
- Counting semaphore
  - Have multiple ‘tokens’
  - Synchronize multiple actions

Binary Semaphore



Binary Semaphore

- Semaphore creation

```
osSemaphoreId osSemaphoreCreate (const osSemaphoreDef_t *semaphore_def, int32_t count)
```

Semaphore handle

Semaphore definition

Semaphore ‘tokens’
For binary semaphore
is 1

- Wait for Semaphore release

```
int32_t osSemaphoreWait (osSemaphoreId semaphore_id, uint32_t millisec)
```

Number of ‘tokens
in semaphore’

Semaphore handle

How long wait for
semaphore release

- Semaphore release

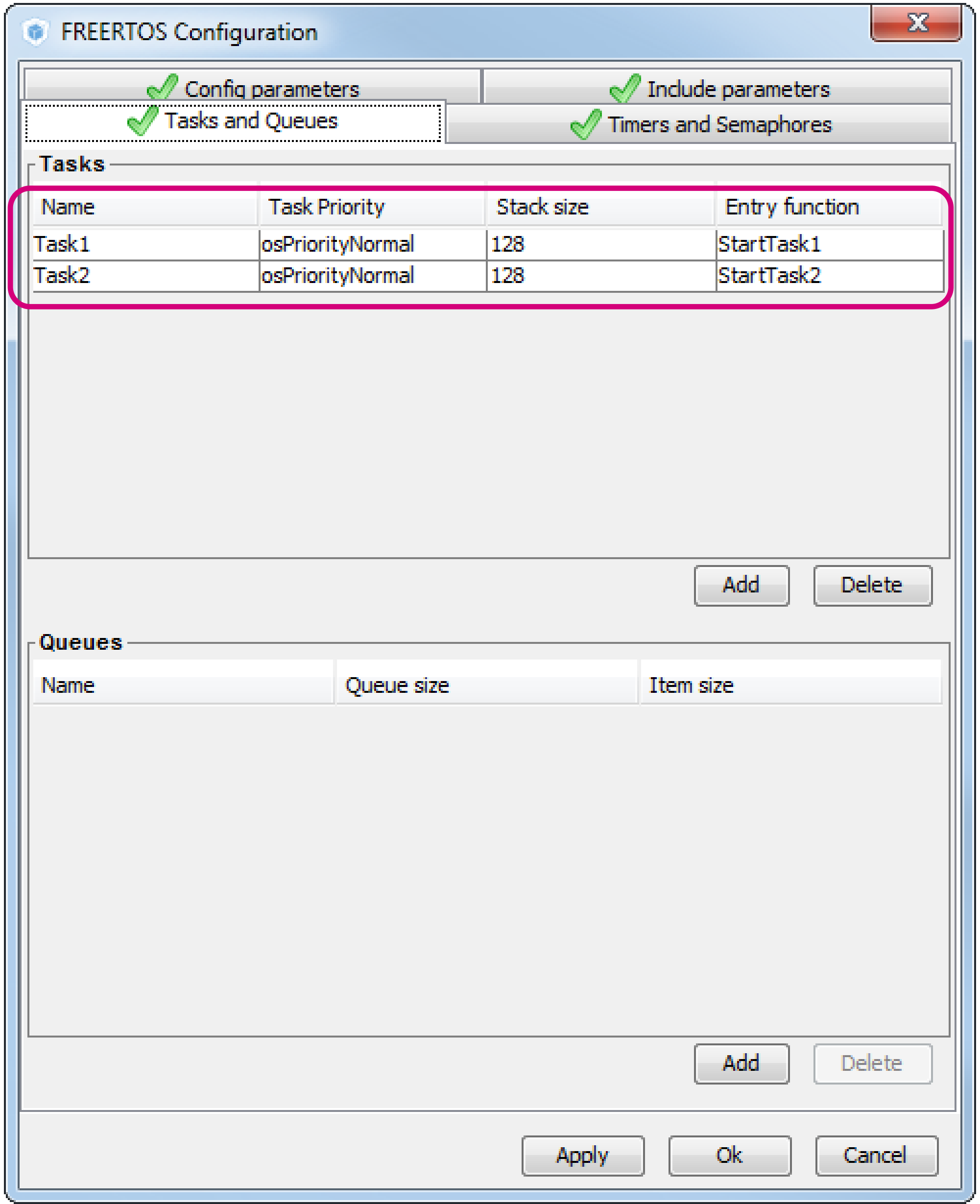
```
osStatus osSemaphoreRelease (osSemaphoreId semaphore_id)
```

Return status

Semaphore handle

Binary Semaphore lab

- Create two tasks
- With same priority
- Button Add
- Set parameters
- Button OK



FREERTOS Configuration

Tasks

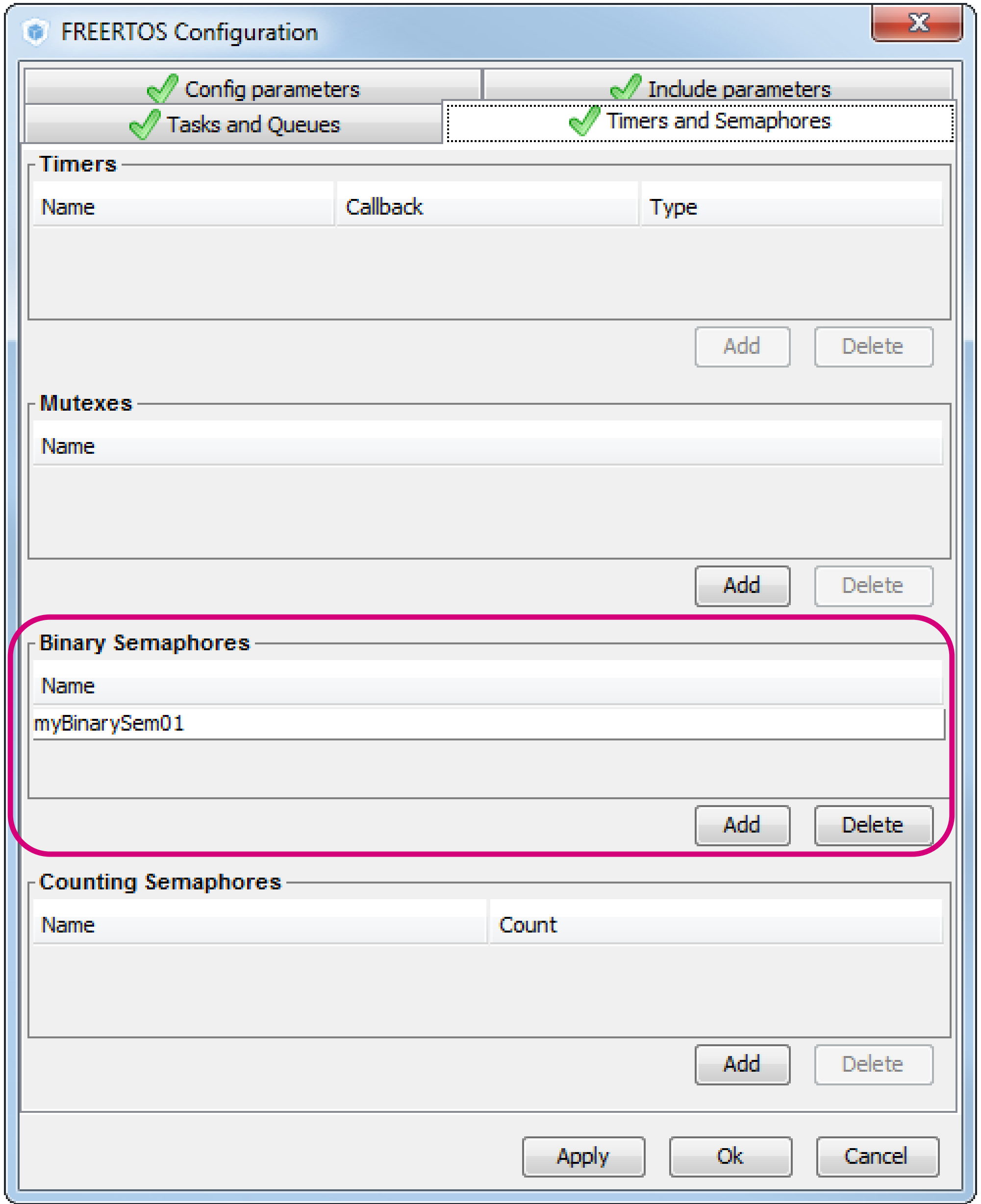
Name	Task Priority	Stack size	Entry function
Task1	osPriorityNormal	128	StartTask1
Task2	osPriorityNormal	128	StartTask2

Queues

Name	Queue size	Item size

Binary Semaphore lab

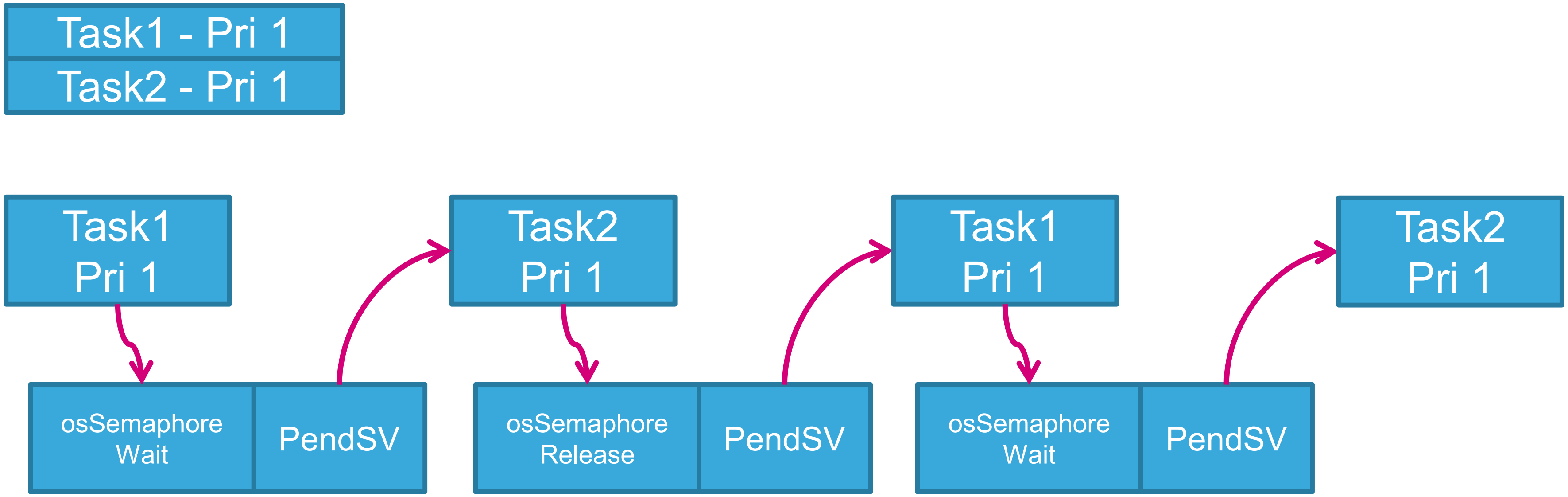
- Create binary binary semaphore
- Button Add
- Set name
- Button OK



myBinarySem01

Binary Semaphore lab

- Task1 is synchronized with Task2



Binary Semaphore lab

- Semaphore handle definition

```
/* Private variables ---------------------------------------------------------*/
osThreadId Task1Handle;
osThreadId Task2Handle;
osSemaphoreId myBinarySem01Handle;
```

- Semaphore creation

```
  /* Create the semaphores(s) */
  /* definition and creation of myBinarySem01 */
  osSemaphoreDef(myBinarySem01);
  myBinarySem01Handle = osSemaphoreCreate(osSemaphore(myBinarySem01), 1);
```

Binary Semaphore lab

- Semaphore use

- If tasks/interrupt is done the semaphore is released

```
void StartTask1(void const * argument)
{

  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
    osDelay(2000);
    printf("Task1 Release semaphore\n");
    osSemaphoreRelease(myBinarySem01Handle);
  }
  /* USER CODE END 5 */
}
```

Binary Semaphore lab

- Semaphore Wait

- Second task waits on semaphore release
  After releasetask is unbocked and continue in work

```
void StartTask2(void const * argument)
{
  /* USER CODE BEGIN StartTask2 */
  /* Infinite loop */
  for(;;)
  {
    osSemaphoreWait(myBinarySem01Handle,4000);
    printf("Task2 synchronized\n");
  }
  /* USER CODE END StartTask2 */
}
```

Counting semaphore

- Os API same as Binary semaphore

- Semaphore creation

```
osSemaphoreId osSemaphoreCreate (const osSemaphoreDef_t *semaphore_def, int32_t count)
```

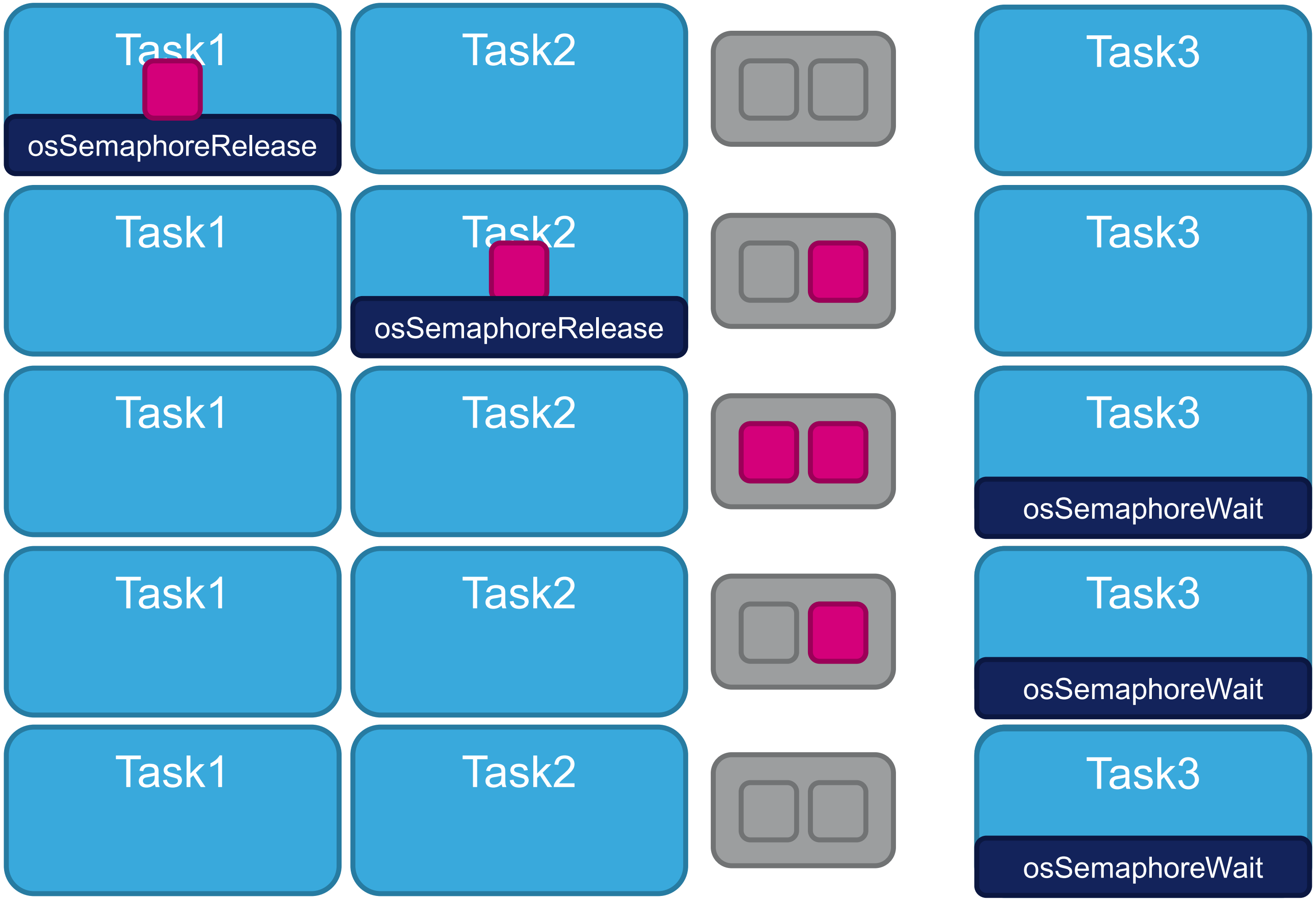
- Wait for Semaphore release

```
int32_t osSemaphoreWait (osSemaphoreId semaphore_id, uint32_t millisec)
```

- Semaphore release

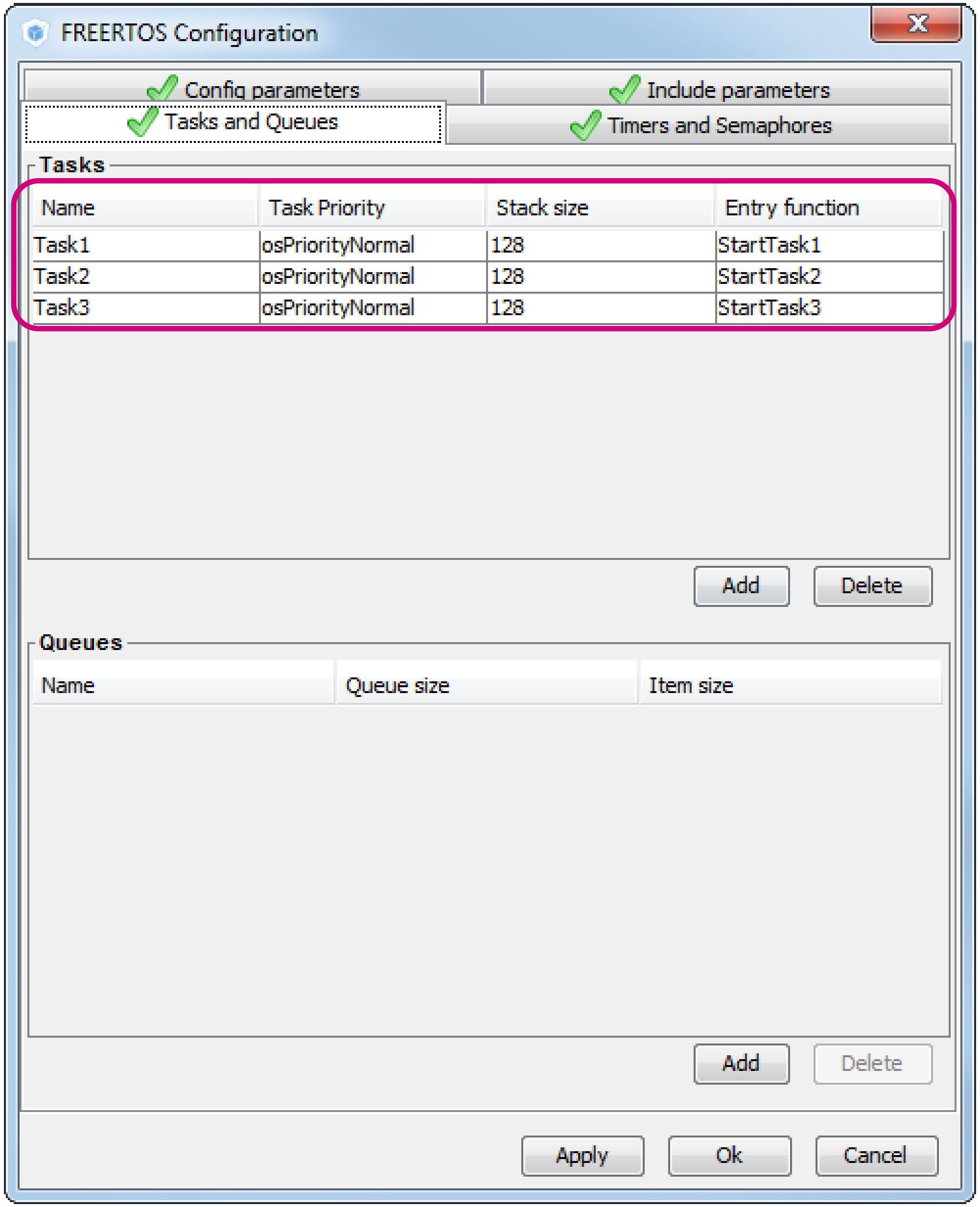
```
osStatus osSemaphoreRelease (osSemaphoreId semaphore_id)
```

Counting Semaphore



Counting Semaphore lab

- Create three tasks
- With same priority
- Button Add
- Set parameters
- Button OK



Tasks

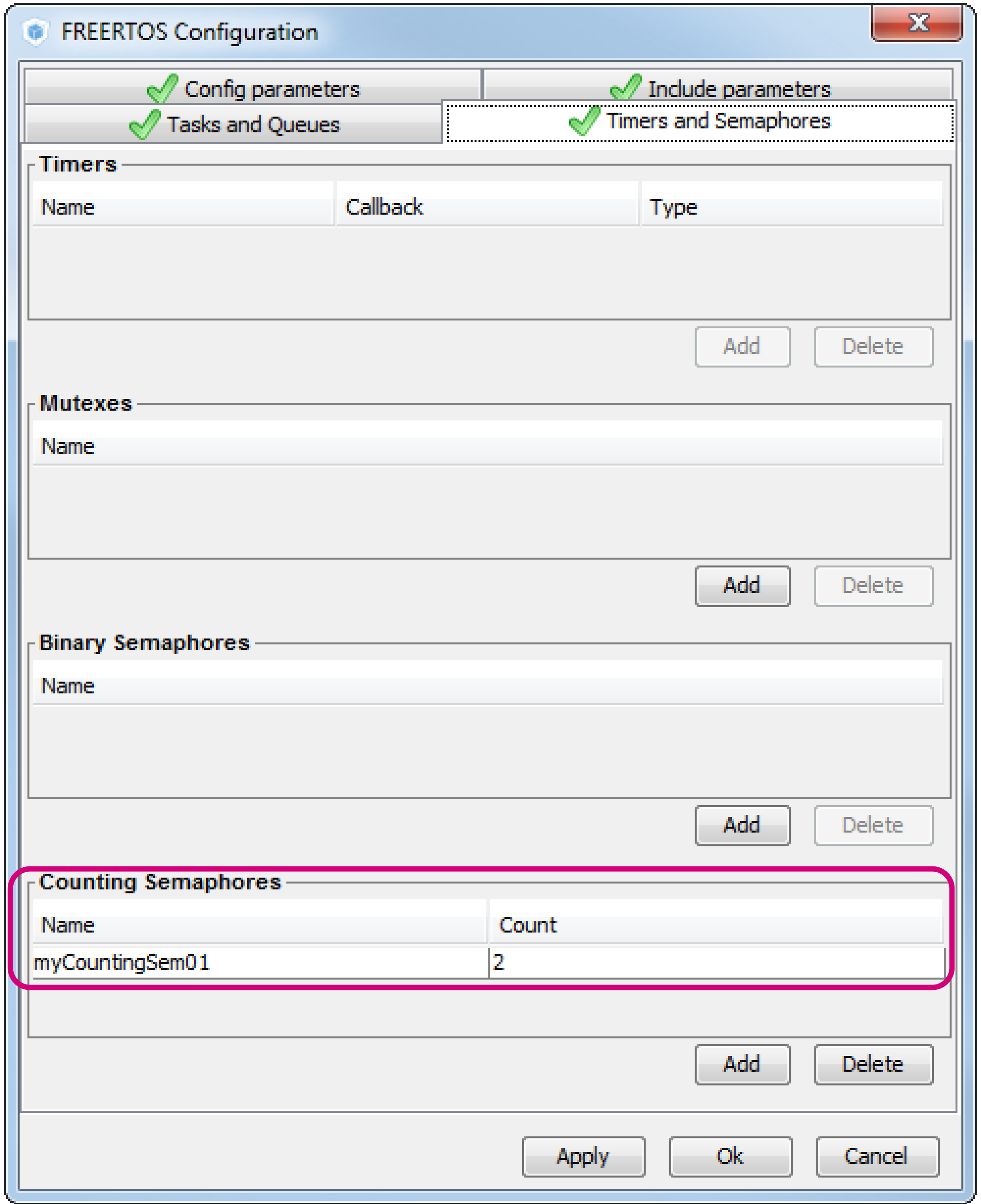
Name	Task Priority	Stack size	Entry function
Task1	osPriorityNormal	128	StartTask1
Task2	osPriorityNormal	128	StartTask2
Task3	osPriorityNormal	128	StartTask3

Queues

Name	Queue size	Item size

Counting Semaphore lab

- Create Counting semaphore
- Set count of tokens
- Button Add
- Set name
- Button OK



Name	Callback	Type

Name

Name

Name	Count
myCountingSem01	2

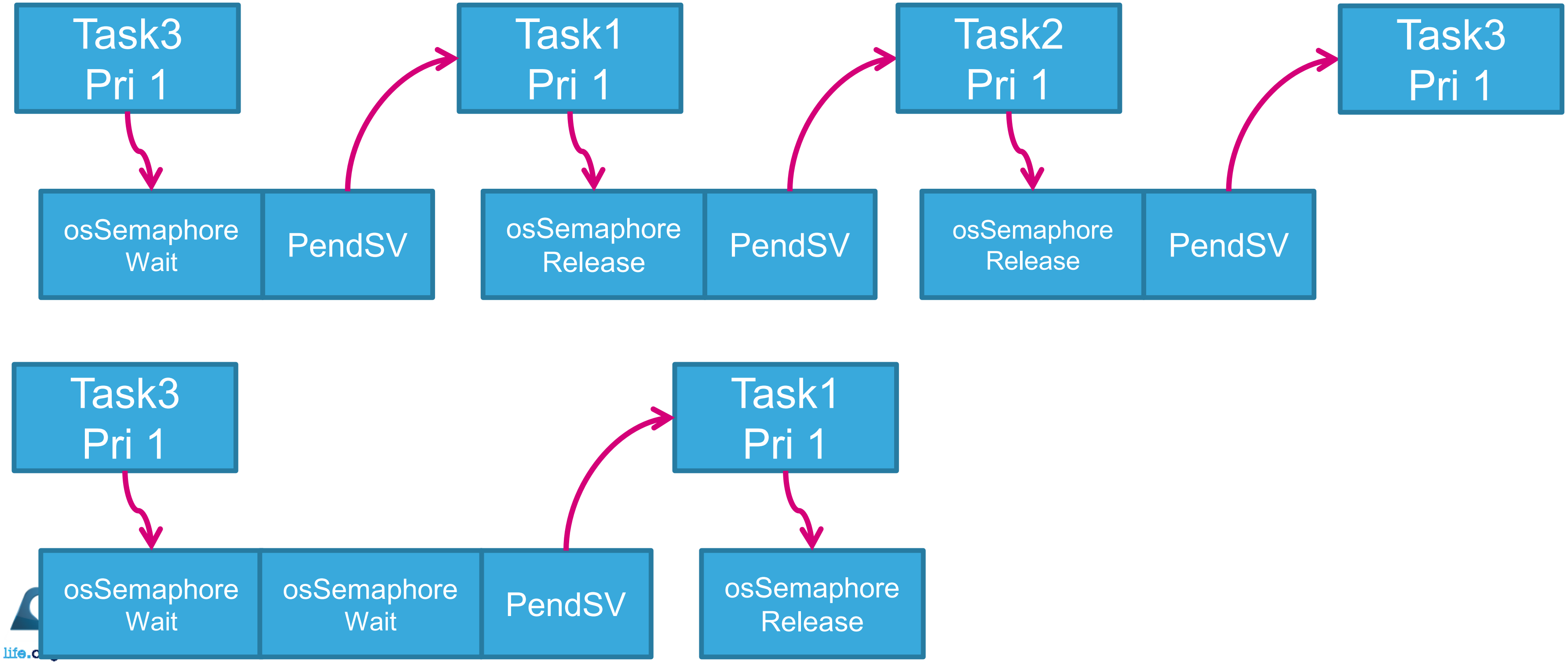
Counting Semaphore lab

- Task1 and Task2 release semaphore
- Task 3 wait for two tokens

Task1- Pri 1

Task2- Pri 1

Task3- Pri 1



Task3 Pri 1

osSemaphore Wait

PendSV

Task1 Pri 1

osSemaphore Release

PendSV

Task2 Pri 1

osSemaphore Release

PendSV

Task3 Pri 1

Task3 Pri 1

osSemaphore Wait

osSemaphore Wait

PendSV

Task1 Pri 1

osSemaphore Release

Counting Semaphore lab

- Create Counting semaphore

```
/* Create the semaphores(s) */
 /* definition and creation of myCountingSem01 */
 osSemaphoreDef(myCountingSem01);
 myCountingSem01Handle = osSemaphoreCreate(osSemaphore(myCountingSem01), 2);
```

- Task1 and Task2 will be same

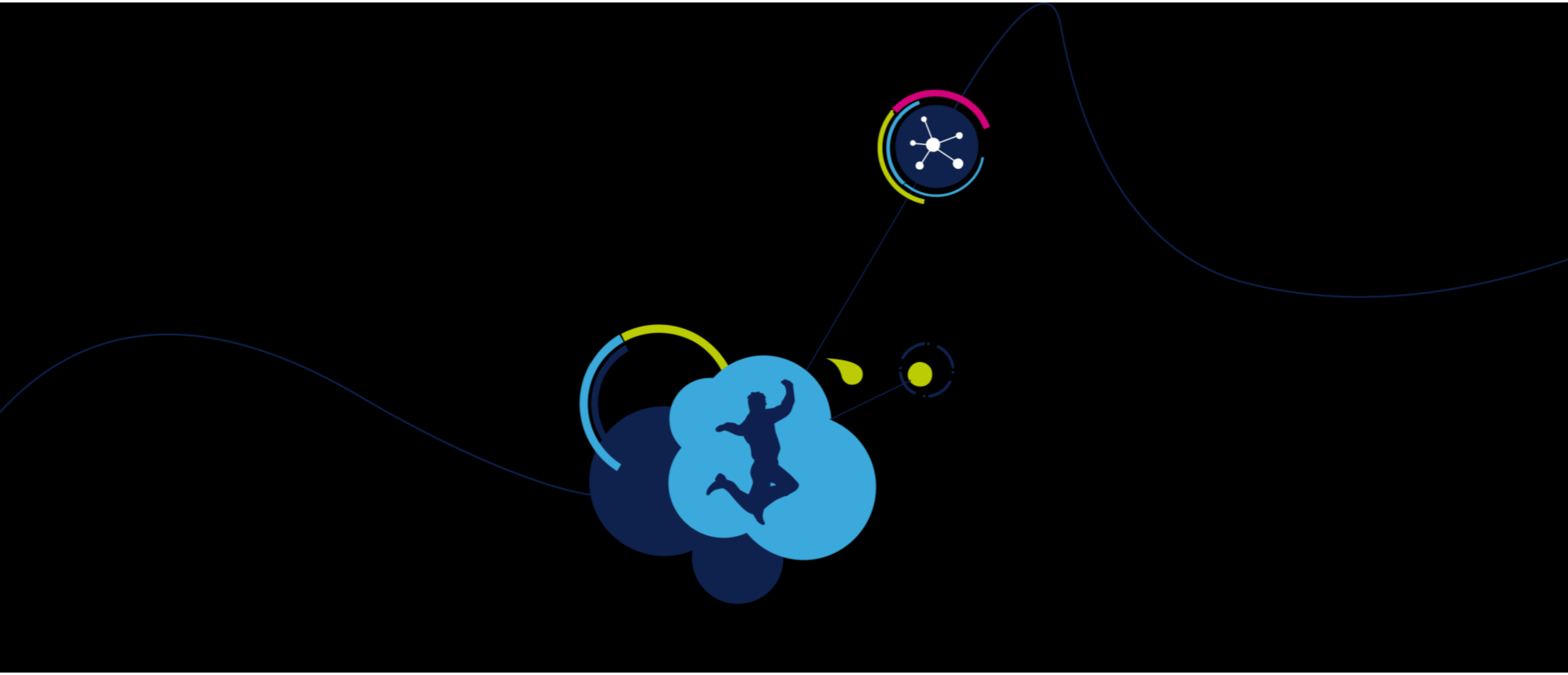
```
void StartTask1(void const * argument)
{

  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
    osDelay(2000);
    printf("Task1 Release counting semaphore\n");
    osSemaphoreRelease(myCountingSem01Handle);
  }
  /* USER CODE END 5 */
}
```

```
void StartTask2(void const * argument)
{
  /* USER CODE BEGIN StartTask2 */
  /* Infinite loop */
  for(;;)
  {
osDelay(2000);
printf("Task2 Release counting semaphore\
osSemaphoreRelease(myCountingSem01Handle)
  }
  /* USER CODE END StartTask2 */
}
```

Counting Semaphore lab

- Task3 will wait until semaphore will be 2 times released

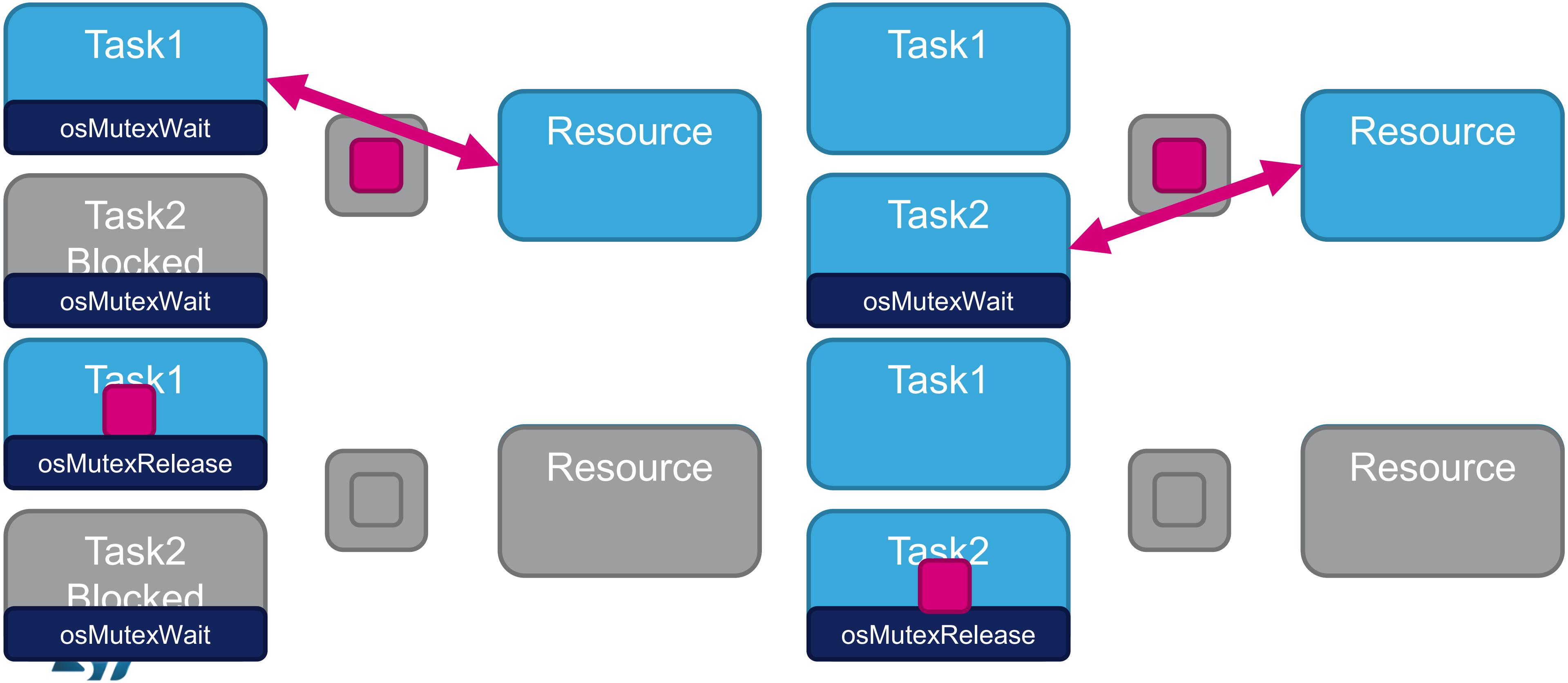
```
void StartTask3(void const * argument)
{
  /* USER CODE BEGIN StartTask3 */
  /* Infinite loop */
  for(;;)
  {
  osSemaphoreWait(myCountingSem01Handle, 4000);
  osSemaphoreWait(myCountingSem01Handle, 4000);
  printf("Task3 synchronized\n");
  }
  /* USER CODE END StartTask3 */
}
```



FreeRTOS
Mutex

Mutex

- Used to guard access to limited recourse
- Work very similar as Semaphores



Mutex

- Mutex creation

```
osMutexId osMutexCreate (const osMutexDef_t *mutex_def)
```

Mutex handle

Mutex definition

- Wait for Mutex release

```
osStatus osMutexWait (osMutexId mutex_id, uint32_t millisec)
```

Return status

Mutex handle

How long wait for mutex release

- Mutex release

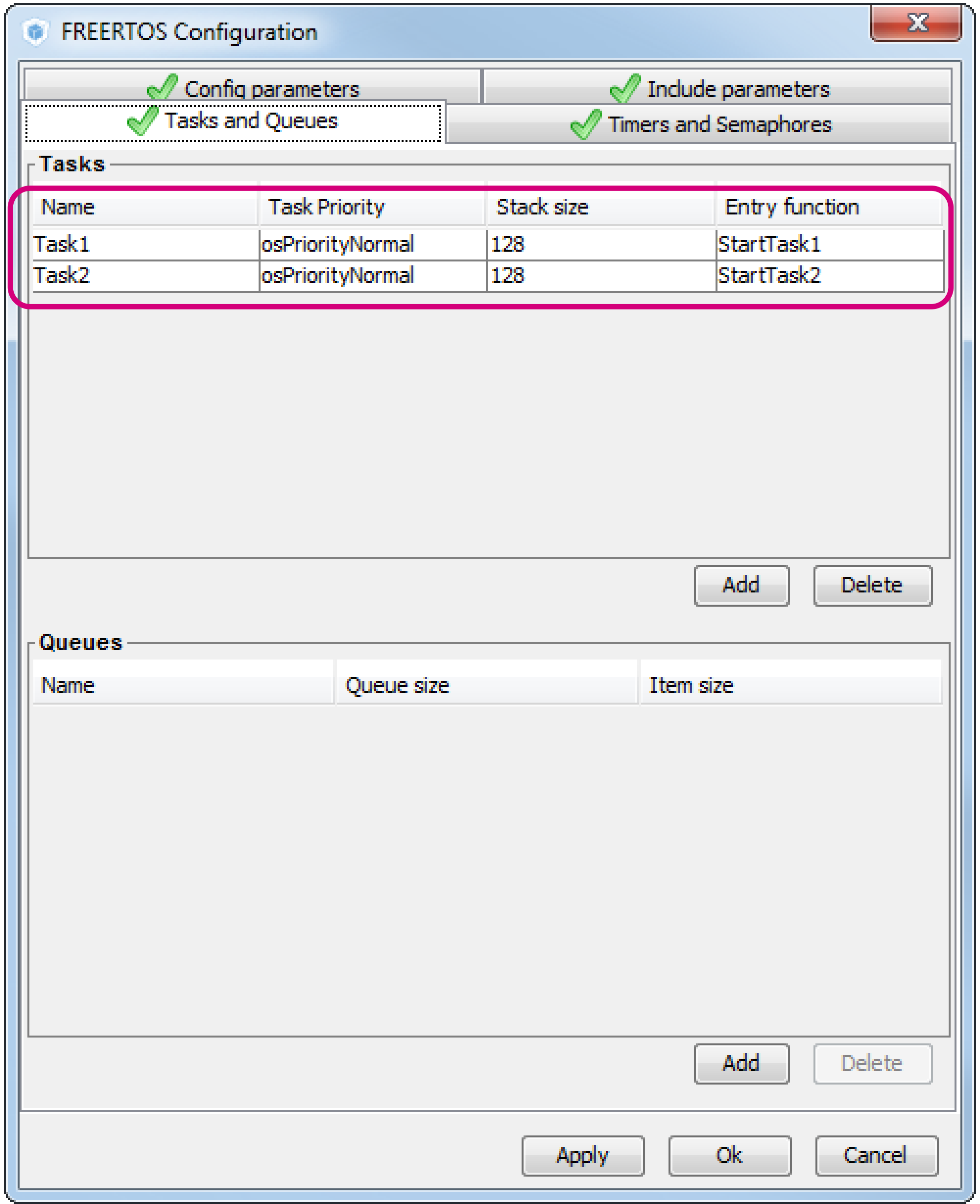
```
osStatus osMutexRelease (osMutexId mutex_id)
```

Return status

Mutex handle

Mutex lab

- Create two tasks
- With same priority
- Button Add
- Set parameters
- Button OK



FREERTOS Configuration

Tasks

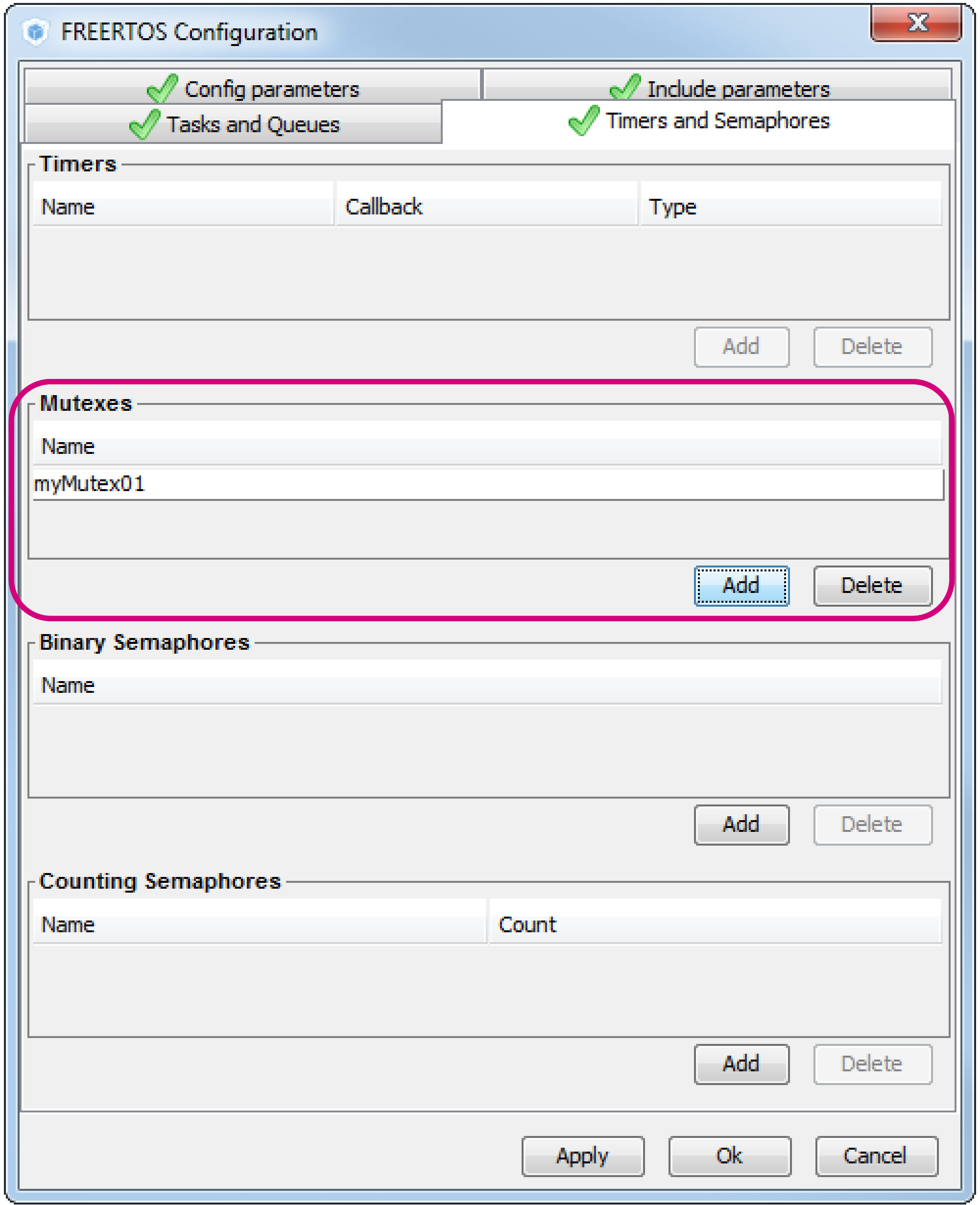
Name	Task Priority	Stack size	Entry function
Task1	osPriorityNormal	128	StartTask1
Task2	osPriorityNormal	128	StartTask2

Queues

Name	Queue size	Item size

Mutex lab

- Create Mutex
- Button Add
- Set name
- Button OK

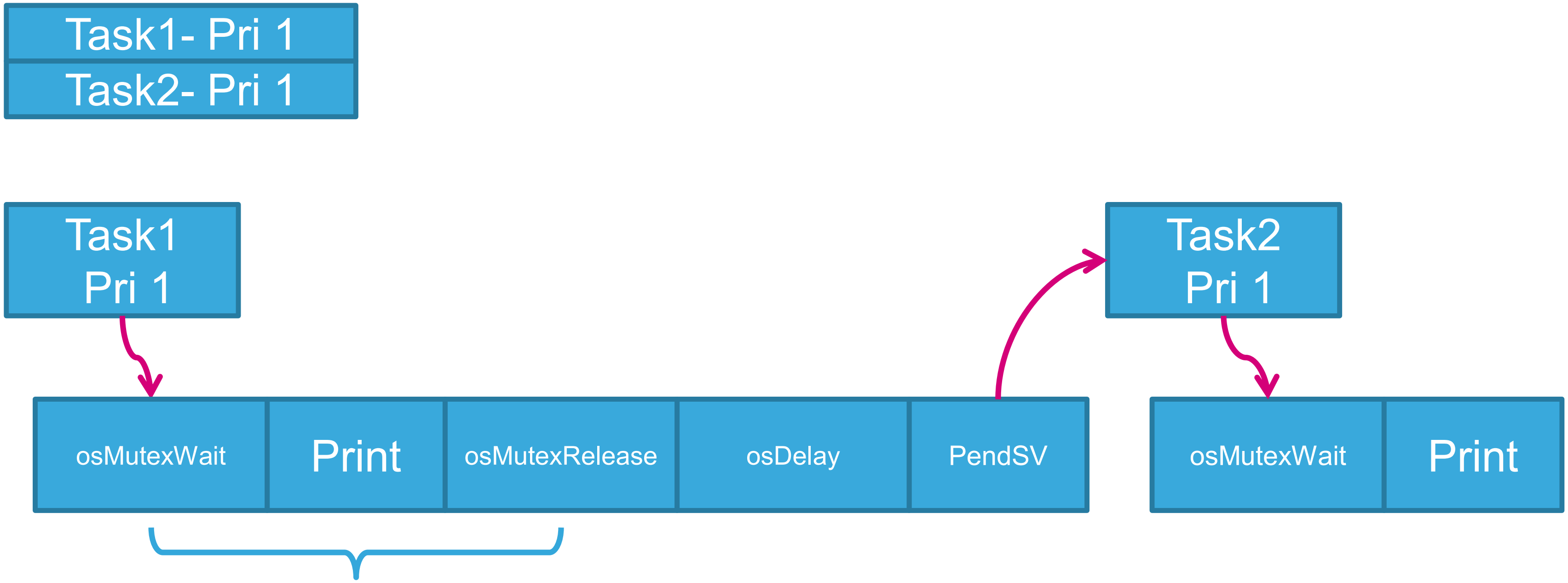


FREERTOS Configuration

myMutex01

Mutex lab

- Both tasks using printf function.
- Mutex is used to avoid collisions



Only one task can have semaphore

Mutex lab

- Mutex handle definition

```
/* Private variables ---------------------------------------------------------*/
osThreadId Task1Handle;
osThreadId Task2Handle;
osMutexId myMutex01Handle;
```

- Mutex creation

```
 /* Create the mutex(es) */
  /* definition and creation of myMutex01 */
  osMutexDef(myMutex01);
  myMutex01Handle = osMutexCreate(osMutex(myMutex01));
```

Mutex lab

- Semaphore use

- If tasks/interrupt is done the semaphore is released

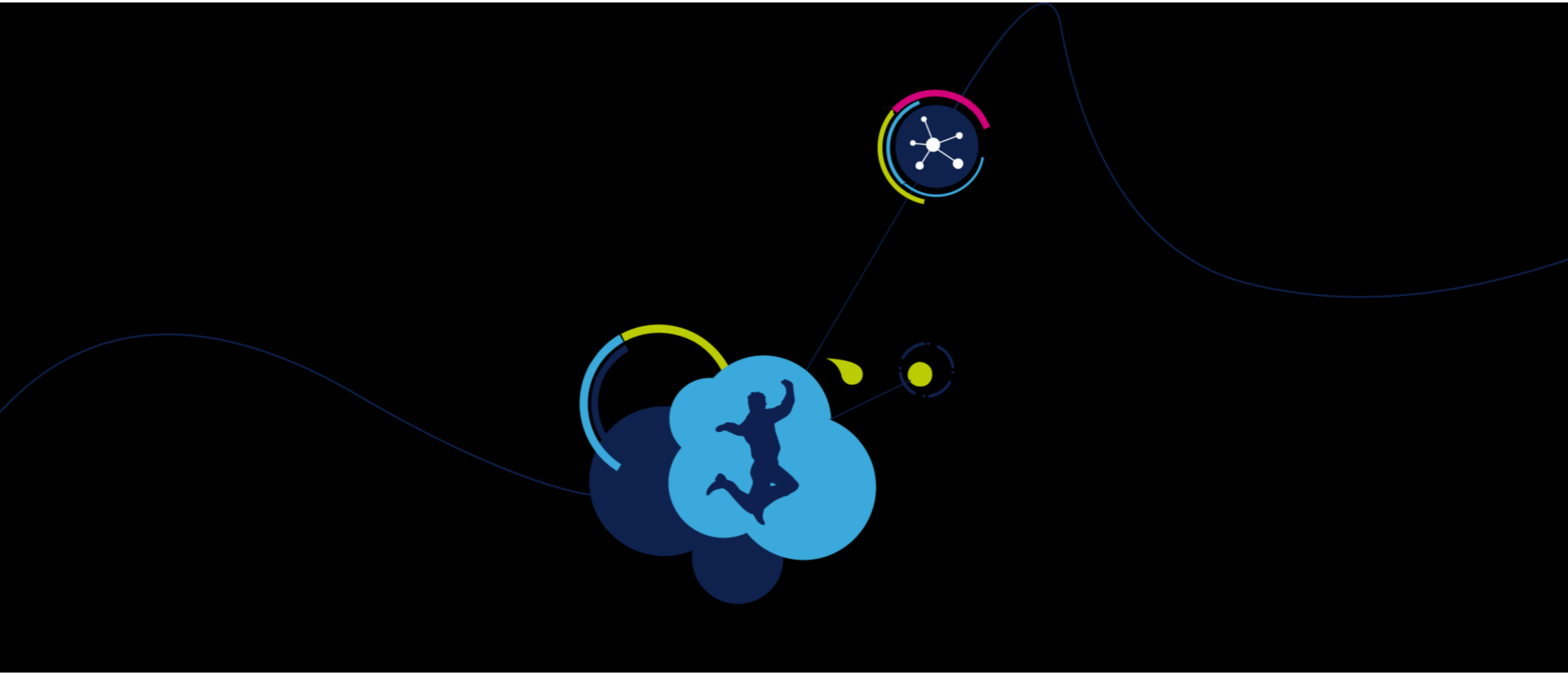
```
void StartTask1(void const * argument)
{

  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
    osDelay(2000);
    printf("Task1 Release semaphore\n");
    osSemaphoreRelease(myBinarySem01Handle);
  }
  /* USER CODE END 5 */
}
```

Mutex lab

- Task1 and Task2 using of Mutex

```
void StartTask1(void const * argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
    osDelay(2000);
    osMutexWait(myMutex01Handle,1000);
    printf("Task1 Print\n");
    osMutexRelease(myMutex01Handle);
  }
  /* USER CODE END 5 */
}
```

```
void StartTask2(void const * argument)
{
  /* USER CODE BEGIN StartTask2 */
  /* Infinite loop */
  for(;;)
  {
    osDelay(2000);
    osMutexWait(myMutex01Handle,1000);
    printf("Task2 Print\n");
    osMutexRelease(myMutex01Handle);
  }
  /* USER CODE END StartTask2 */
}
```

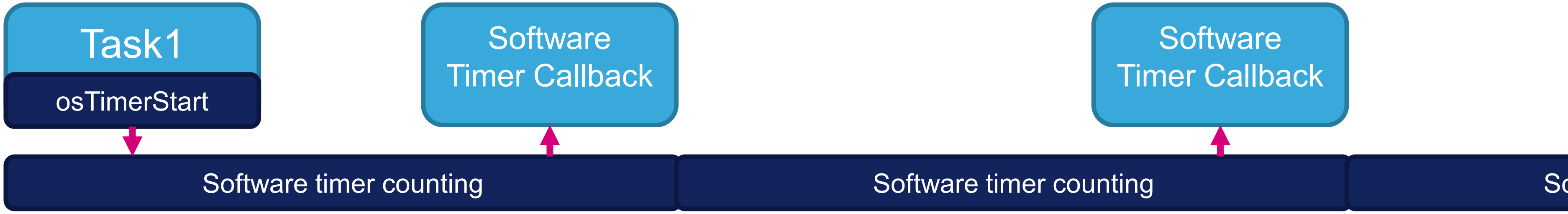


FreeRTOS Software Timers

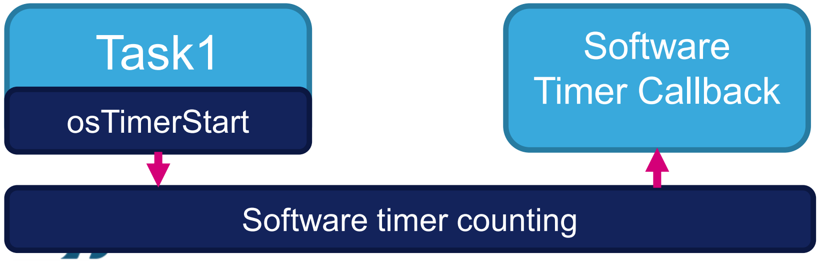
life.augmented

Software Timers

- Software timer is one of component in RTOS
- Can extend number of Timers in STM32
- Are not precise but can handle periodic actions or delay actions
- Two modes
  - Periodic

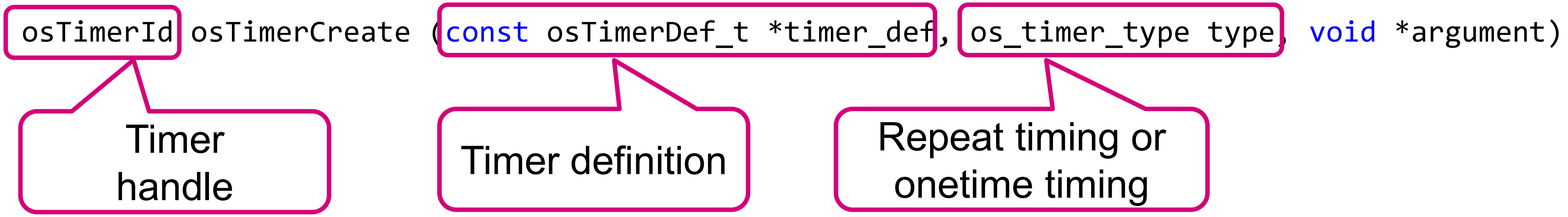


  - One Pulse



Software Timers

- Software timer creation



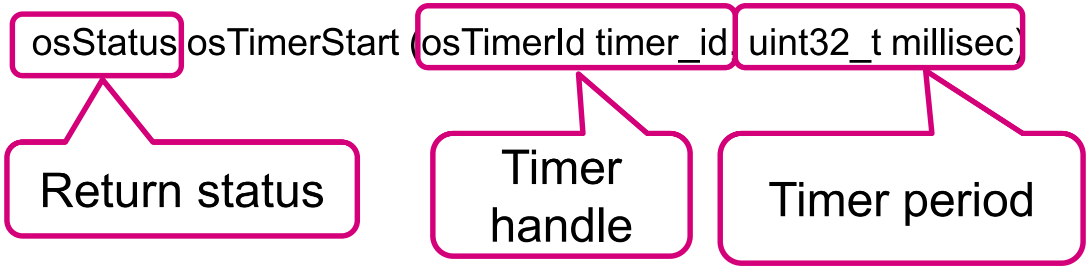
```
osTimerId osTimerCreate (const osTimerDef_t *timer_def, os_timer_type type, void *argument)
```

Timer handle

Timer definition

Repeat timing or onetime timing

- Software timer start



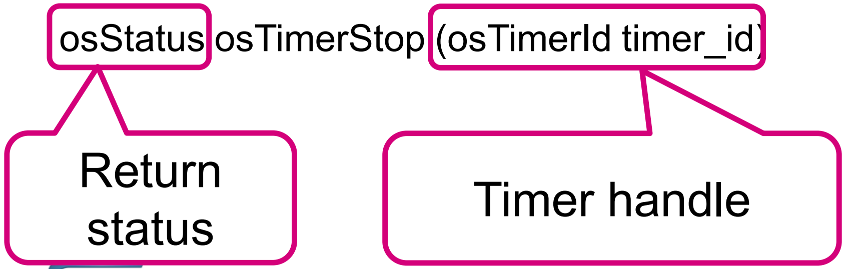
```
osStatus osTimerStart (osTimerId timer_id, uint32_t millisec)
```

Return status

Timer handle

Timer period

- Software timer stop



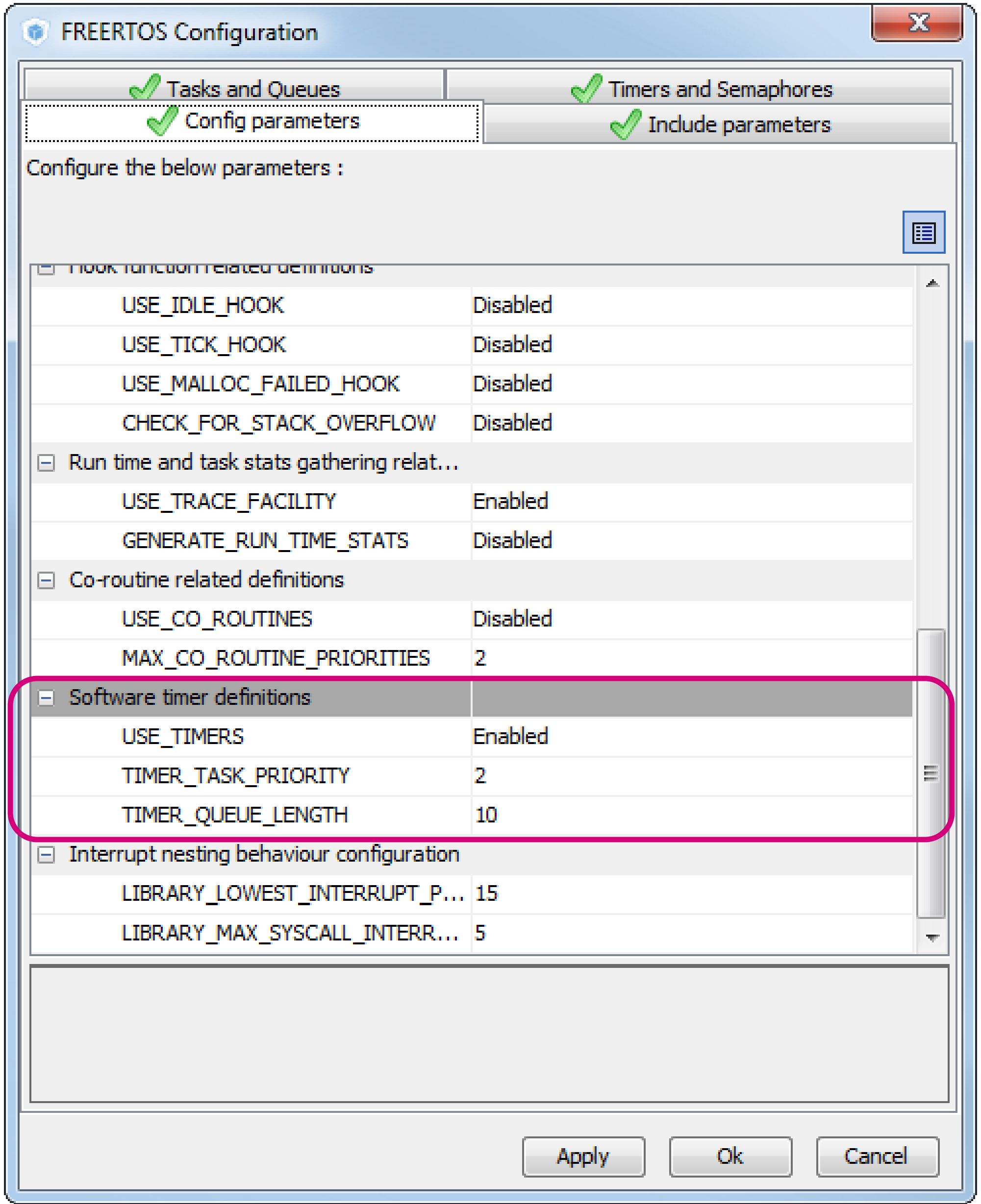
```
osStatus osTimerStop (osTimerId timer_id)
```

Return status

Timer handle

Software Timers lab

- Software timers are by default disabled in CubeMX
- FreeRTOS configuration
- Configuration TAB
- Enable Software timers

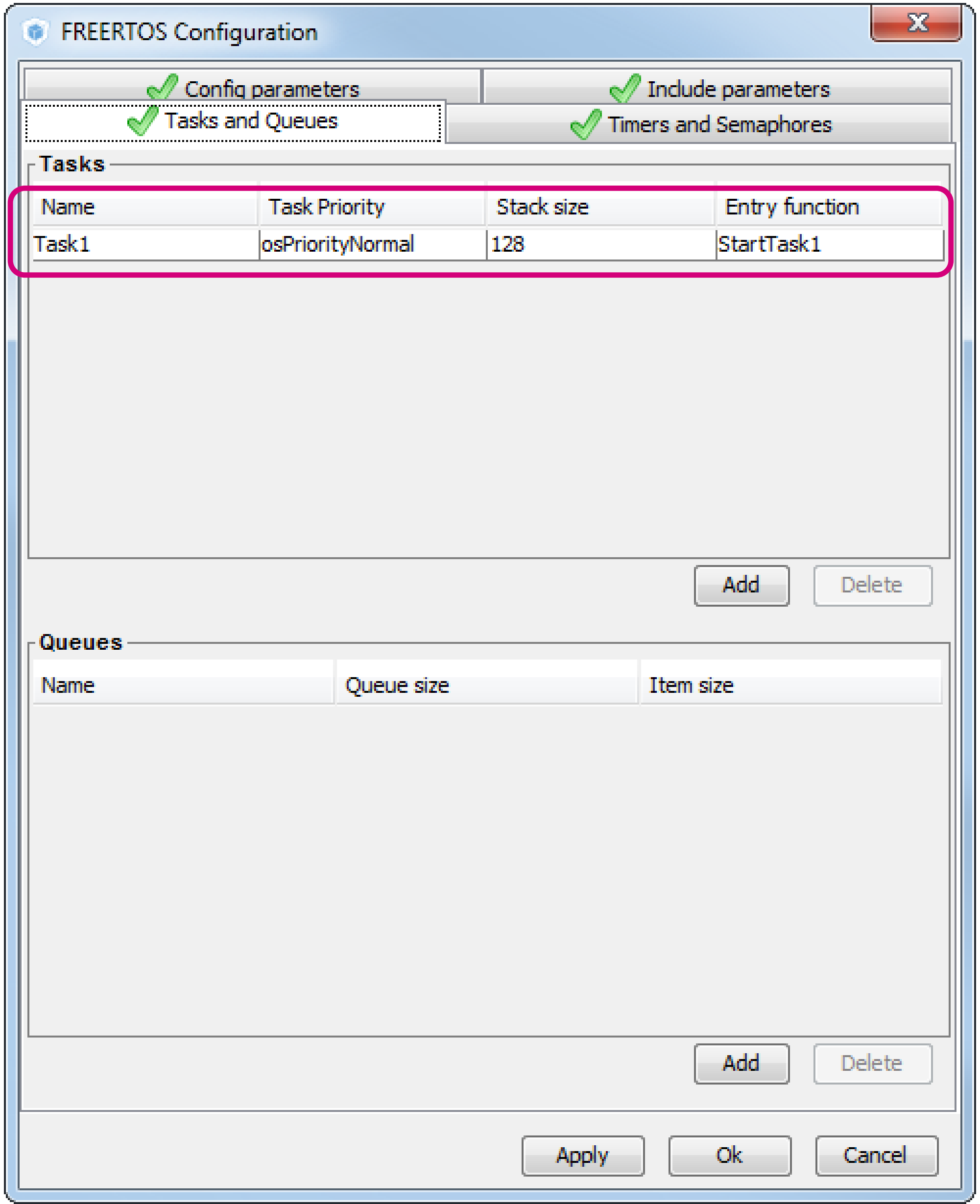


Configure the below parameters :

⊟ Hook function related definitions	
USE_IDLE_HOOK	Disabled
USE_TICK_HOOK	Disabled
USE_MALLOC_FAILED_HOOK	Disabled
CHECK_FOR_STACK_OVERFLOW	Disabled
⊟ Run time and task stats gathering relat…	
USE_TRACE_FACILITY	Enabled
GENERATE_RUN_TIME_STATS	Disabled
⊟ Co-routine related definitions	
USE_CO_ROUTINES	Disabled
MAX_CO_ROUTINE_PRIORITIES	2
⊟ Software timer definitions	
USE_TIMERS	Enabled
TIMER_TASK_PRIORITY	2
TIMER_QUEUE_LENGTH	10
⊟ Interrupt nesting behaviour configuration	
LIBRARY_LOWEST_INTERRUPT_P…	15
LIBRARY_MAX_SYSCALL_INTERR…	5

Software Timers lab

- Create one task
- Button Add
- Set parameters
- Set name
- Button OK



Tasks

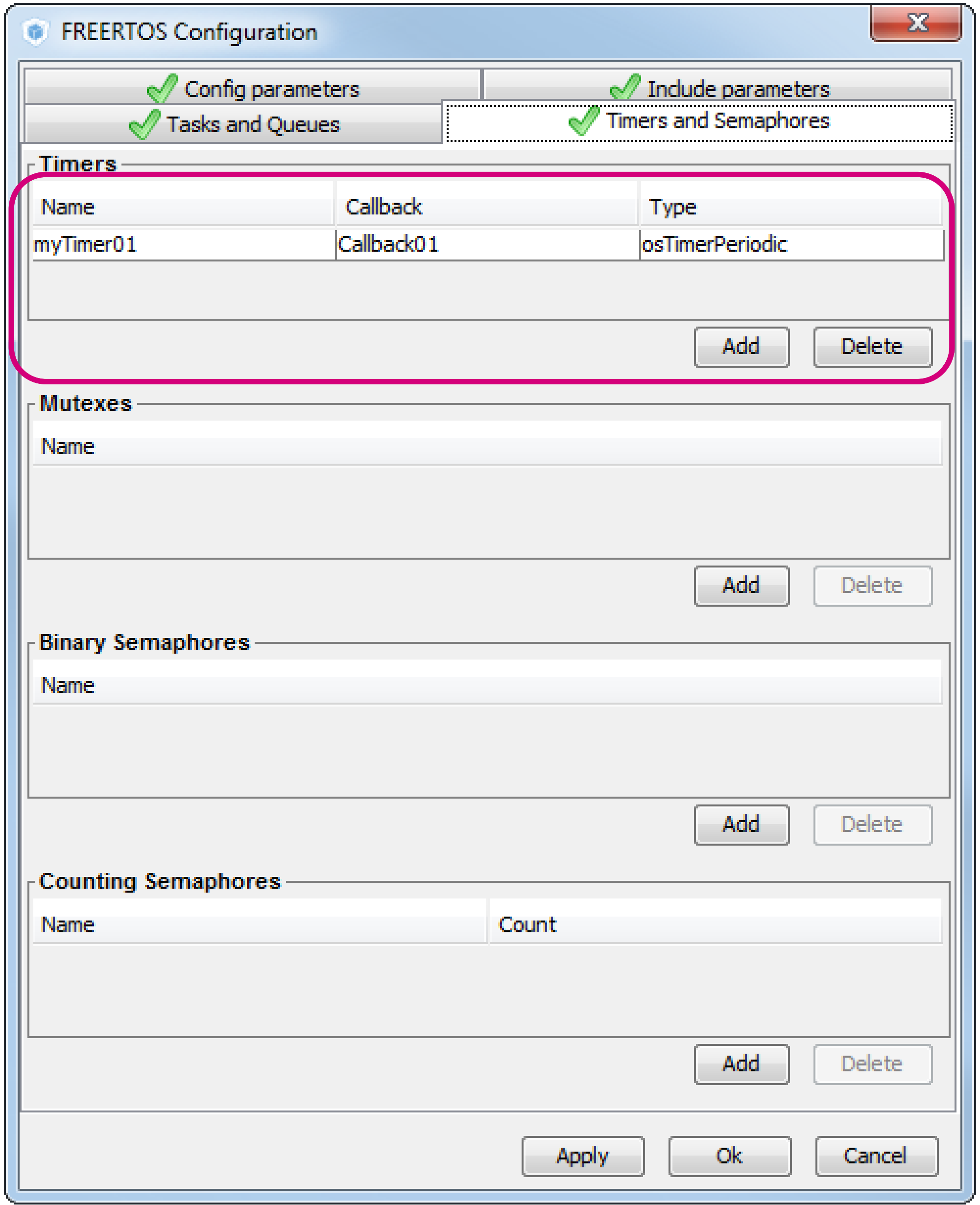
Name	Task Priority	Stack size	Entry function
Task1	osPriorityNormal	128	StartTask1

Queues

Name	Queue size	Item size

Software Timers lab

- Create Timer
- Button Add
- Set timer name
- Timer callback name
- Button OK



Name	Callback	Type
myTimer01	Callback01	osTimerPeriodic

Software Timers lab

- Software timer handle definition

```
/* Private variables ---------------------------------------------------------*/
osThreadId Task1Handle;
osTimerId myTimer01Handle;
```

- Software timer creation

```
 /* Create the timer(s) */
  /* definition and creation of myTimer01 */
  osTimerDef(myTimer01, Callback01);
  myTimer01Handle = osTimerCreate(osTimer(myTimer01), osTimerPeriodic, NULL);
```

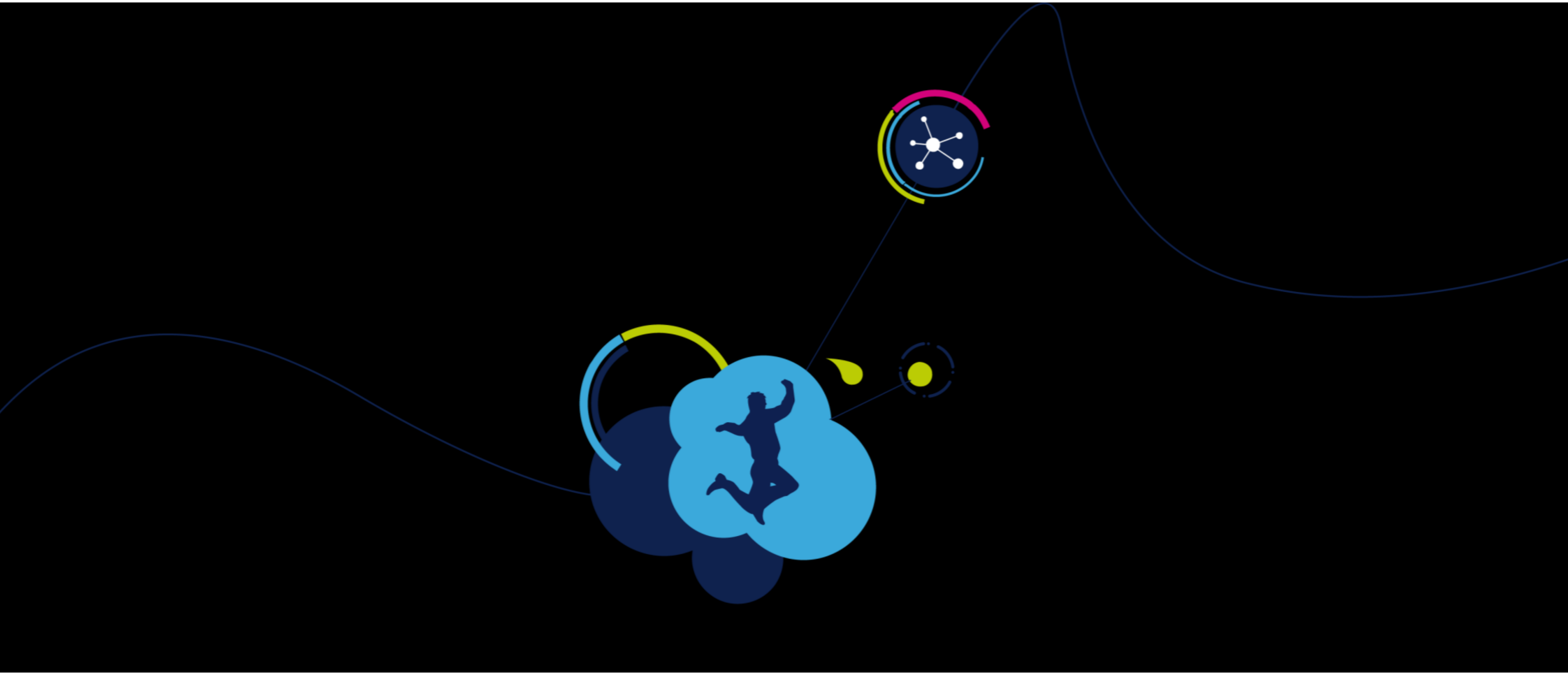
- Software timer start

```
void StartTask1(void const * argument)
{
  /* USER CODE BEGIN 5 */
  osTimerStart(myTimer01Handle,1000);
  /* Infinite loop */
  for(;;)
  {
    osDelay(2000);
    printf("Task1 Print\n");
  }
  /* USER CODE END 5 */
}
```

Software Timers lab

- Software timer callback

```
/* Callback01 function */
void Callback01(void const * argument)
{
  /* USER CODE BEGIN Callback01 */
  printf("Timer Print\n");
  /* USER CODE END Callback01 */
}
```



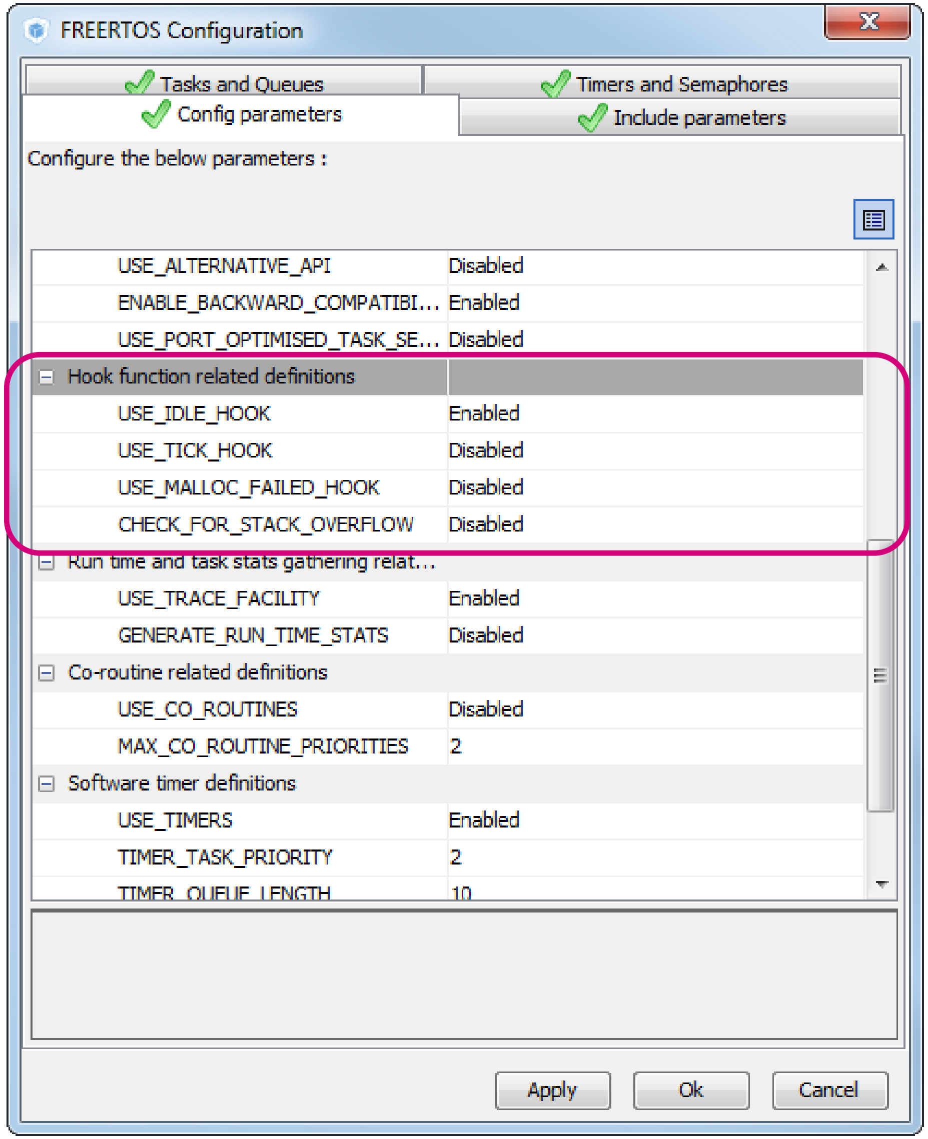
FreeRTOS advanced Hooks

Hooks

- Callbacks supported by FreeRTOS core
- Can help with FreeRTOS fault handling
- Type of hooks:
  - Idle Hool
  - Tick Hook
  - Malloc Failed Hook
  - Stack Overflow Hook
- CubeMX will create hooks in freertos.c file

Idle Hook

- If the scheduler cannot run any task it goes into idle mode
- Idle hook is callback from idle mode
- In to this task is possible to put power saving function



Configure the below parameters :

USE_ALTERNATIVE_API	Disabled
ENABLE_BACKWARD_COMPATIBI…	Enabled
USE_PORT_OPTIMISED_TASK_SE…	Disabled
⊟ Hook function related definitions	
USE_IDLE_HOOK	Enabled
USE_TICK_HOOK	Disabled
USE_MALLOC_FAILED_HOOK	Disabled
CHECK_FOR_STACK_OVERFLOW	Disabled
⊟ Run time and task stats gathering relat…	
USE_TRACE_FACILITY	Enabled
GENERATE_RUN_TIME_STATS	Disabled
⊟ Co-routine related definitions	
USE_CO_ROUTINES	Disabled
MAX_CO_ROUTINE_PRIORITIES	2
⊟ Software timer definitions	
USE_TIMERS	Enabled
TIMER_TASK_PRIORITY	2
TIMER_QUEUE_LENGTH	10

Idle Hook

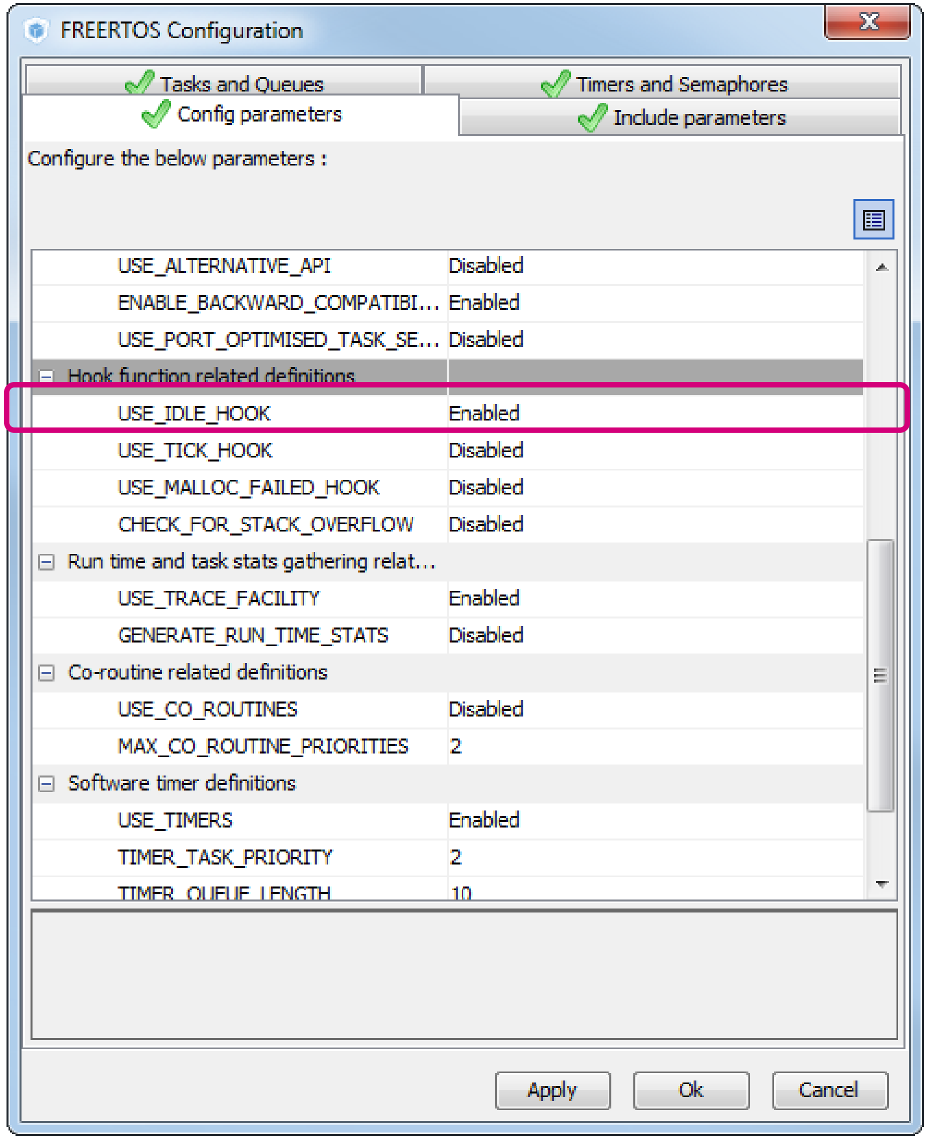
- Idle hook callback in freertos.c created by CubeMX

```
/* USER CODE END FunctionPrototypes */
/* Hook prototypes */
void vApplicationIdleHook(void);

/* USER CODE BEGIN 2 */
void vApplicationIdleHook( void )
{
   /* vApplicationIdleHook() will only be called if configUSE_IDLE_HOOK is set
   to 1 in FreeRTOSConfig.h. It will be called on each iteration of the idle
   task. It is essential that code added to this hook function never attempts
   to block in any way (for example, call xQueueReceive() with a block time
   specified, or call vTaskDelay()). If the application makes use of the
   vTaskDelete() API function (as this demo application does) then it is also
   important that vApplicationIdleHook() is permitted to return to its calling
   function, because it is the responsibility of the idle task to clean up
   memory allocated by the kernel to any task that has since been deleted. */
}
/* USER CODE END 2 */
```

- **Do not use blocking functons(osDelay, …) in this function or while(1)**

Idle Hook

- If the scheduler cannot run any task it goes into idle mode
- Idle hook is callback from idle mode
- In to this task is possible to put power saving function



Configure the below parameters :

USE_ALTERNATIVE_API	Disabled
ENABLE_BACKWARD_COMPATIBI…	Enabled
USE_PORT_OPTIMISED_TASK_SE…	Disabled
⊟ Hook function related definitions	
USE_IDLE_HOOK	Enabled
USE_TICK_HOOK	Disabled
USE_MALLOC_FAILED_HOOK	Disabled
CHECK_FOR_STACK_OVERFLOW	Disabled
⊟ Run time and task stats gathering relat…	
USE_TRACE_FACILITY	Enabled
GENERATE_RUN_TIME_STATS	Disabled
⊟ Co-routine related definitions	
USE_CO_ROUTINES	Disabled
MAX_CO_ROUTINE_PRIORITIES	2
⊟ Software timer definitions	
USE_TIMERS	Enabled
TIMER_TASK_PRIORITY	2
TIMER_QUEUE_LENGTH	10

Idle Hook

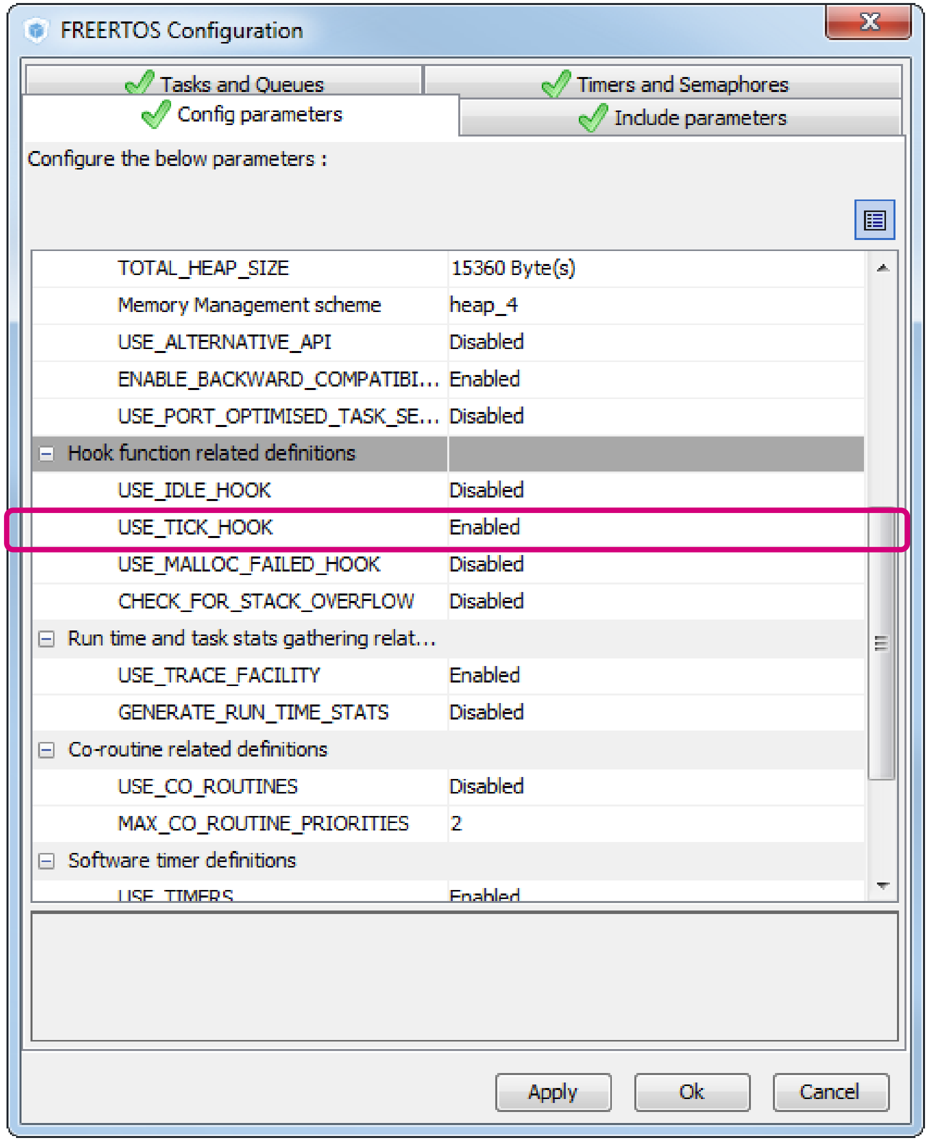
- Idle hook callback in freertos.c created by CubeMX

```
/* USER CODE END FunctionPrototypes */
/* Hook prototypes */
void vApplicationIdleHook(void);

/* USER CODE BEGIN 2 */
void vApplicationIdleHook( void )
{
   /* vApplicationIdleHook() will only be called if configUSE_IDLE_HOOK is set
   to 1 in FreeRTOSConfig.h. It will be called on each iteration of the idle
   task. It is essential that code added to this hook function never attempts
   to block in any way (for example, call xQueueReceive() with a block time
   specified, or call vTaskDelay()). If the application makes use of the
   vTaskDelete() API function (as this demo application does) then it is also
   important that vApplicationIdleHook() is permitted to return to its calling
   function, because it is the responsibility of the idle task to clean up
   memory allocated by the kernel to any task that has since been deleted. */
}
/* USER CODE END 2 */
```

- **Do not use blocking functons(osDelay, …) in this function or while(1)**

Tick Hook

- Every time the SysTick interrupt is trigger the TickHook is called
- Is possible use TickHook for periodic events like watchdog refresh



FREERTOS Configuration

Configure the below parameters :

TOTAL_HEAP_SIZE	15360 Byte(s)
Memory Management scheme	heap_4
USE_ALTERNATIVE_API	Disabled
ENABLE_BACKWARD_COMPATIBI...	Enabled
USE_PORT_OPTIMISED_TASK_SE...	Disabled
⊟ Hook function related definitions	
USE_IDLE_HOOK	Disabled
USE_TICK_HOOK	Enabled
USE_MALLOC_FAILED_HOOK	Disabled
CHECK_FOR_STACK_OVERFLOW	Disabled
⊟ Run time and task stats gathering relat...	
USE_TRACE_FACILITY	Enabled
GENERATE_RUN_TIME_STATS	Disabled
⊟ Co-routine related definitions	
USE_CO_ROUTINES	Disabled
MAX_CO_ROUTINE_PRIORITIES	2
⊟ Software timer definitions	
USE_TIMERS	Enabled

Tick Hook

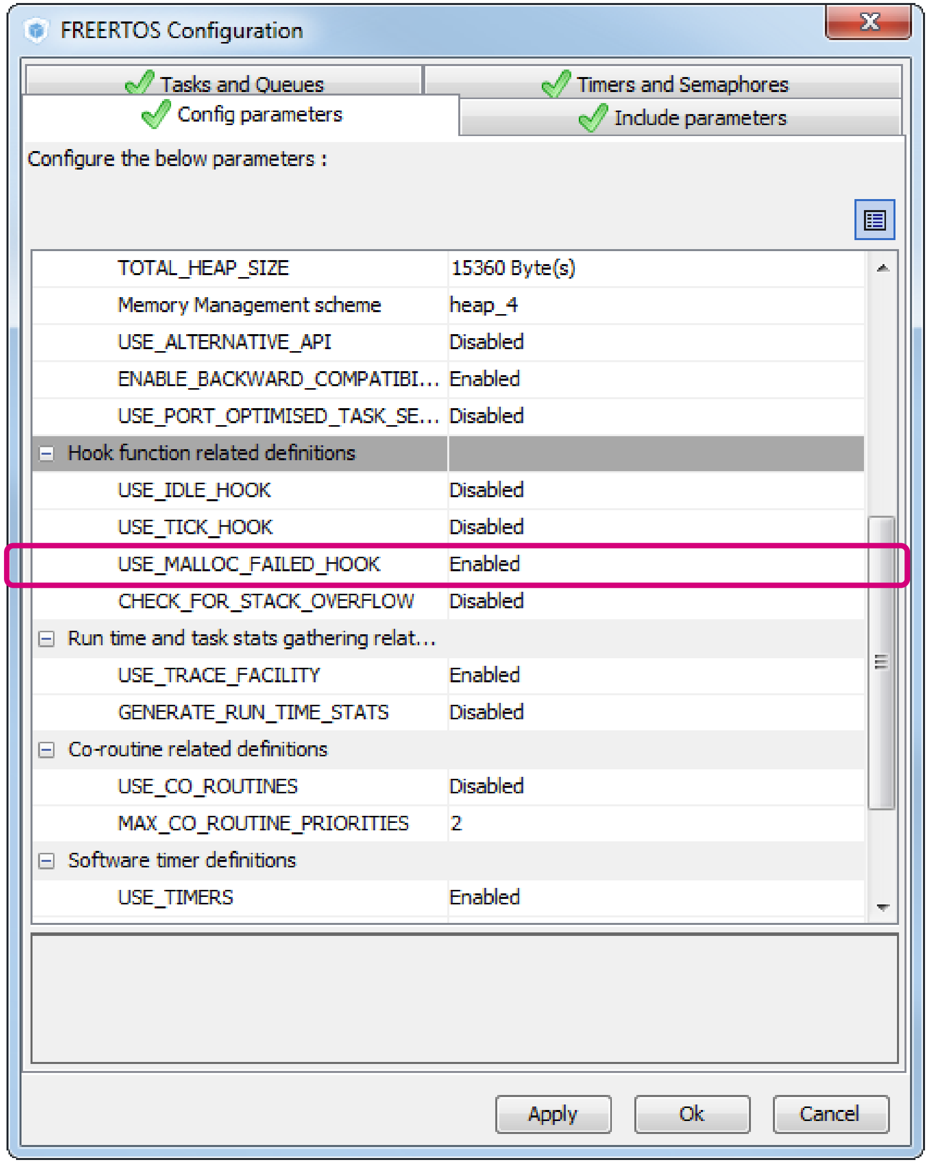
- Tick hook callback in freertos.c created by CubeMX

```
/* Hook prototypes */
void vApplicationTickHook(void);

/* USER CODE BEGIN 3 */
void vApplicationTickHook( void )
{
   /* This function will be called by each tick interrupt if
   configUSE_TICK_HOOK is set to 1 in FreeRTOSConfig.h. User code can be
   added here, but the tick hook is called from an interrupt context, so
   code must not attempt to block, and only the interrupt safe FreeRTOS API
   functions can be used (those that end in FromISR()). */
}
/* USER CODE END 3 */
```

- **Do not use blocking functons(osDelay, …) in this function or while(1)**

Malloc Failed Hook

- This callback is called if the memory allocation process fails
- Helps to react on malloc problems, when function return is not handelt



Configure the below parameters :

TOTAL_HEAP_SIZE	15360 Byte(s)
Memory Management scheme	heap_4
USE_ALTERNATIVE_API	Disabled
ENABLE_BACKWARD_COMPATIBI...	Enabled
USE_PORT_OPTIMISED_TASK_SE...	Disabled
⊟ Hook function related definitions	
USE_IDLE_HOOK	Disabled
USE_TICK_HOOK	Disabled
USE_MALLOC_FAILED_HOOK	Enabled
CHECK_FOR_STACK_OVERFLOW	Disabled
⊟ Run time and task stats gathering relat...	
USE_TRACE_FACILITY	Enabled
GENERATE_RUN_TIME_STATS	Disabled
⊟ Co-routine related definitions	
USE_CO_ROUTINES	Disabled
MAX_CO_ROUTINE_PRIORITIES	2
⊟ Software timer definitions	
USE_TIMERS	Enabled

Malloc Failed Hook

- Malloc Failed hook callback in freertos.c created by CubeMX

```
/* Hook prototypes */
void vApplicationMallocFailedHook(void);

/* USER CODE BEGIN 5 */
void vApplicationMallocFailedHook(void)
{
   /* vApplicationMallocFailedHook() will only be called if
   configUSE_MALLOC_FAILED_HOOK is set to 1 in FreeRTOSConfig.h. It is a hook
   function that will get called if a call to pvPortMalloc() fails.
   pvPortMalloc() is called internally by the kernel whenever a task, queue,
   timer or semaphore is created. It is also called by various parts of the
   demo application. If heap_1.c or heap_2.c are used, then the size of the
   heap available to pvPortMalloc() is defined by configTOTAL_HEAP_SIZE in
   FreeRTOSConfig.h, and the xPortGetFreeHeapSize() API function can be used
   to query the size of free heap space that remains (although it does not
   provide information on how the remaining heap might be fragmented). */
}
/* USER CODE END 5 */
```

- **Do not use blocking functons(osDelay, …) in this function or while(1)**

Malloc Failed Hook

- Malloc fail test

- Malloc fail hook:

```
/* USER CODE BEGIN 5 */
void vApplicationMallocFailedHook(void)
{
  printf("malloc fails\n");
}
/* USER CODE END 5 */
```

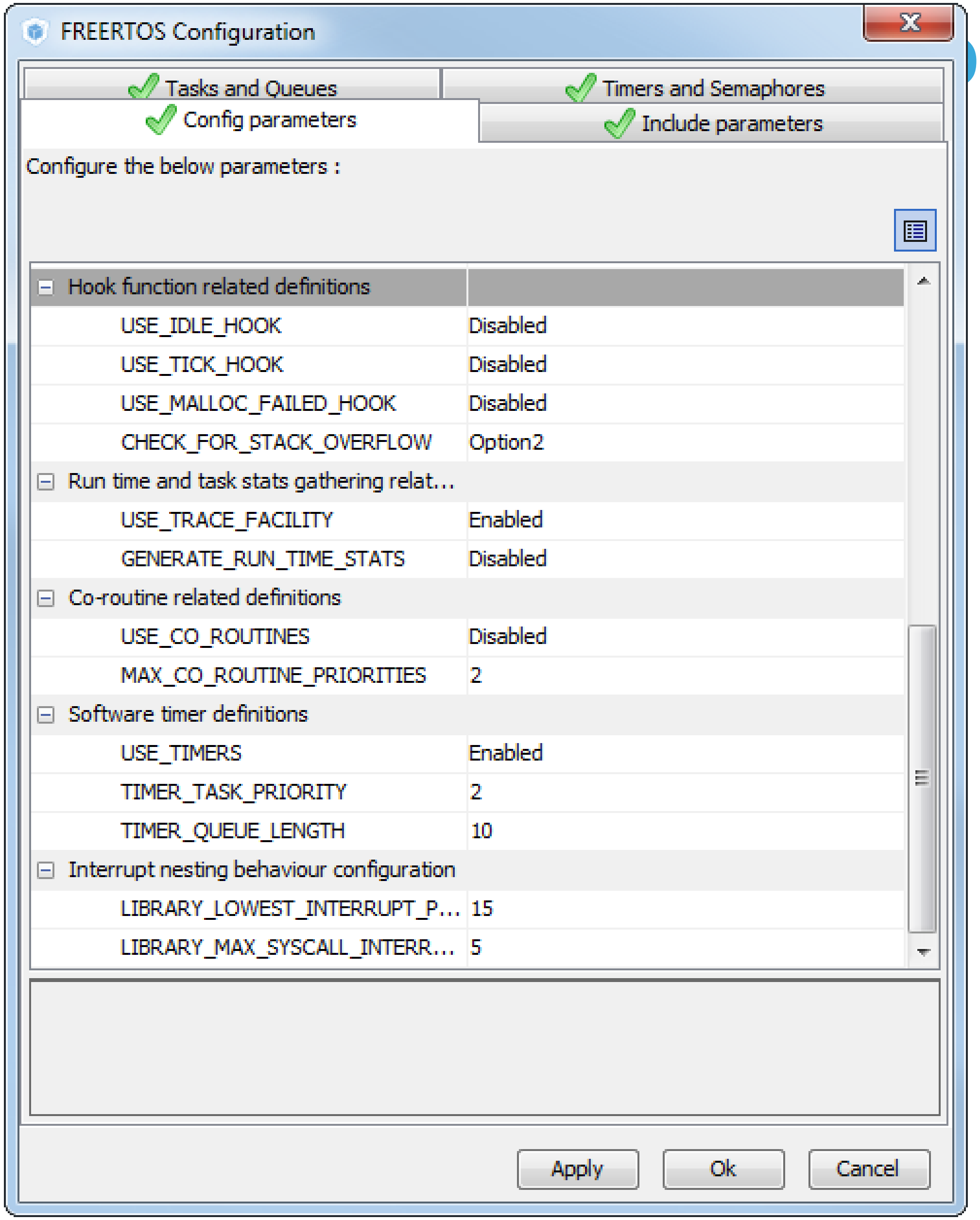
- Do impossible memory allocation

```
void StartTask1(void const * argument)
{
  /* USER CODE BEGIN 5 */
  osPoolDef(Memory,0x10000000,uint8_t);
  /* Infinite loop */
  for(;;)
  {
    PoolHandle = osPoolCreate(osPool(Memory));
    osDelay(5000);
  }
  /* USER CODE END 5 */
}
```

```
/* Private variables --------------------
osThreadId Task1Handle;
osPoolId PoolHandle;
```

Impossible memory allocation

w hook

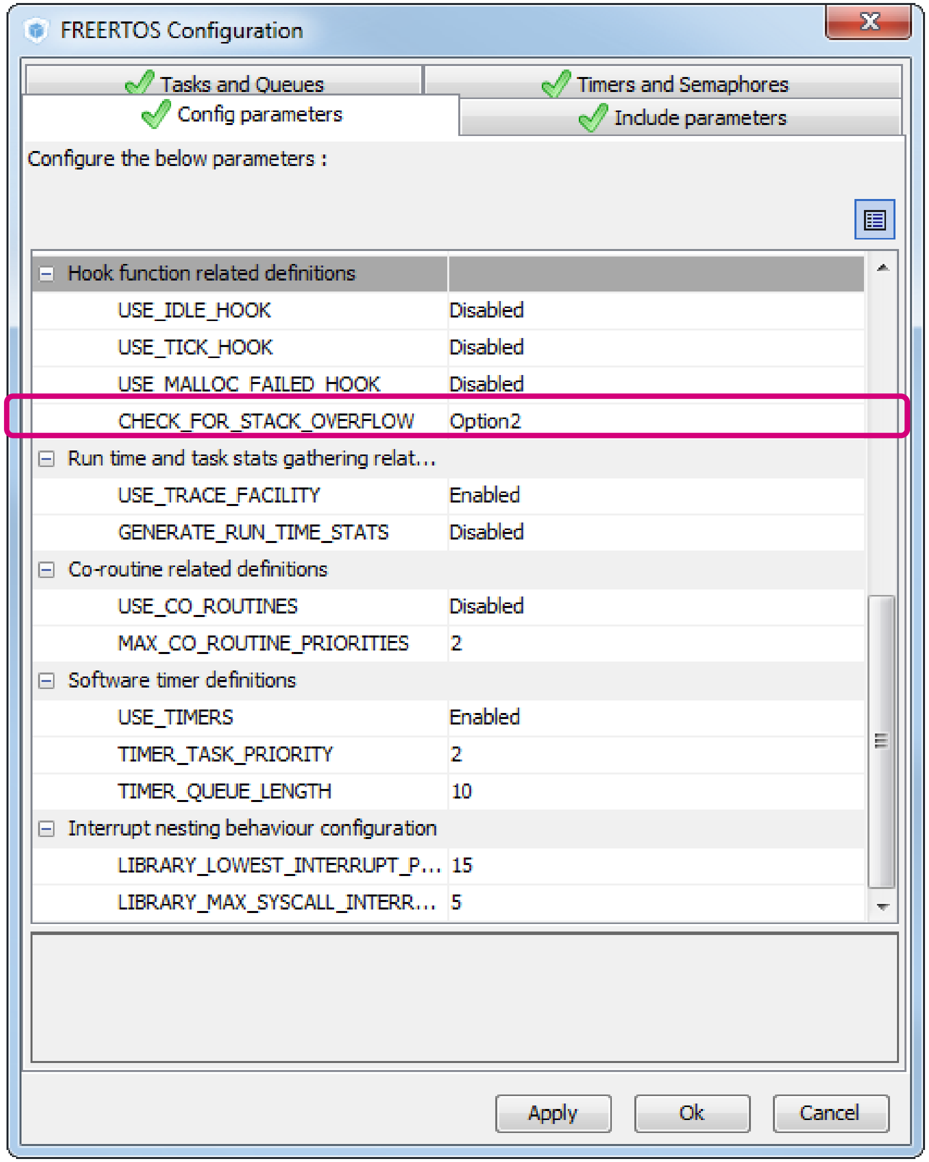


Configure the below parameters :

Hook function related definitions	
USE_IDLE_HOOK	Disabled
USE_TICK_HOOK	Disabled
USE_MALLOC_FAILED_HOOK	Disabled
CHECK_FOR_STACK_OVERFLOW	Option2
Run time and task stats gathering relat...	
USE_TRACE_FACILITY	Enabled
GENERATE_RUN_TIME_STATS	Disabled
Co-routine related definitions	
USE_CO_ROUTINES	Disabled
MAX_CO_ROUTINE_PRIORITIES	2
Software timer definitions	
USE_TIMERS	Enabled
TIMER_TASK_PRIORITY	2
TIMER_QUEUE_LENGTH	10
Interrupt nesting behaviour configuration	
LIBRARY_LOWEST_INTERRUPT_P...	15
LIBRARY_MAX_SYSCALL_INTERR...	5

Stack overflow hook

- FreeRTOS is able to check stack against overflow
- Two options
- Option 1
  - FreeRTOS check if the stack is in range which was defined on task creation
- Option 2
  - FreeRTOS use Option 1
  - Secondary on the bottom of the stack is known pattern
  - If pattern is overwritten the stack is corrupted



Configure the below parameters :

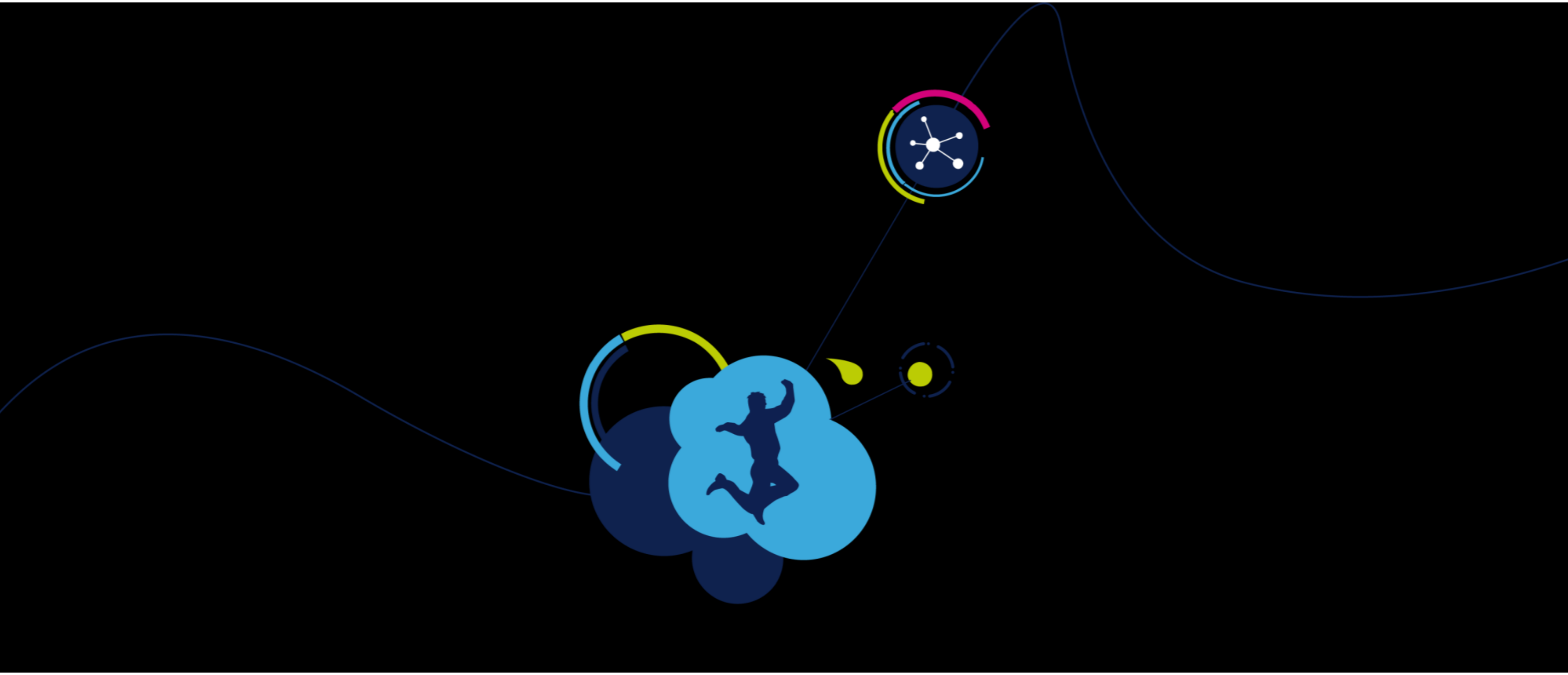
Hook function related definitions	
USE_IDLE_HOOK	Disabled
USE_TICK_HOOK	Disabled
USE_MALLOC_FAILED_HOOK	Disabled
CHECK_FOR_STACK_OVERFLOW	Option2
Run time and task stats gathering relat...	
USE_TRACE_FACILITY	Enabled
GENERATE_RUN_TIME_STATS	Disabled
Co-routine related definitions	
USE_CO_ROUTINES	Disabled
MAX_CO_ROUTINE_PRIORITIES	2
Software timer definitions	
USE_TIMERS	Enabled
TIMER_TASK_PRIORITY	2
TIMER_QUEUE_LENGTH	10
Interrupt nesting behaviour configuration	
LIBRARY_LOWEST_INTERRUPT_P...	15
LIBRARY_MAX_SYSCALL_INTERR...	5

Stack overflow hook

- Stack overflow hook callback in freertos.c created by CubeMX

```
/* Hook prototypes */
void vApplicationStackOverflowHook(xTaskHandle xTask, signed char *pcTaskName);

/* USER CODE BEGIN 4 */
void vApplicationStackOverflowHook(xTaskHandle xTask, signed char *pcTaskName)
{
   /* Run time stack overflow checking is performed if
   configCHECK_FOR_STACK_OVERFLOW is defined to 1 or 2. This hook function is
   called if a stack overflow is detected. */
}
/* USER CODE END 4 */
```

- **Do not use blocking functons(osDelay, …) in this function or while(1)**



End

Thanks for your attention

life.augmented