



FreeRTOS with CubeMX

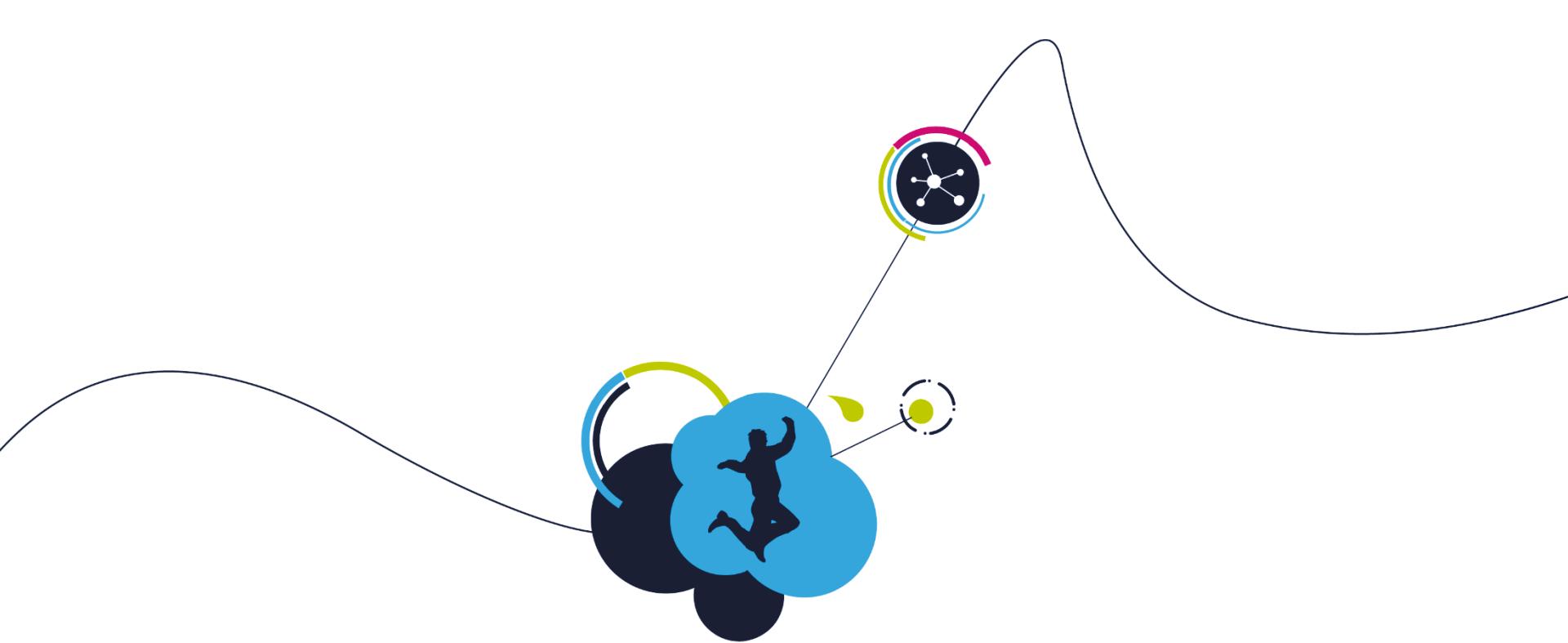
T.O.M.A.S – Technically Oriented Microcontroller Application Services (v0.02)

Modified by MCD Korea team

Contents

2

- Using SWO to print info. from STM32
- Introduction of FreeRTOS (w/ Memory allocation types)
- Task
- Queues
- Semaphore and Mutex
- Software Timer

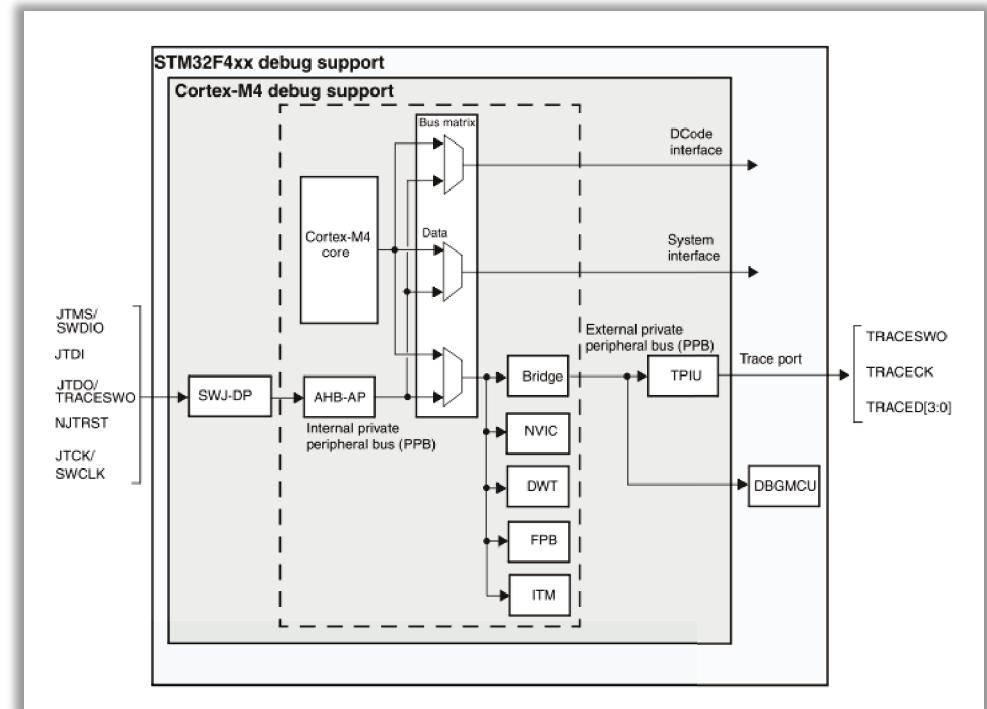


Using SWO to print info. from STM32

Using SWO

4

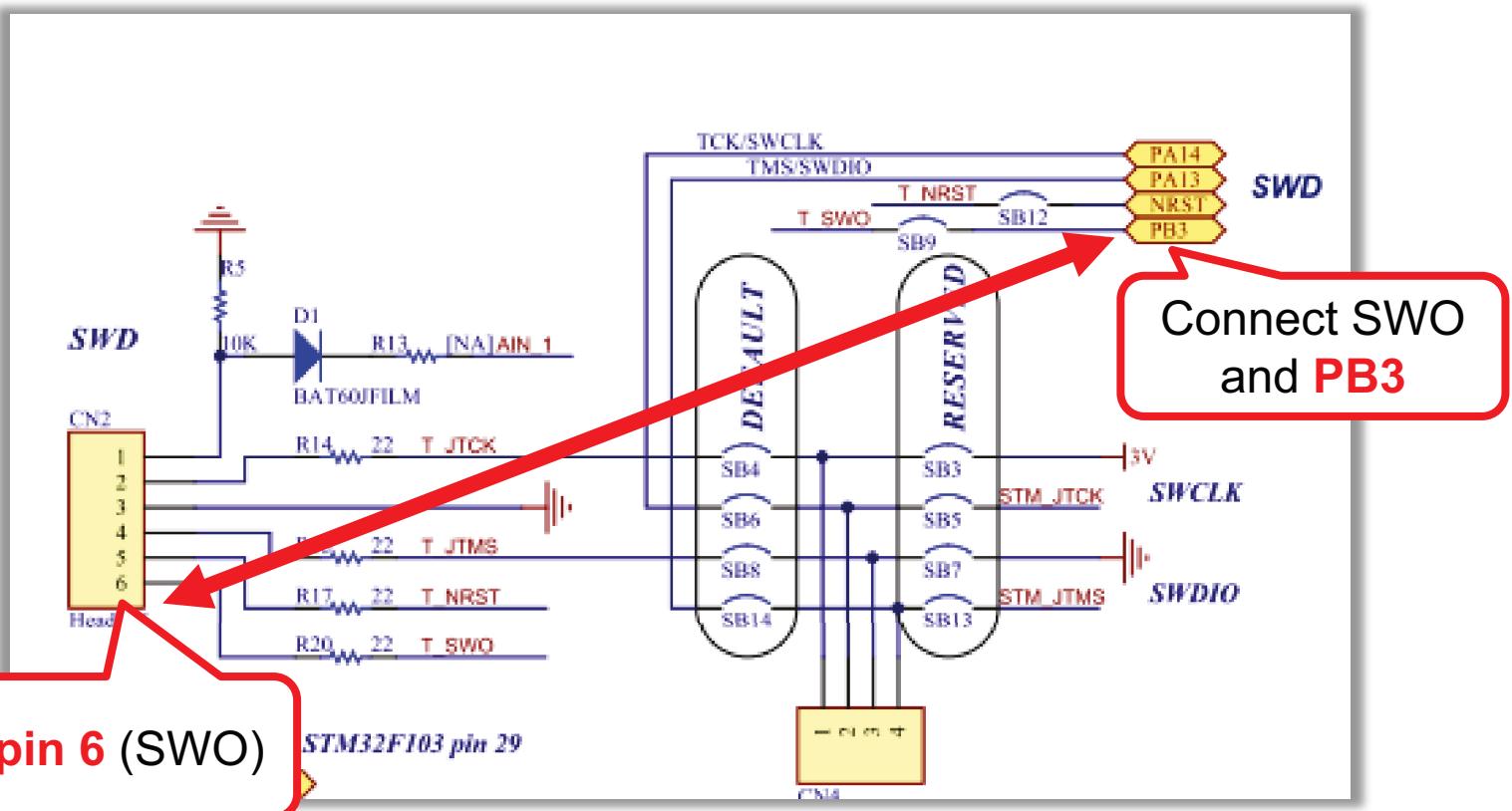
- On some STM32, there is periphery called ITM, not mix with ETM(real trace)
- This periphery can be used to internal send data from MCU over SWO pin
- Is possible to redirect the printf into this periphery
- And also some IDEs can display this information during debug
- It is similar to USART but we don't need any additional wires and PC terminal



Using SWO

5

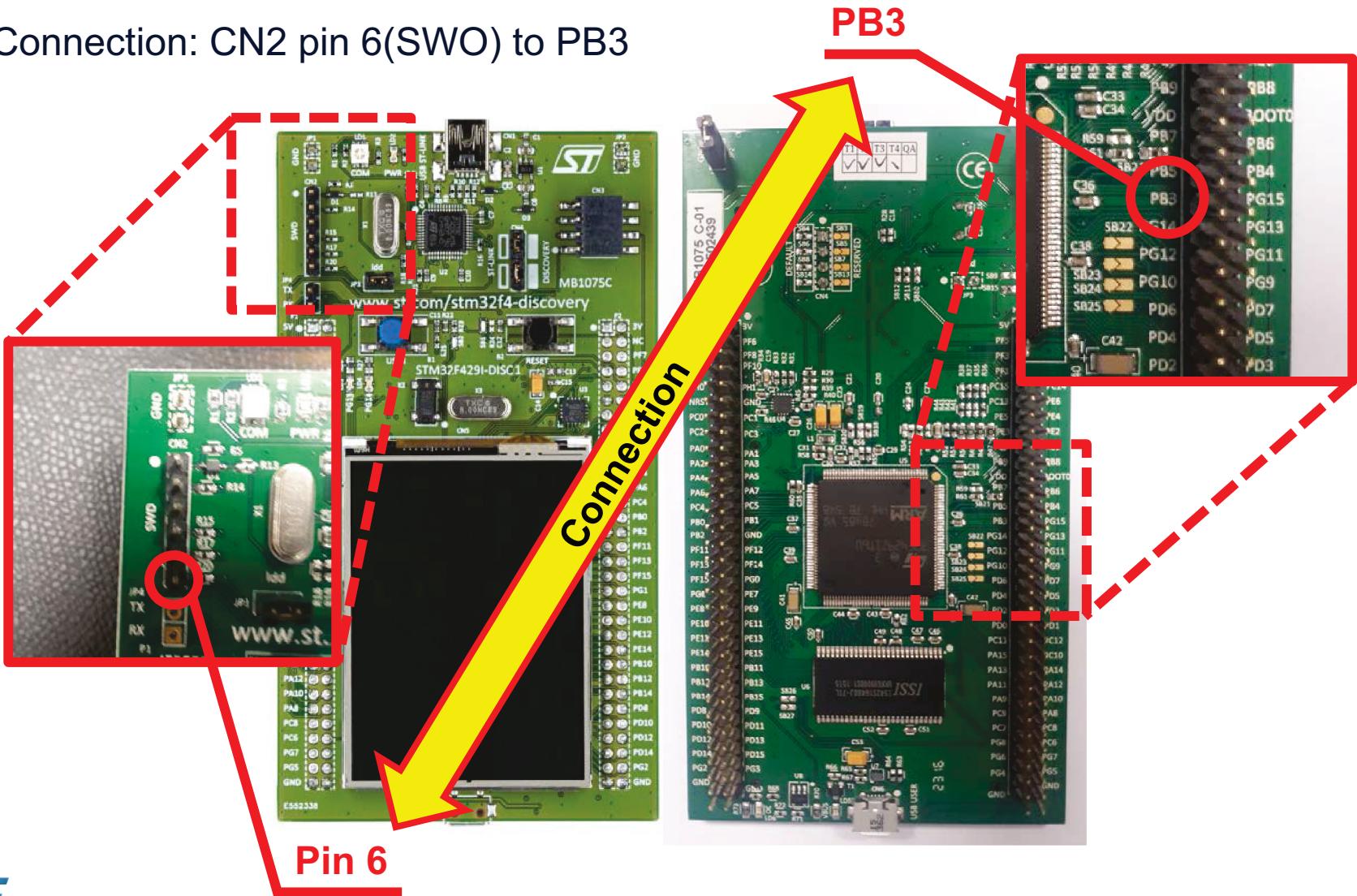
- To make SWO working you must connect the PB3 and Debugger SWO pin together



Using SWO

6

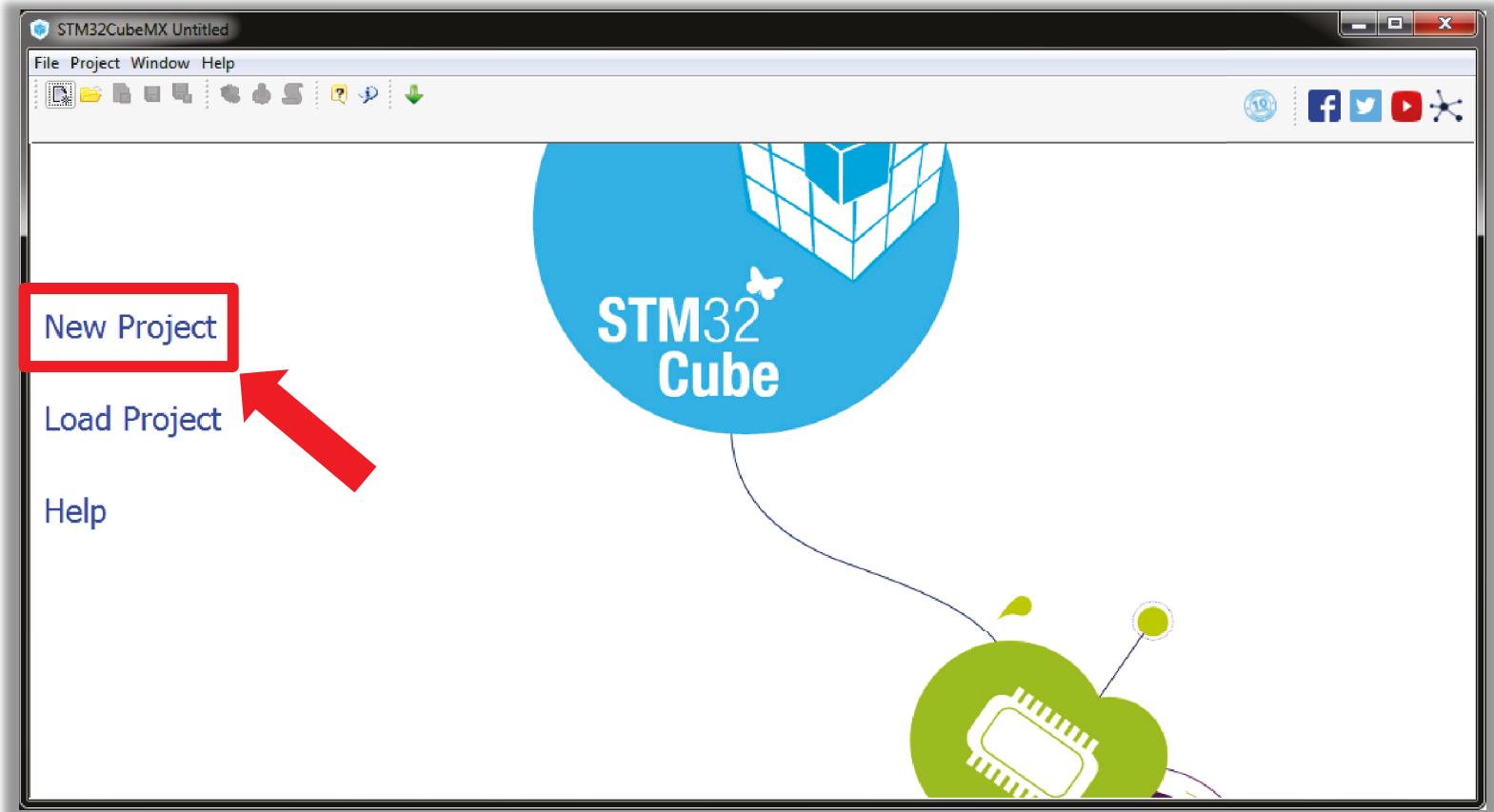
- Connection: CN2 pin 6(SWO) to PB3



Using SWO for printf

7

- Create project in CubeMX
 - Menu > File > New Project



Using SWO for printf

8

- Create project in CubeMX
 - Searching “STM32F429ZI”
 - Select “STM32F429ZITx” (LQFP144 package)

The screenshot shows the STM32CubeMX software interface. A red dashed arrow points from the search bar in the left sidebar to the search results in the main panel. Another red dashed arrow points from the selected item in the main panel to the table below.

Search Bar: STM32F429ZI

Main Panel: STM32F429ZI

Table:

Reference	Marketing Status	Unit Price for 10kU (US\$)	Board	Package
STM32F429ZITx	0.0	N...	S...	LQFP144

Text Overlay: STM32F429ZIT > LQFP144

MCU List Table:

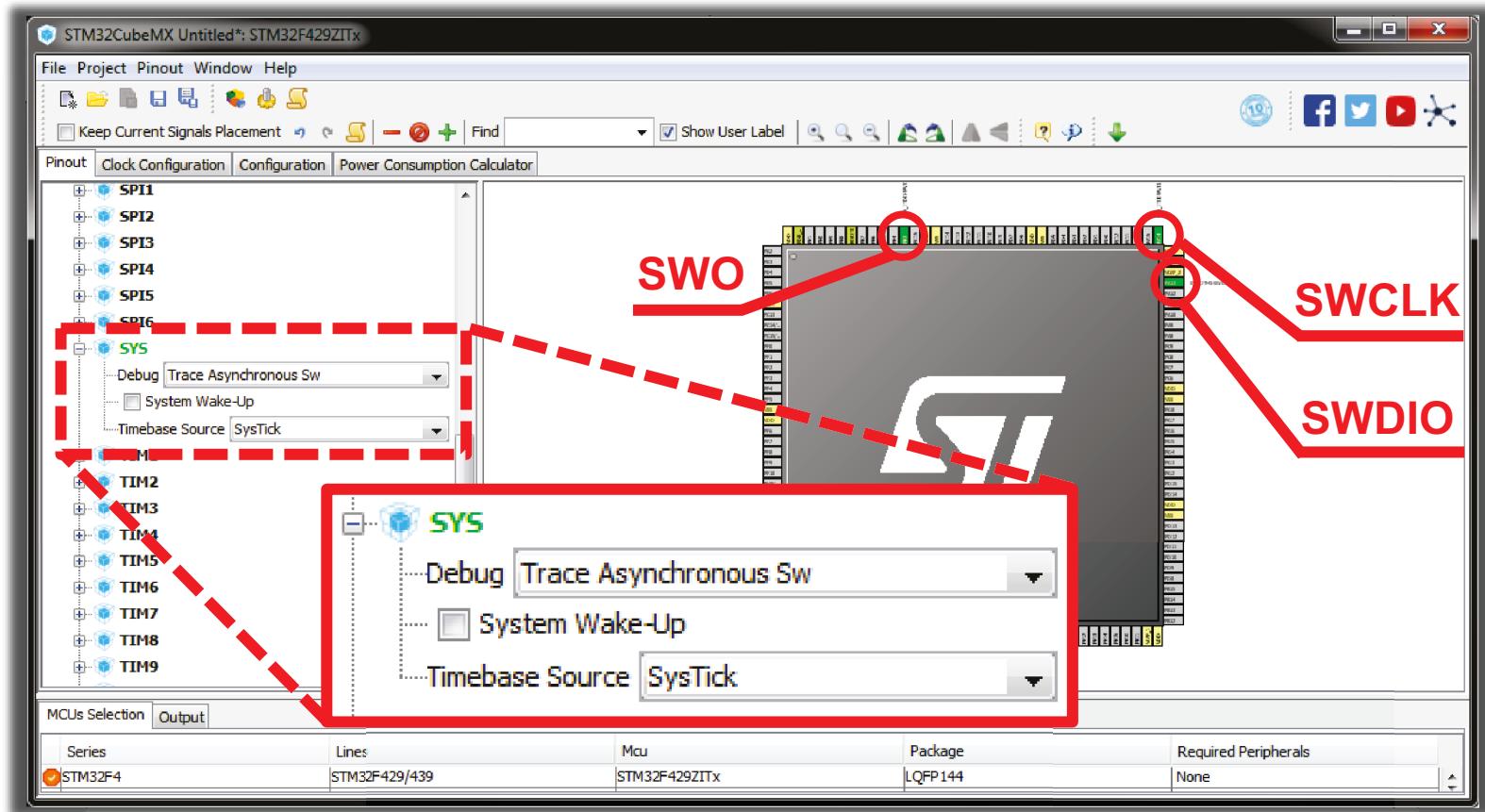
Part No	Reference	Marketing Status	Unit Price for 10kU (US\$)	Board	Package	Flash	RAM	IO	Freq.
STM32F429ZI	STM32F429ZITx	0.0	N...	S...	LQFP144	2048 kBytes	256 kBytes	114	180 MHz
		0.0			WLCSPI43	2048 kBytes	256 kBytes	114	180 MHz

Using SWO for printf

9

- Pinout

- We set the “Debug” to “Trace Asynchronous Sw”

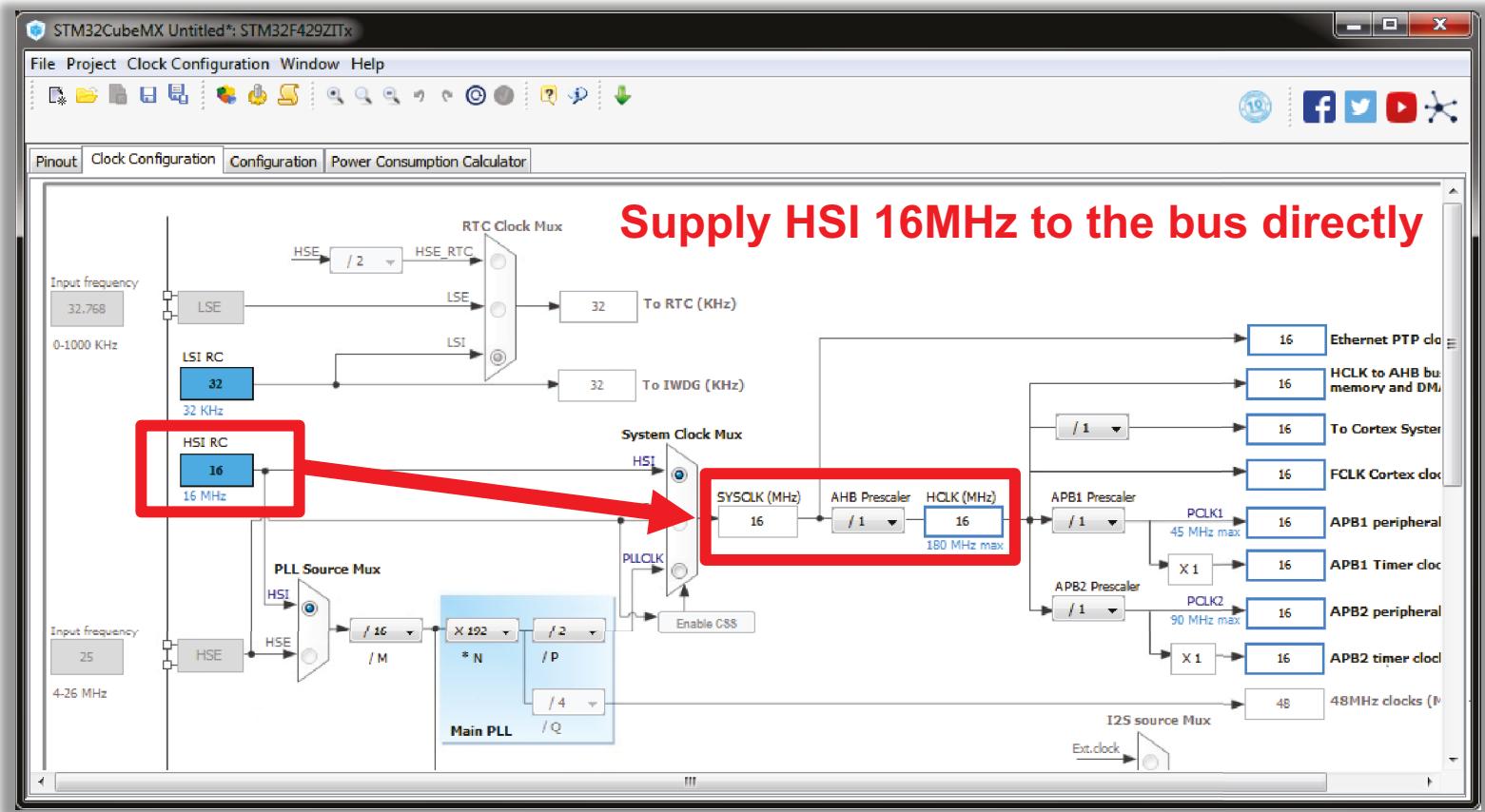


Using SWO for printf

10

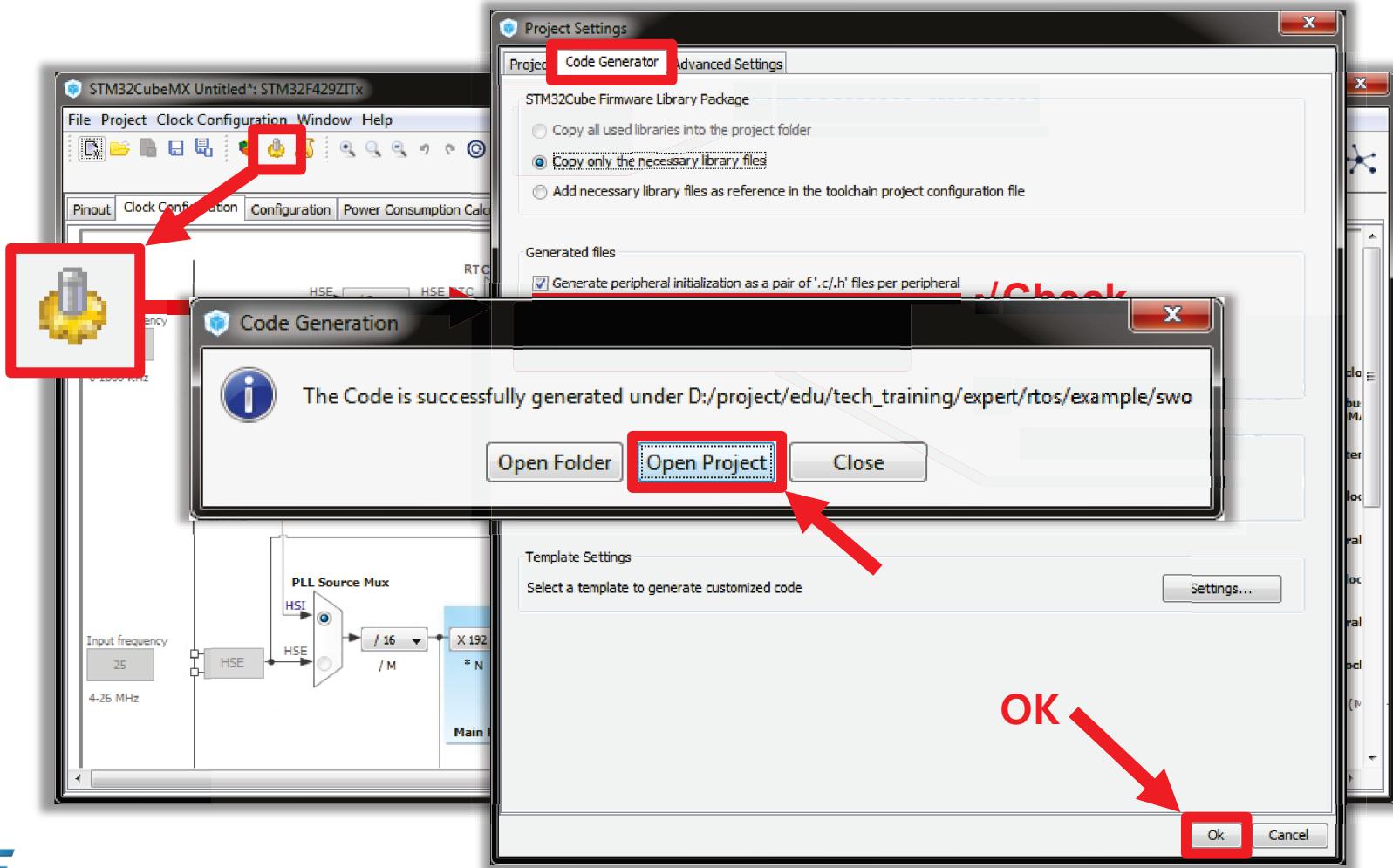
- Clock Configuration

- We need only blank project with clock initialization (keep core speed to default)



Using SWO for printf

- Now we set the project details for generation



Using SWO for printf in Atollic TrueSTUDIO

12

- To make printf() output its data to the TrueSTUDIO debugger, simply change the _write(), like this:
 - PATH: /Src/main.c

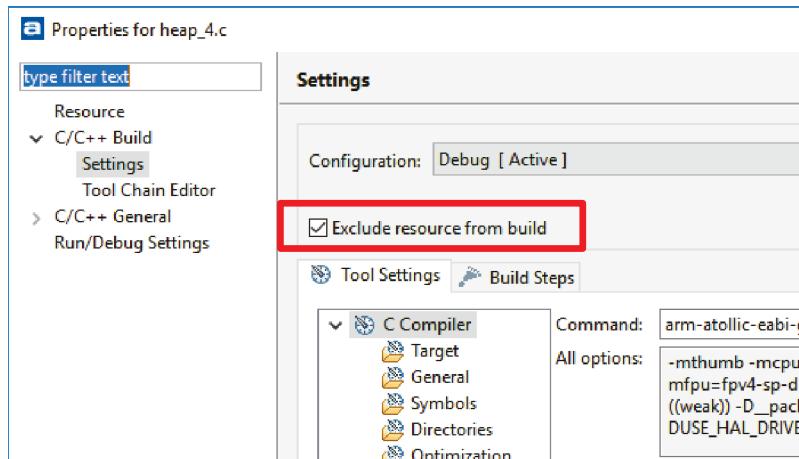
```
/* USER CODE BEGIN 4 */
int _write(int pf, char *p, int len)
{
    /* Implement your write code here, this is used by puts and printf */
    for (int i = 0; i < len; i++) {
        ITM_SendChar(*p++);
    }
    return len;
}
/* USER CODE END 4 */
```

- **ITM_SendChar()** function is CMSIS code

Using SWO for printf in Atollic TrueSTUDIO

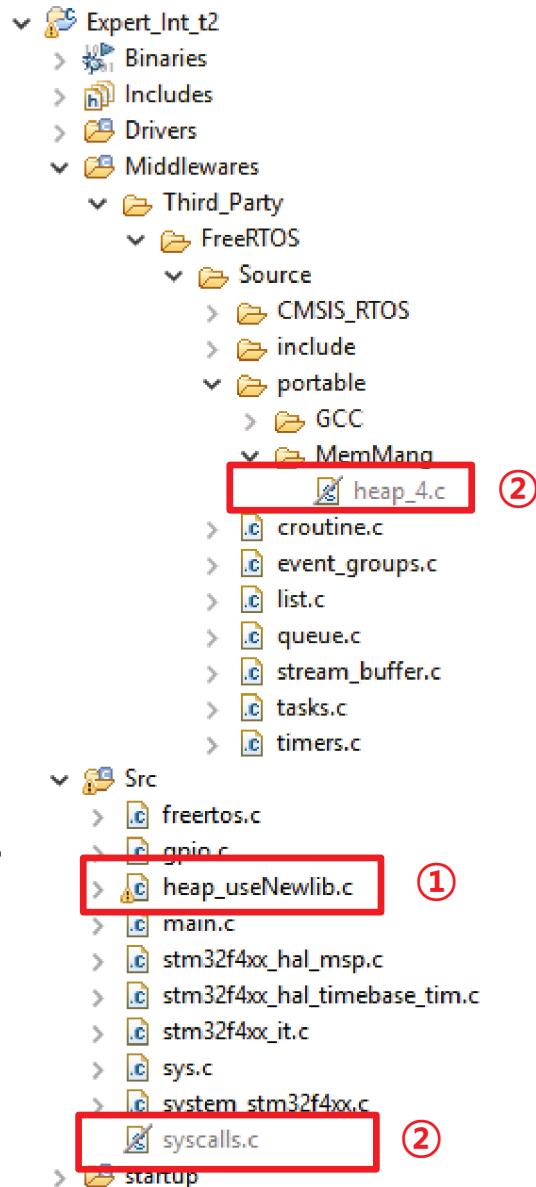
13

- Temporary patch for FreeRTOS and Newlib issue:
 - (1) Add (copy) “heap_useNewlib.c” to Src folder
 - (2) Right click on “heap_4.c” and “syscalls.c”, and check “Exclude resource from build”



- (3) Add define “#define configUSE_NEWLIB_REENTRANT 1” in FreeRTOSConfig.h

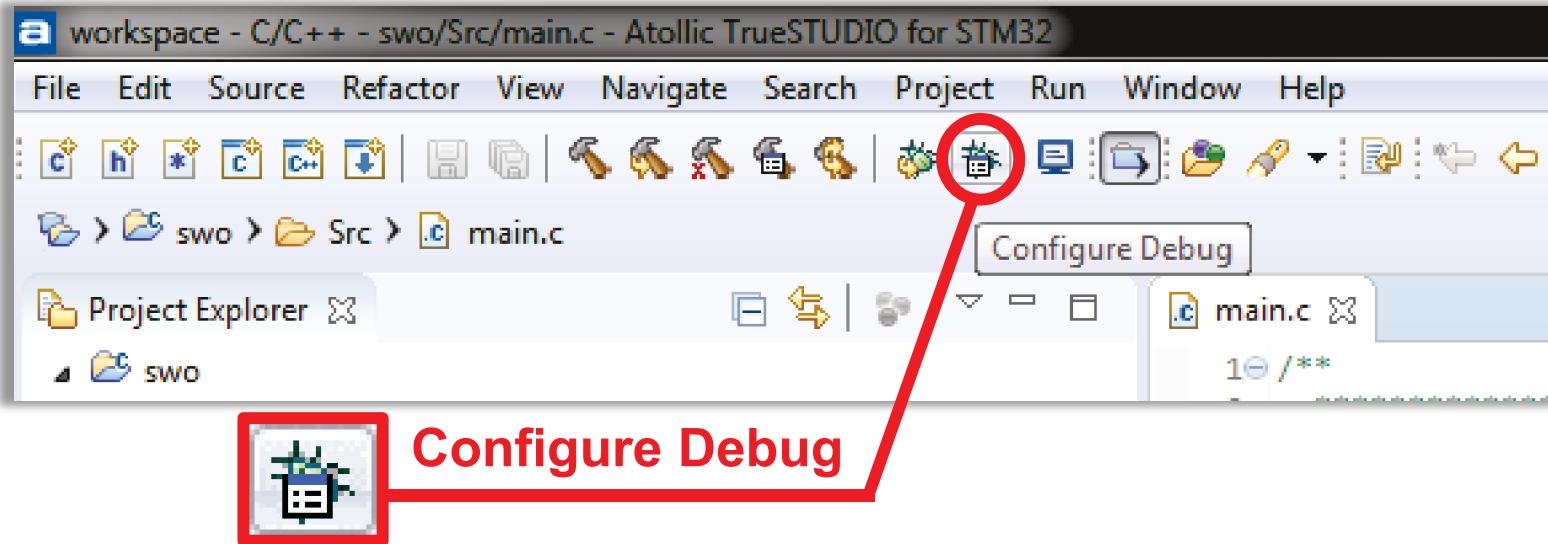
```
126 /* USER CODE BEGIN Defines */  
127 /* Section where parameter definitions can be added  
128 #define configUSE_NEWLIB_REENTRANT 1  
129 /* USER CODE END Defines */  
130  
131 #endif /* FREERTOS_CONFIG_H */
```



Using SWO for printf in Atollic TrueSTUDIO

14

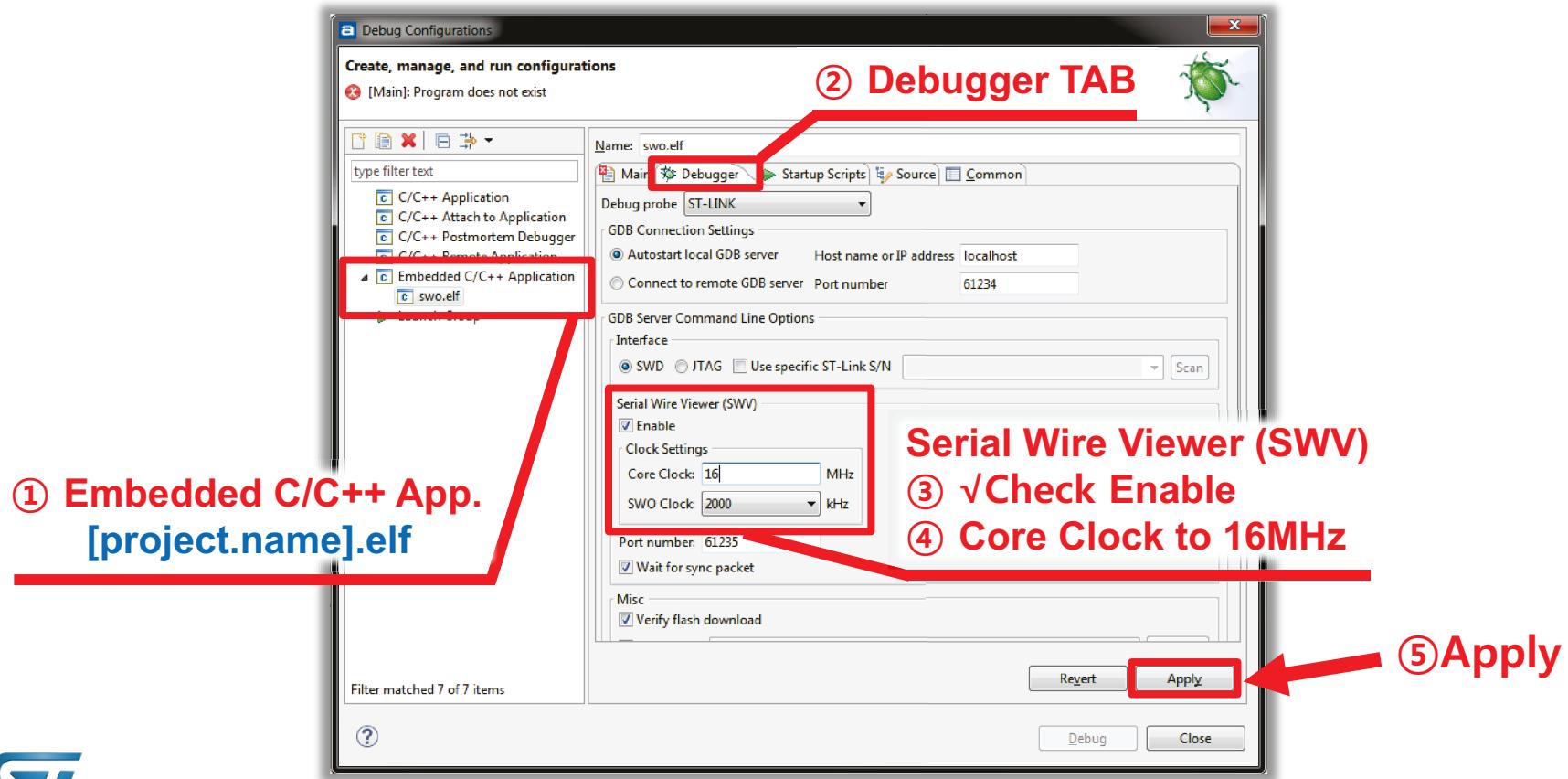
- Enable SWV tracing
 - Run [Configure Debug] → [Debugger]
※ “Run” → “Debug Configurations...”



Using SWO for printf in Atollic TrueSTUDIO

15

- Enable SWV tracing
 - Check [Enable] checkbox in Serial Wire Viewer
 - Set a core clock according to system clock of microcontroller

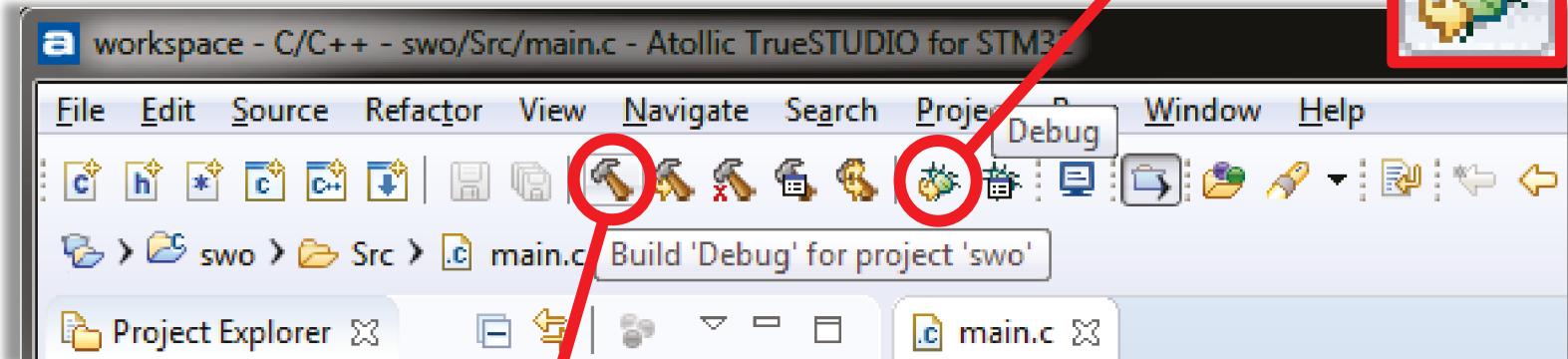


Using SWO for printf in Atollic TrueSTUDIO

16

- Configure SWV for ITM

1. Build Project ($\text{⌘ } \text{Ctrl} + \text{B}$), You can also find related menu at “Project” → {Build options}
2. Connect Host(PC) and Target board
3. Enter Debug mode ($\text{⌘ } \text{F11}$)



① Build project



Check-up Console
whether build is
OK or not

The screenshot shows the CDT Build Console window with the title "CDT Build Console [swo]". The log output is as follows:

```
Generate build reports...
Print size information
    text      data      bss      dec      hex filename
    3472       20     1568     5060   13c4 swo.elf
Print size information done
Generate listing file
Output sent to: swo.list
Generate listing file done
Generate build reports done

21:31:44 Build Finished (took 9s.837ms)
```

A red checkmark is placed next to the final line of the log.

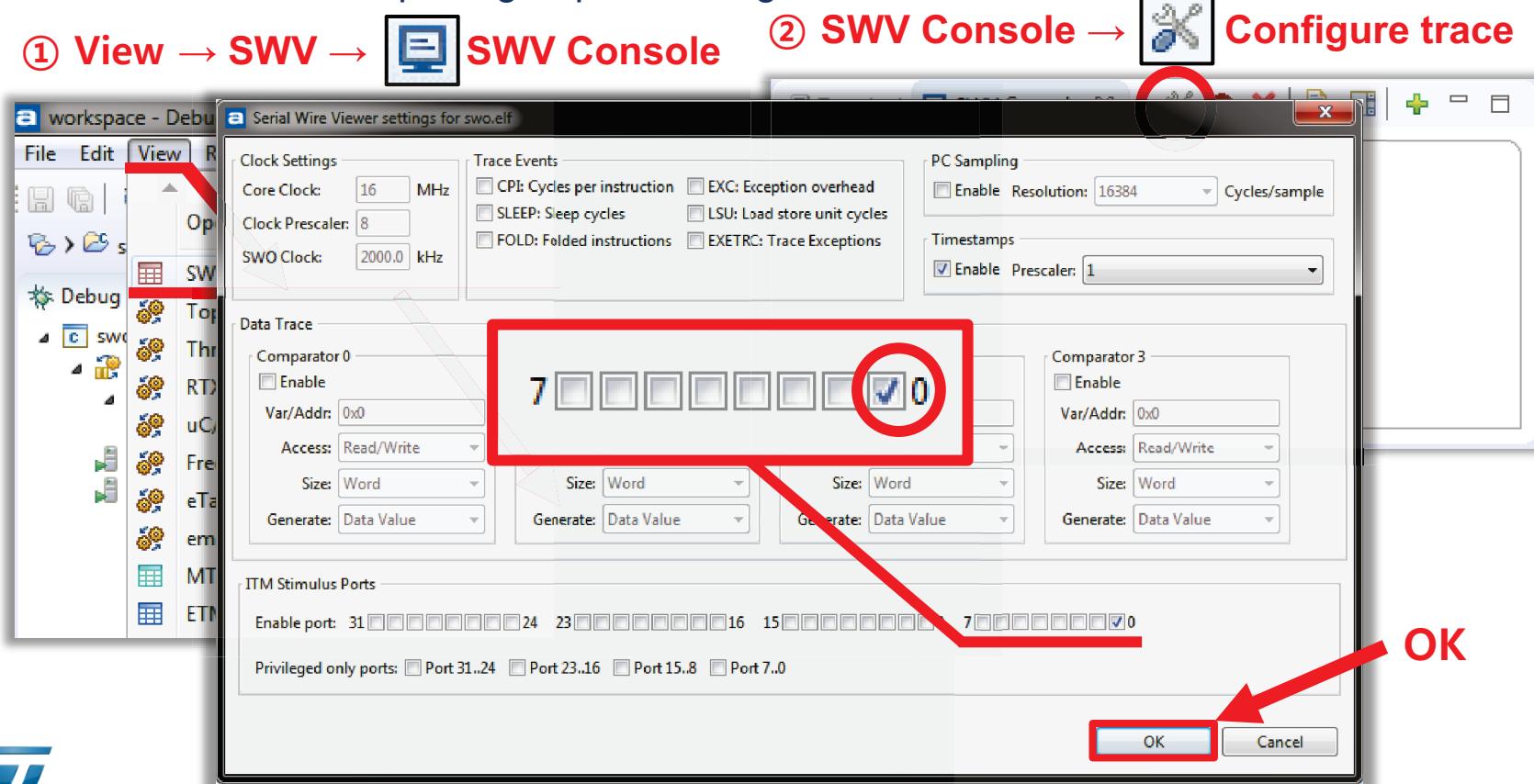
Using SWO for printf in Atollic TrueSTUDIO

17

- Configure SWV for ITM

- In the SWV Console configure window,
Configure the data ports to be traced by enabling the ITM channel 0 check box in
the ITM stimulus ports group according to this screenshot.

① View → SWV →  SWV Console ② SWV Console →  Configure trace



Using SWO for printf in Atollic TrueSTUDIO

18

- SWO Test code
 - FUNCTION: main()@/Src/main.c

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
int count = 0;
while (1)
{
    HAL_Delay(1000);
    printf("elapsed %d sec...\n", count++);
/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */
```

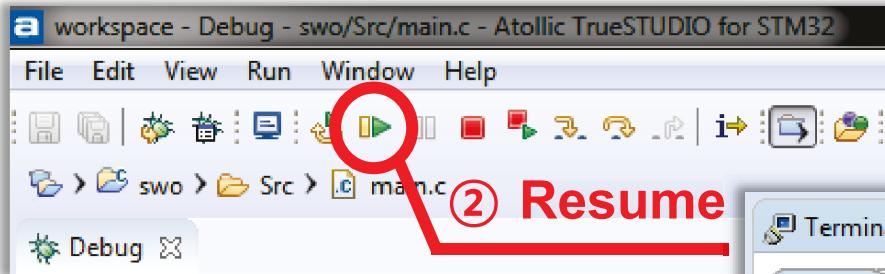
- Rebuild and Re-enter the Debug mode for SWO testing
 1. Exit from Debug mode ($\text{⌘} \text{Ctrl} + \text{F2}$)
 2. Rebuild ($\text{⌘} \text{Ctrl} + \text{B}$) and re-enter the Debug mode ($\text{⌘} \text{F11}$)

Using SWO for printf in Atollic

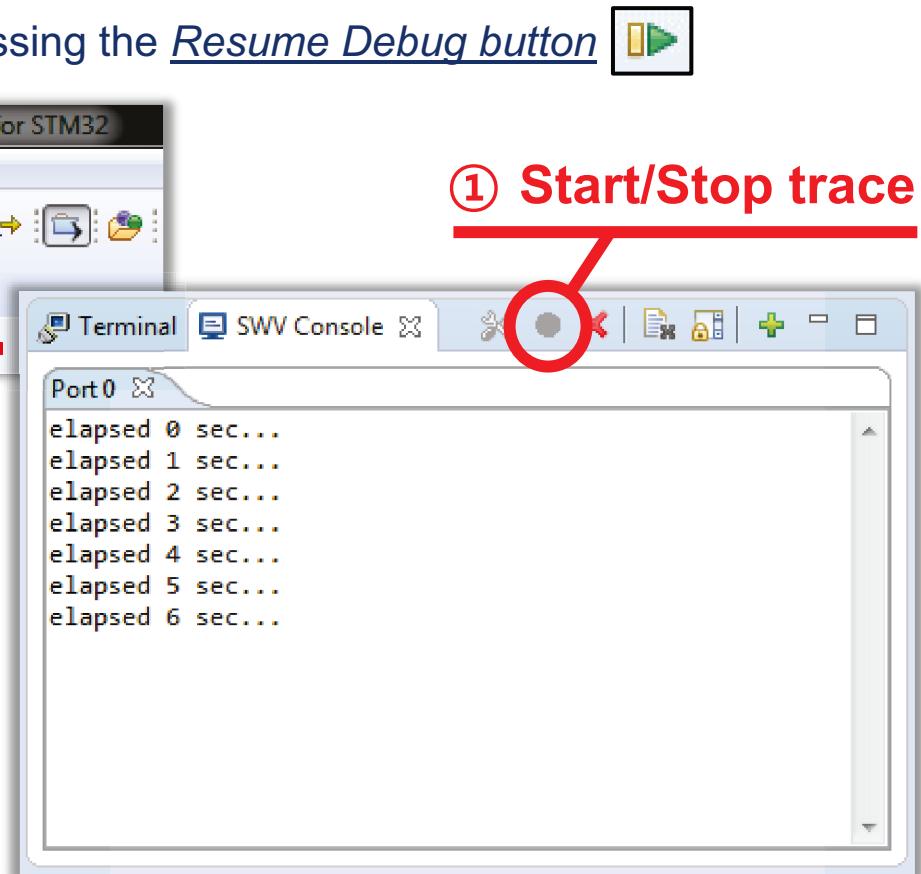
19

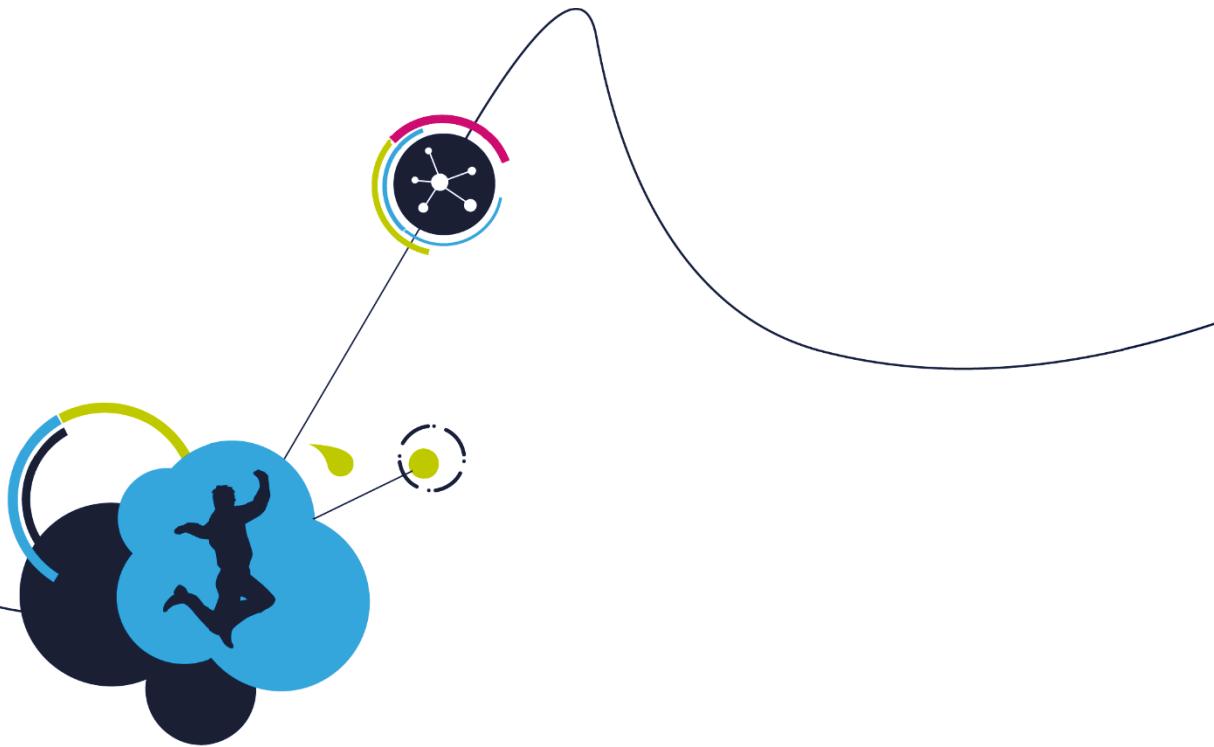
- Output debugging message

1. Press the red Start/Stop Trace button  to send the SWV configuration to the target board.
2. Start the target execution by pressing the Resume Debug button 



3. The SWV console will now show the printf() debugging message.





FreeRTOS

FreeRTOS

About FreeRTOS

21

- 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)

23

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
	uxTaskPriorityGet	Get 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 taskEXIT_CRITICAL	Enter(Exit from) a critical section (When entering, it stops context switching)

FreeRTOS APIs overview (2/2)

24

API category	FreeRTOS API	Description
Message Queue	xQueueCreate	Creates a queue
	xQueueSend	Sends data to queue
	xQueueReceive	Receive data from the queue
Semaphore Mutex	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

25

- Implementation in file ***cmsis-os.c***
 - “\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

26

- 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

27

- 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

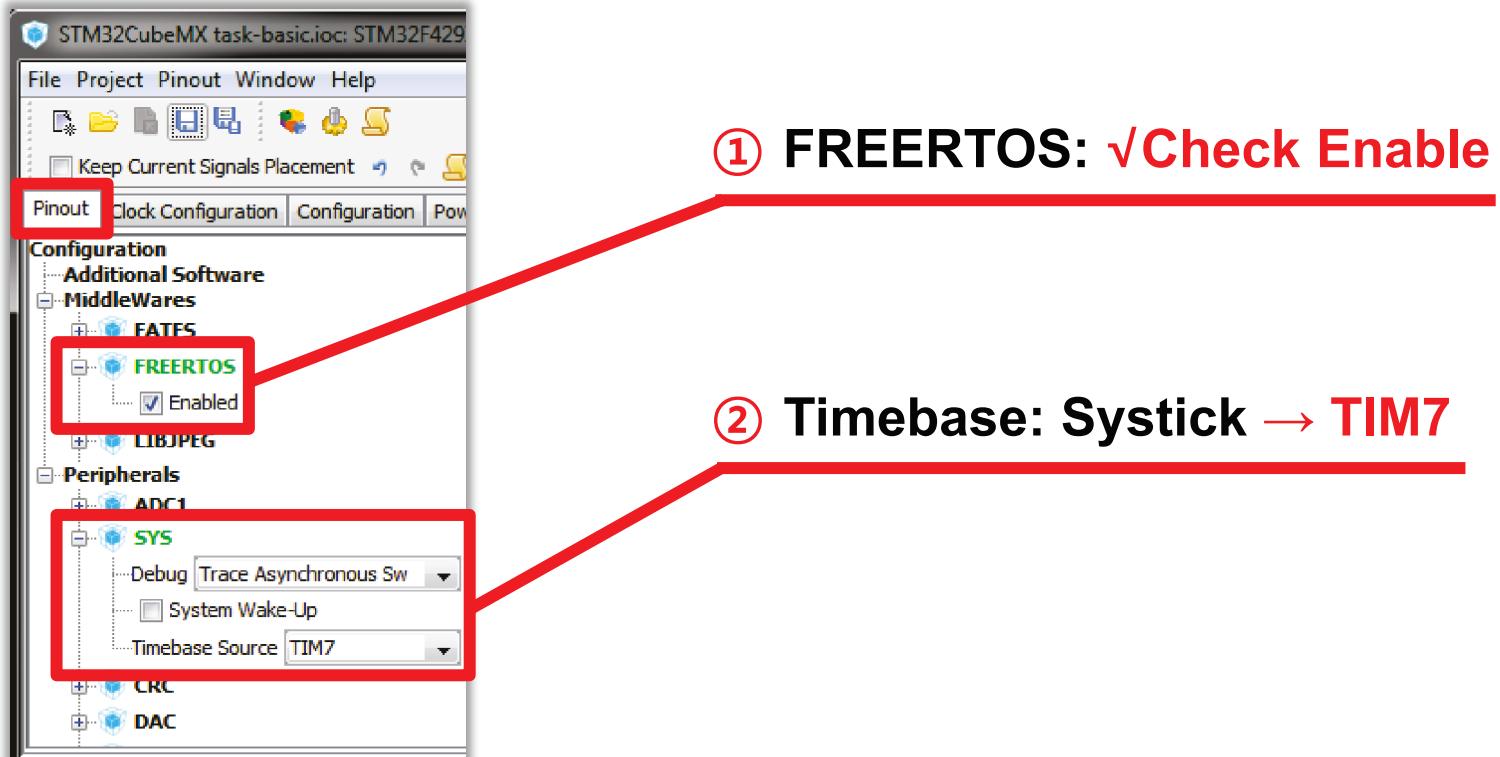
28

- 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 portSUPPRESS_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

29

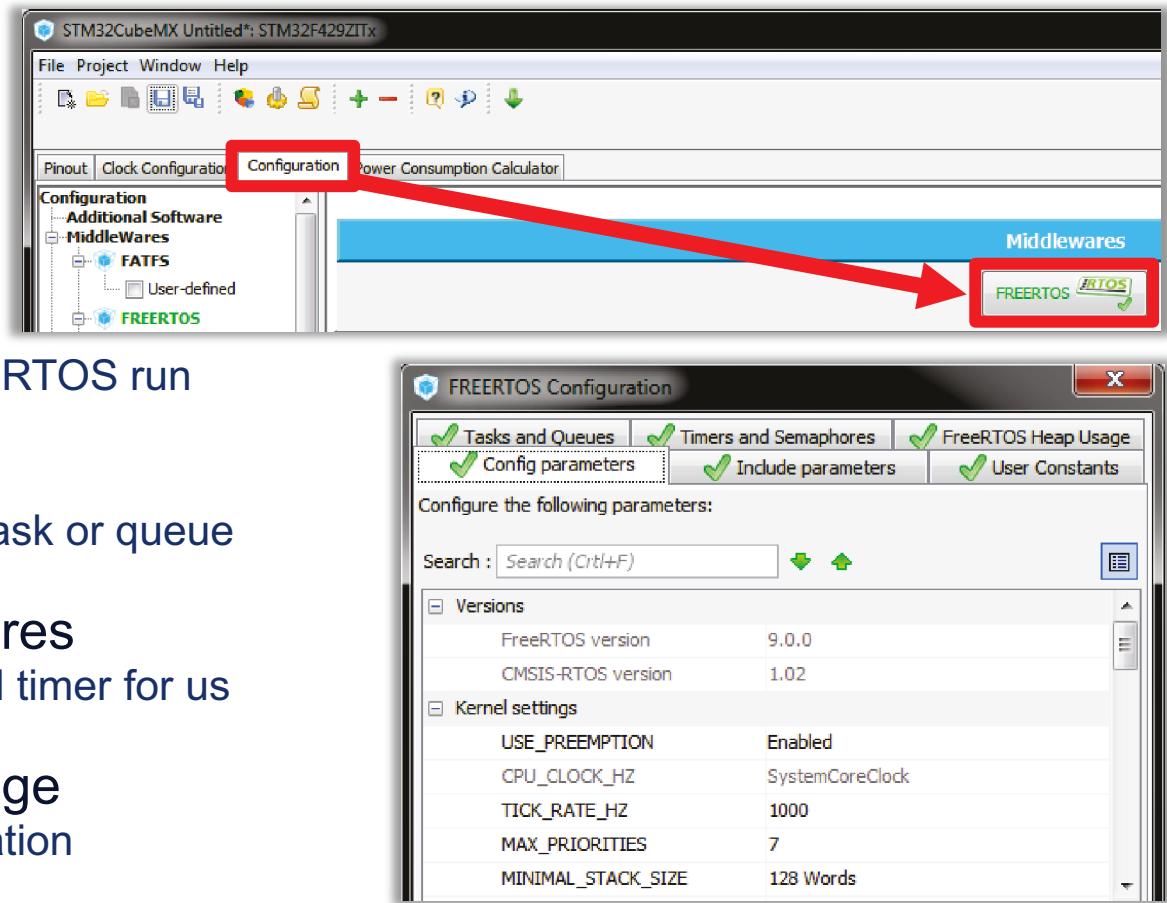
- Use CubeMX project from printf example
- MiddleWares view in Pinout TAB, select in **FreeRTOS**
- Peripherals view, “SYS → Timebase source” change to **TIM7**



FreeRTOS in CubeMX

30

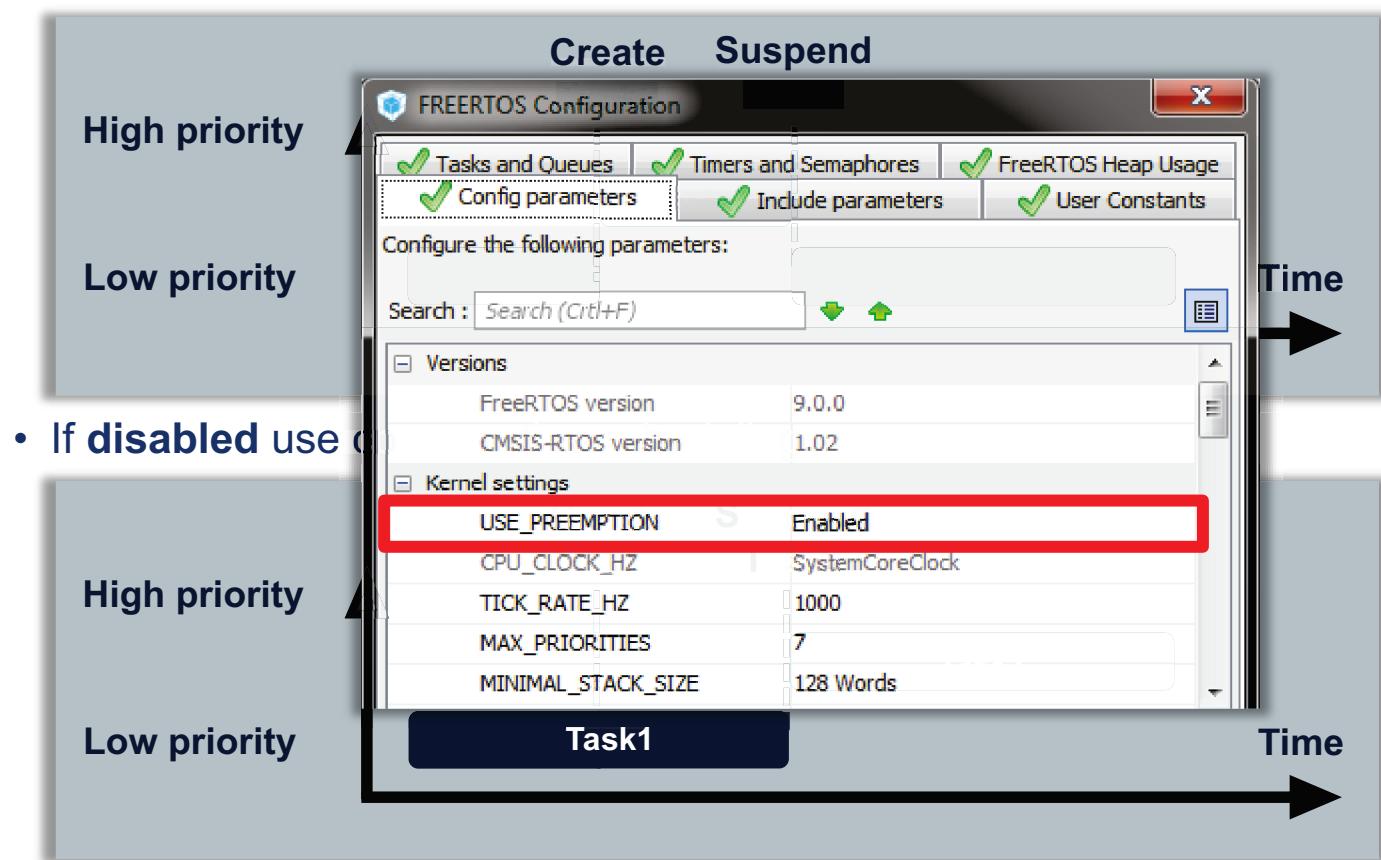
- In Configuration TAB is now possible to configure FreeRTOS
- FreeRTOS configuration supported by CubeMX
- Config parameters
 - About kernel features
 - About Memory mgmt.
- Include parameters
 - Additional APIs
 - Not necessary for FreeRTOS run
- Tasks and Queues
 - We can easily create task or queue
- Timers and Semaphores
 - Create semaphore and timer for us
- FreeRTOS Heap Usage
 - Memory usage information

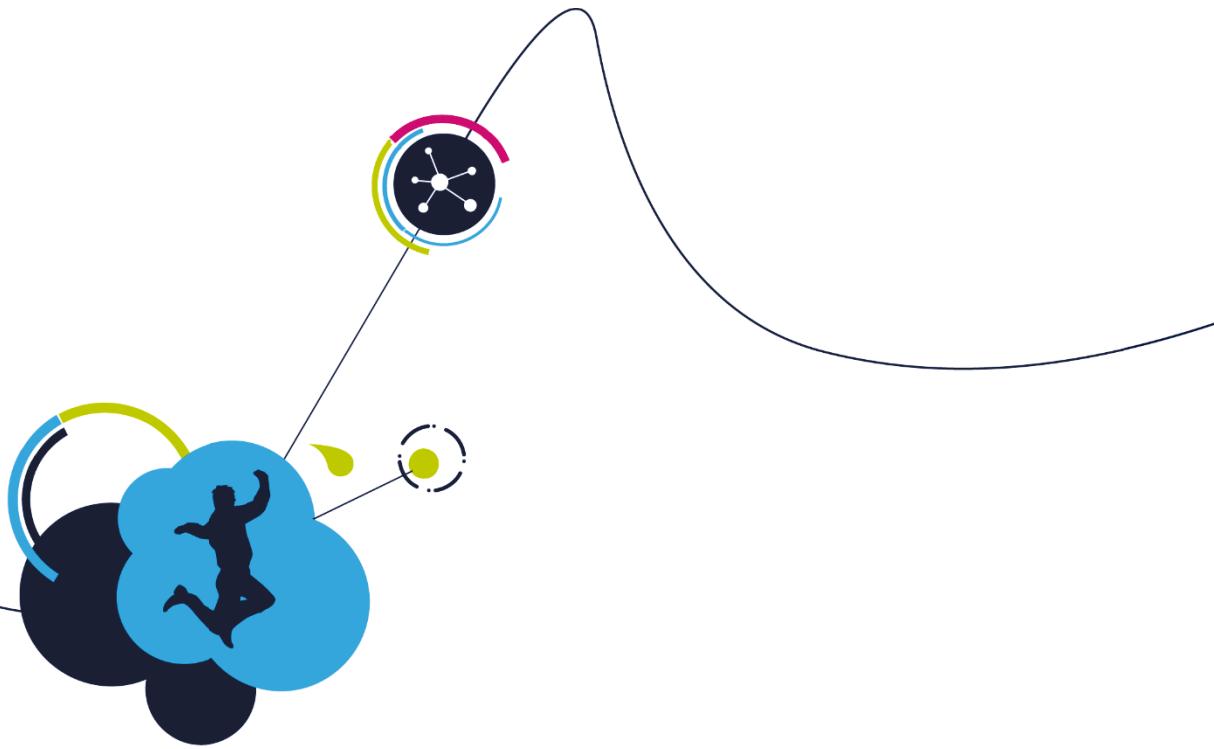


Kernel settings

31

- Use preemption





FreeRTOS Memory allocations types

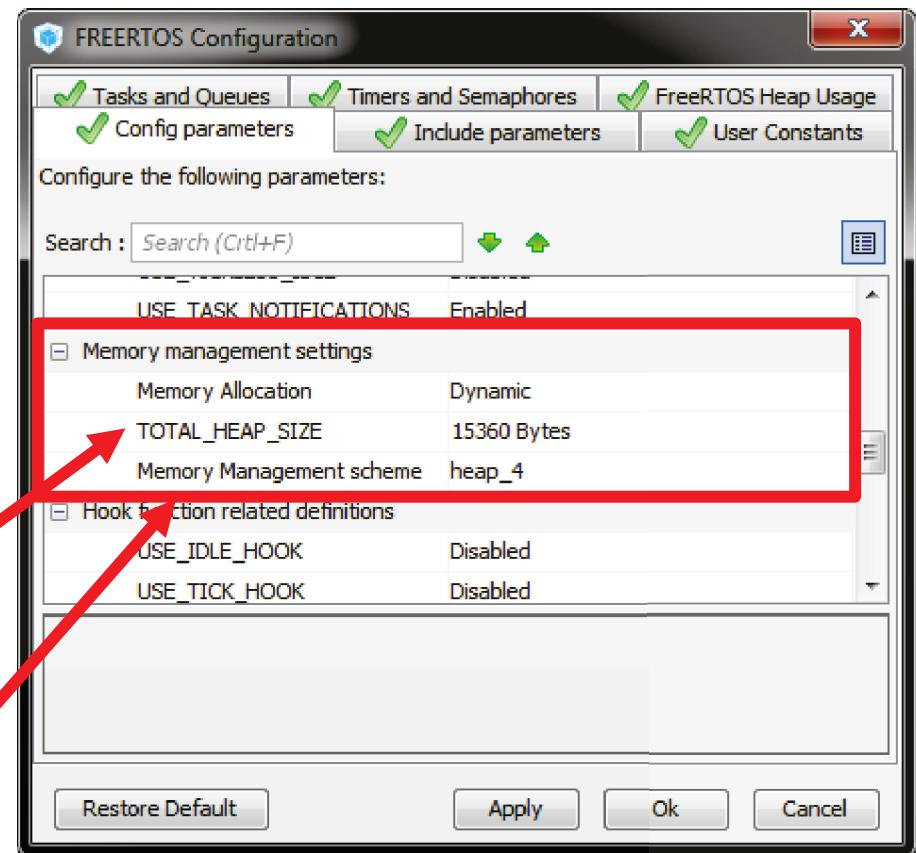


Dynamic memory management

- FreeRTOS have own heap which is use for components
 - Tasks
 - Queues
 - Semaphores
 - Mutexes
 - Dynamic memory allocation
- Is possible to select type of memory allocation

Total heap size for RTOS

How is memory allocated and deallocated



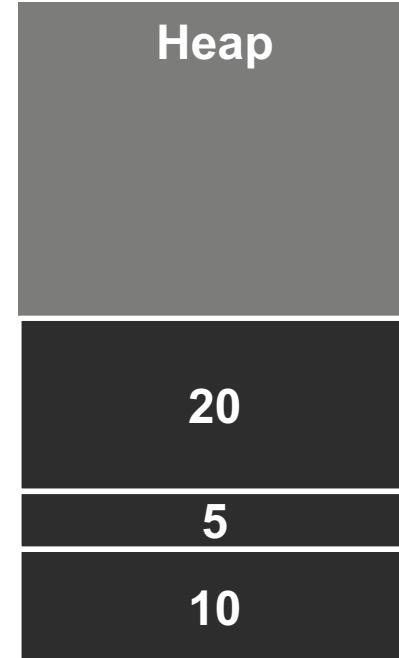
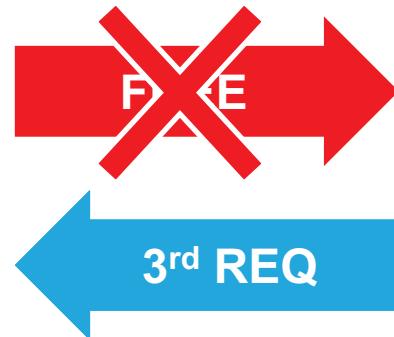


Dynamic memory management

- Heap_1.c

- Simplest allocation method (deterministic sequential allocation), but does not allow freeing of allocated memory
 - Could be interesting when no memory freeing is necessary

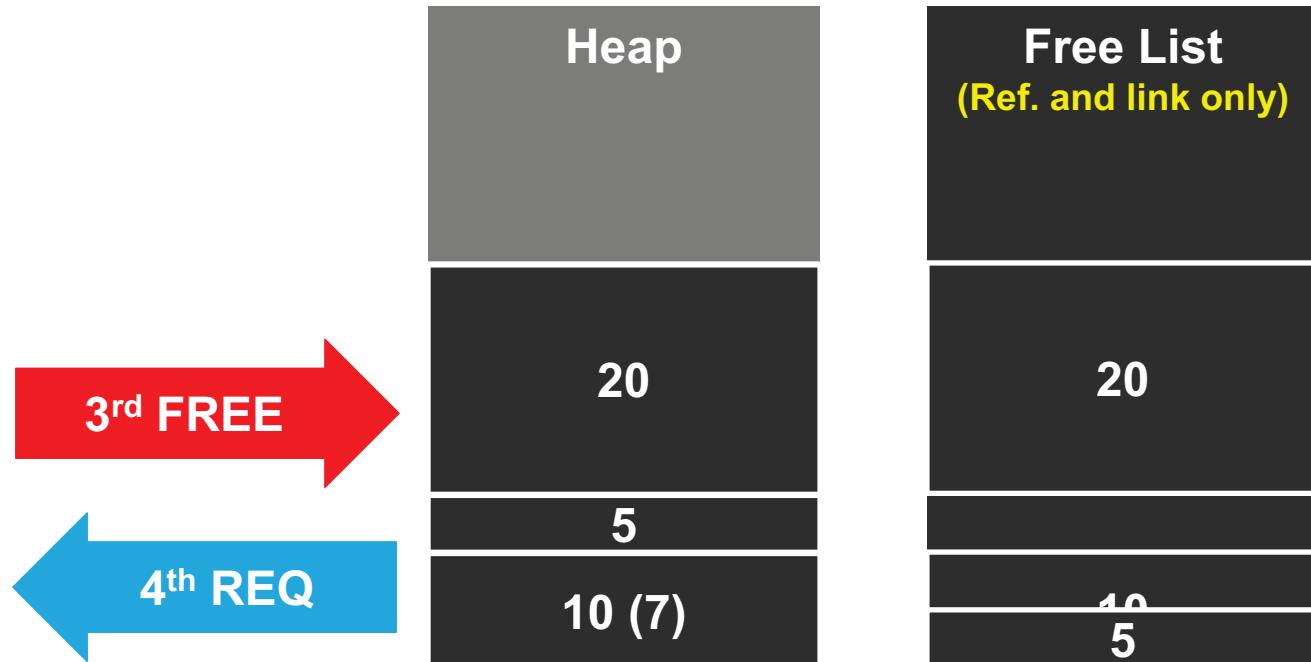
Is not possible to return memory to heap



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



FreeRTOS

Dynamic memory management

36

- Heap_3.

- 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

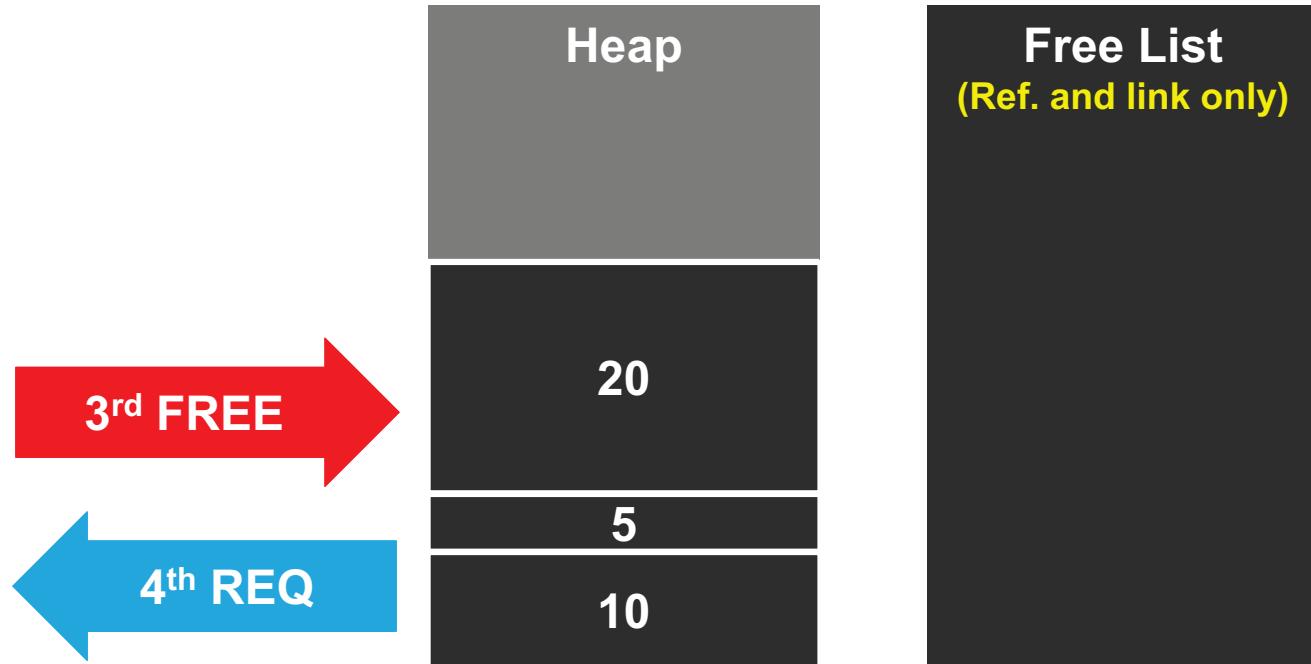




Dynamic memory management

- 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

38

- Use heap_4.c
- Memory Handler definition

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

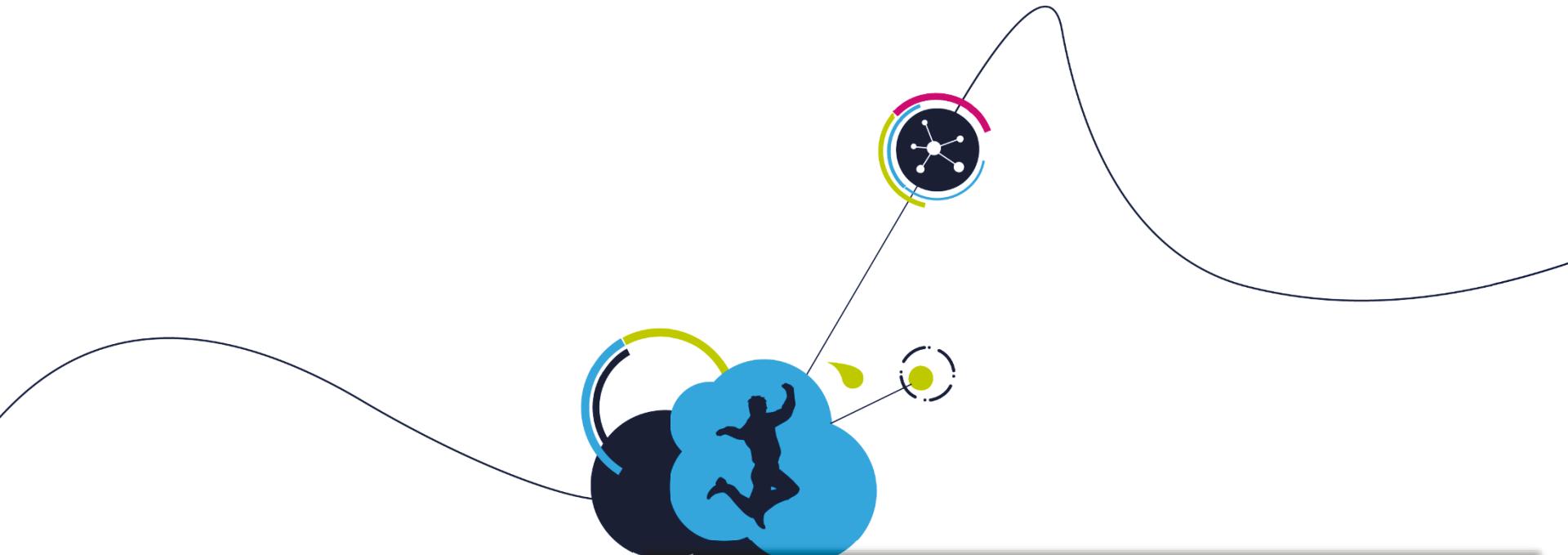
- Memory allocation

```
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

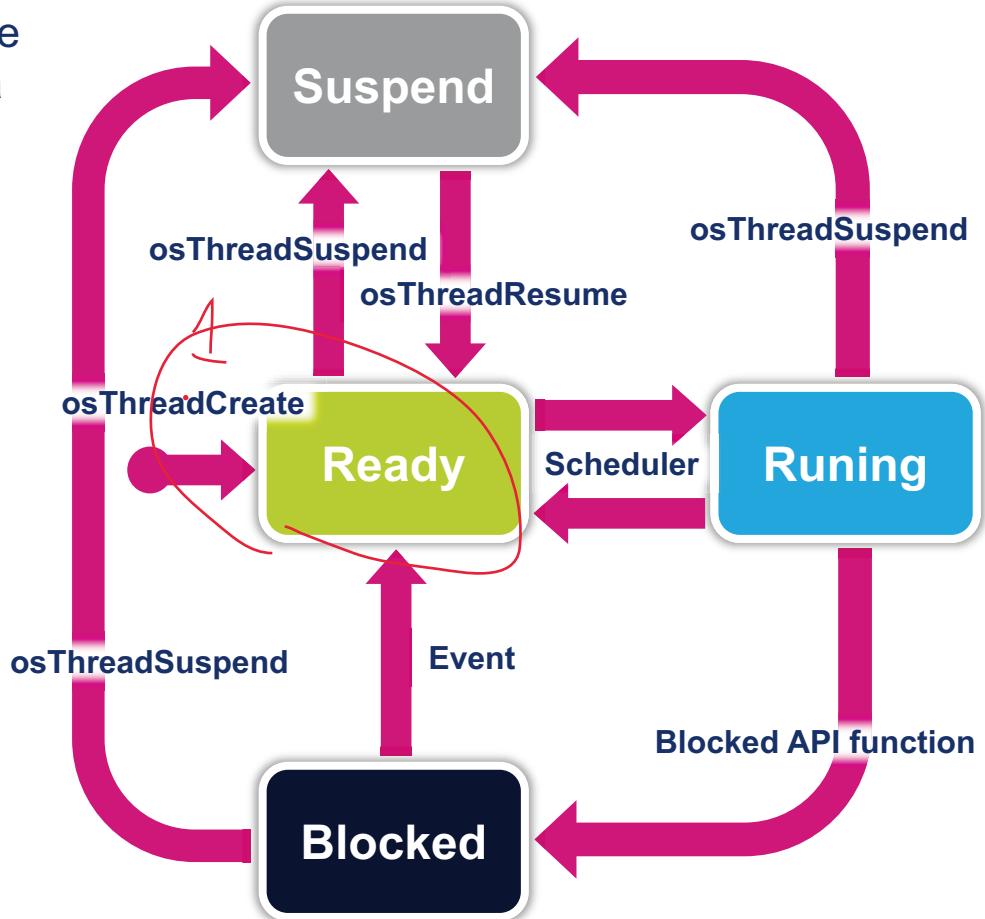




FreeRTOS Task states

40

- **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 either a temporal or external event
- **Suspended**
 - Task not available for scheduling



- Task switching on STM32?
- Cortex core 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

42

- PendSV interrupt

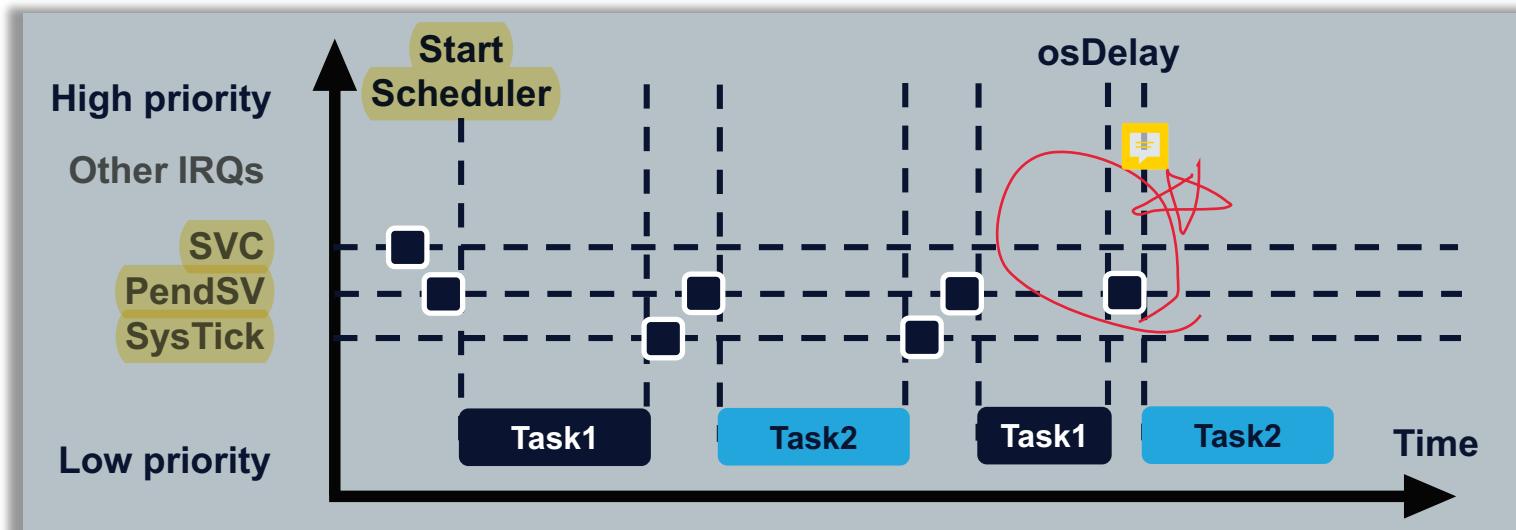
- Manage context switching
- Lowest NVIC interrupt priority
- Not triggered by any peripherals
- Pending state set from other interrupts or from task which want end earlier (non MPU version)

- SVC interrupt

- Interrupt risen by SVC instruction
- Called if task want end earlier (MPU version)
- In this interrupt set pending state PendSV (MPU version)

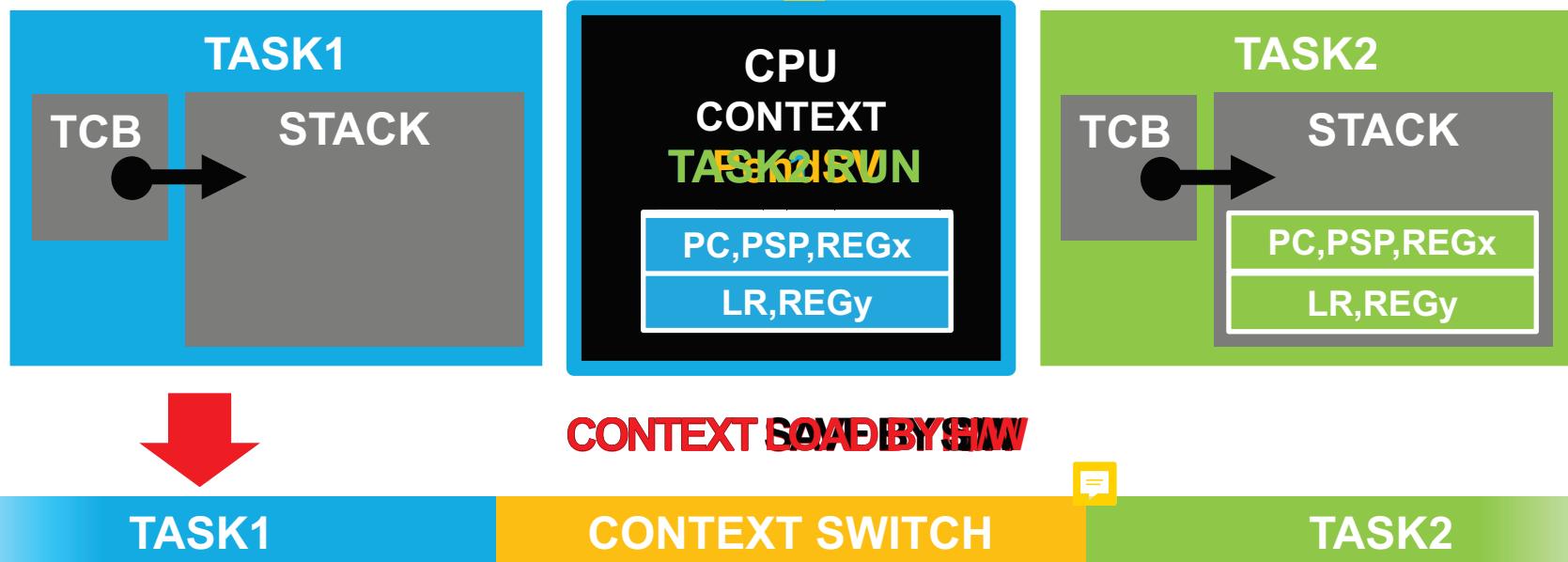
- SysTick timer

- Set PendSV if context switch is necessary



Stack pointer and Context Switch

- 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





- 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);
```

- 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

UZI 21/11/2023
H. CUBEVIX

- By default defined one defaultTask

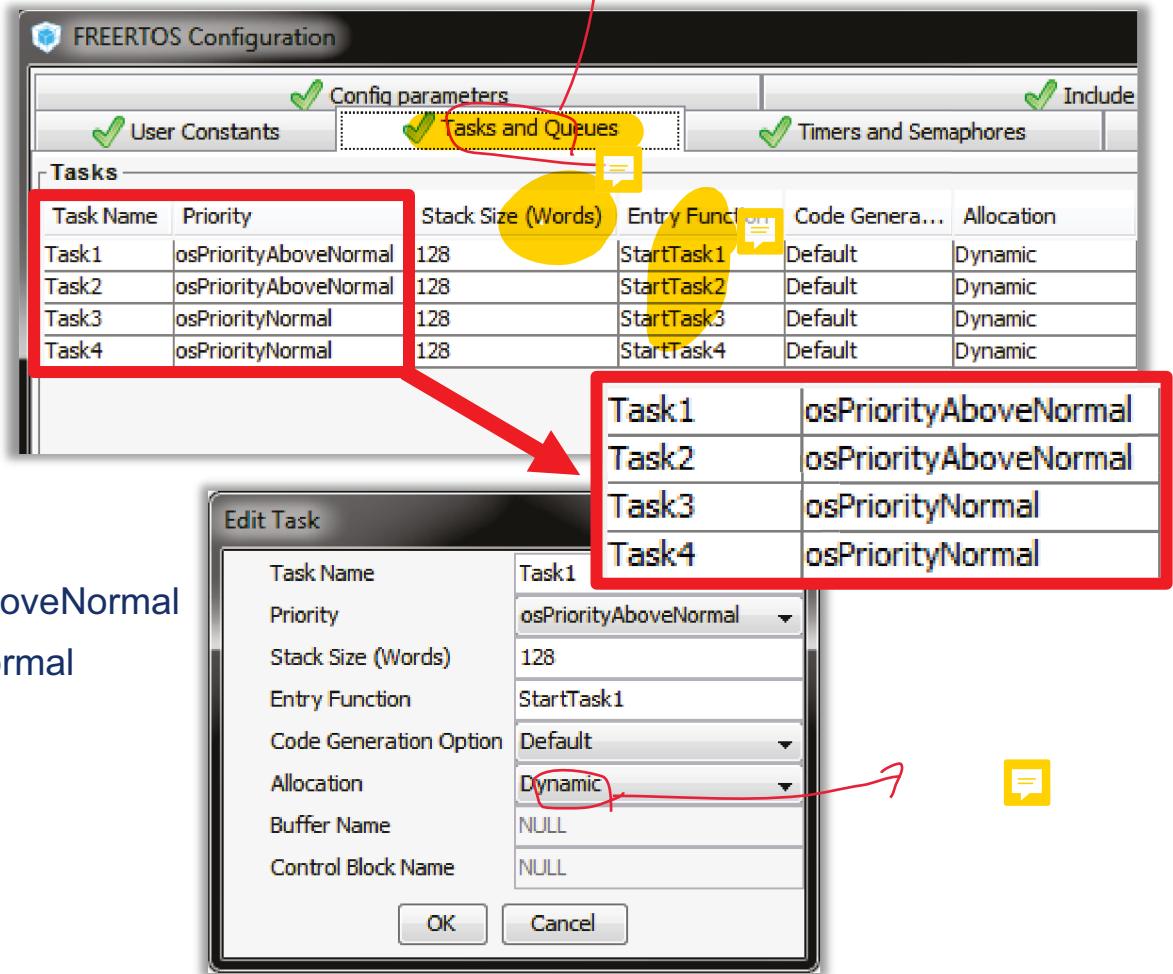
- Task is defined by

- Name
- Priority
- Stack size
- Name of entry function

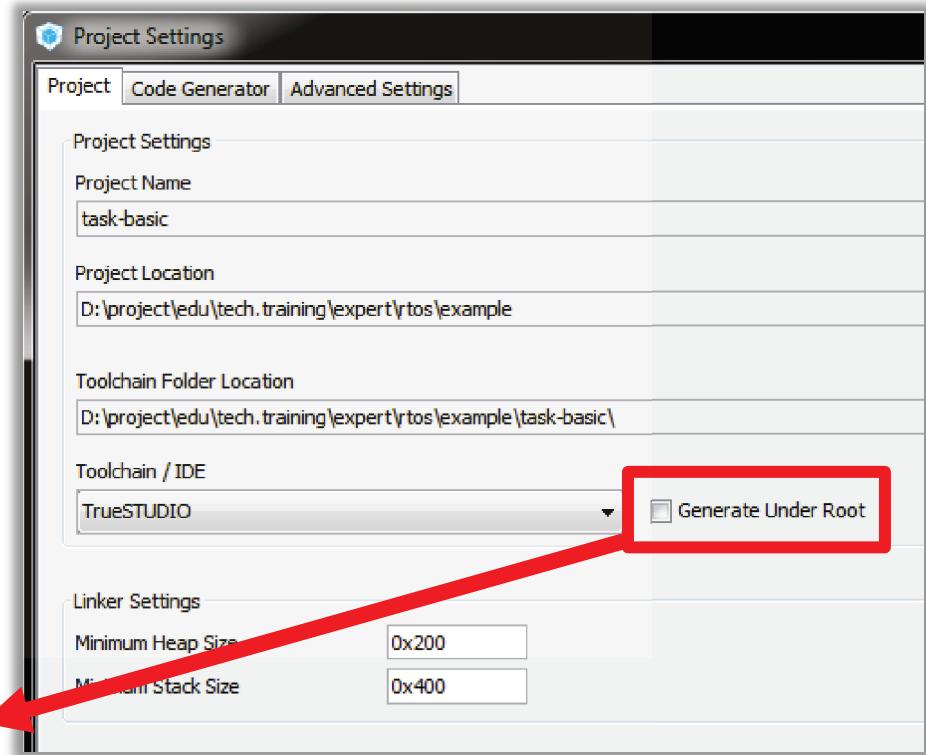
- Define 4 tasks

- With different priority

- Task1, Task2 - osPriorityAboveNormal
- Task3, Task4 - osPriorityNormal



- 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



Disable “Generate Under Root”

- User RTOS code generated to “freertos.c” file
- Any component in FreeRTOS need to have handle, very similar to CubeMX

```
/* Private variables -----  
--- */  
osThreadId Task1Handle;  
osThreadId Task2Handle;  
osThreadId Task3Handle;  
osThreadId Task4Handle;
```

- Task function prototypes, names was taken from CubeMX

```
/* Private function prototypes -----  
--- */  
void StartTask1(void const * argument);  
void StartTask2(void const * argument);  
void StartTask3(void const * argument);  
void StartTask4(void const * argument);
```

- Before the scheduler is start we must create tasks
 - [MX_FREERTOS_Init\(\) @ freertos.c](#)

```
/* Create the thread(s) */  
/* definition and creation of Task1 */  
osThreadDef(Task1, StartTask1, osPriorityAboveNormal, 0, 128);  
Task1Handle = osThreadCreate(osThread(Task1), NULL);  
  
/* definition and creation of Task2 */  
osThreadDef(Task2, StartTask2, osPriorityAboveNormal, 0, 128);  
Task2Handle = osThreadCreate(osThread(Task2), NULL);  
  
/* definition and creation of Task3 */  
osThreadDef(Task3, StartTask3, osPriorityNormal, 0, 128);  
Task3Handle = osThreadCreate(osThread(Task3), NULL);  
  
/* definition and creation of Task4 */  
osThreadDef(Task4, StartTask4, osPriorityNormal, 0, 128);  
Task4Handle = osThreadCreate(osThread(Task4), NULL);
```

Create task, allocate memory

Define task parameters

- Start the scheduler, the scheduler function never ends

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

- Task must have inside infinity loop in case we don't want to end the task
 - Task1 and Task2 call "osDelay"

```
void StartTask{1 or 2}(void const * argument)
{
    /* USER CODE BEGIN StartTask1 */
    /* Infinite loop */
    for(;;) {
        printf("Task {1 or 2}\n");
        osDelay(1000);
    }
    /* USER CODE END StartTask1 */
}
```

- Second loop is same as previous except “HAL_Delay”

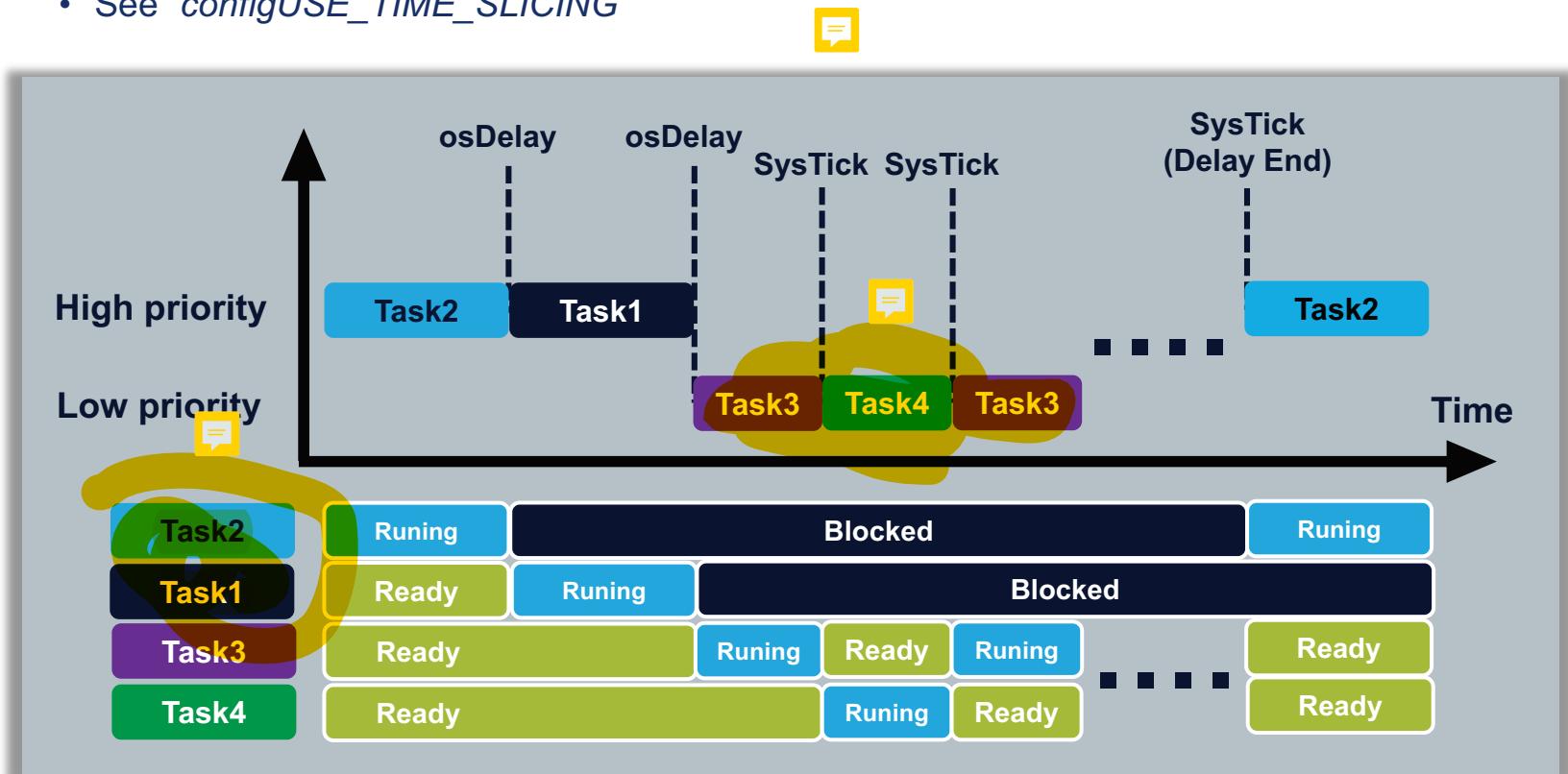
```
/* StartTask2 function */
void StartTask{3 or 4}(void const * argument)
{
    /* USER CODE BEGIN StartTask2 */
    /* Infinite loop */
    for(;;) {
        printf("Task {3 or 4}\n");
        HAL_Delay(1000);    
    }
    /* USER CODE END StartTask2 */
}
```

- Compile and run project in debug and watch terminal window

Tasks lab

52

- Tasks which have OS delays are yield the processing to other task
- Without OS delays the tasks will be in running state or in Ready state
 - See “*configUSE_TIME_SLICING*”



- Delay function

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

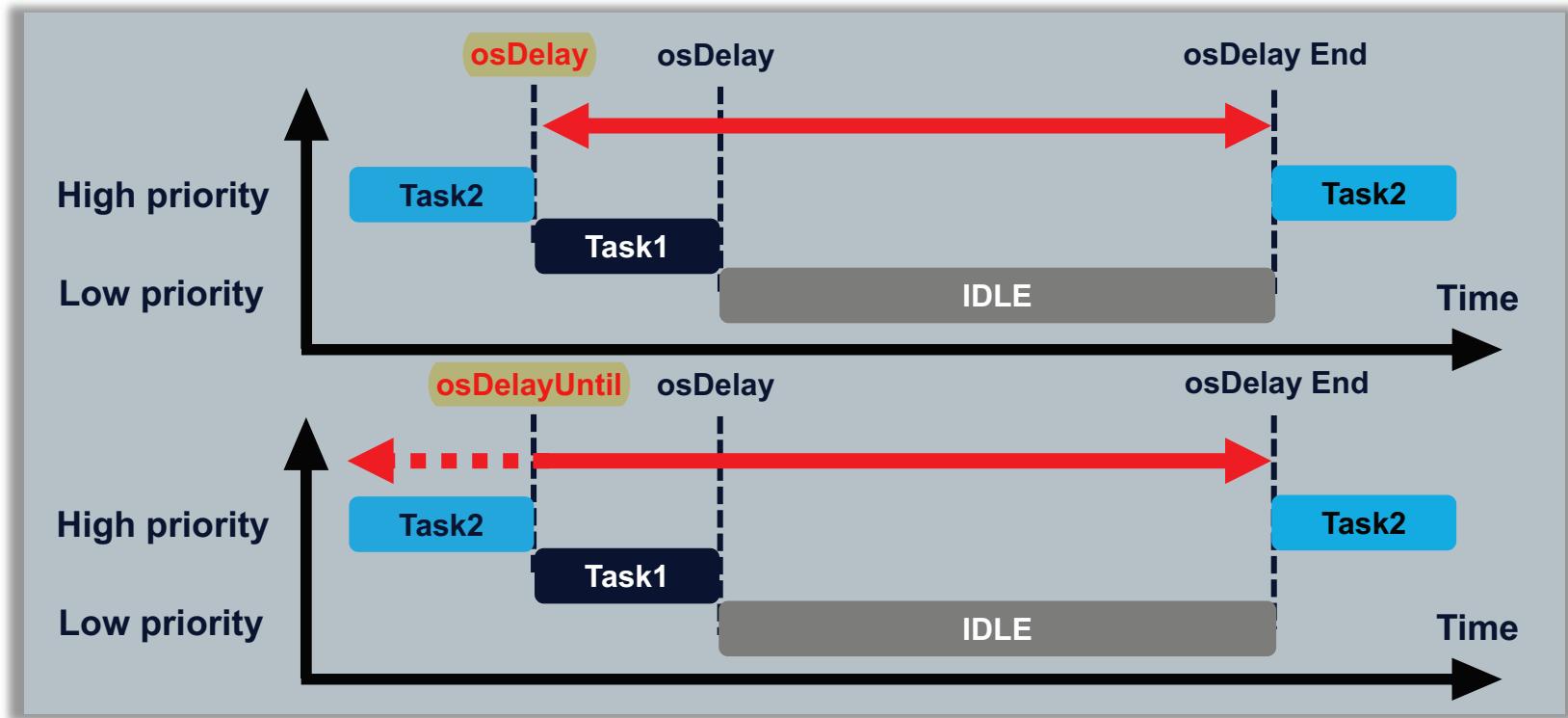
- Delay function which measure time from which is delay measured

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

osDelay function

54

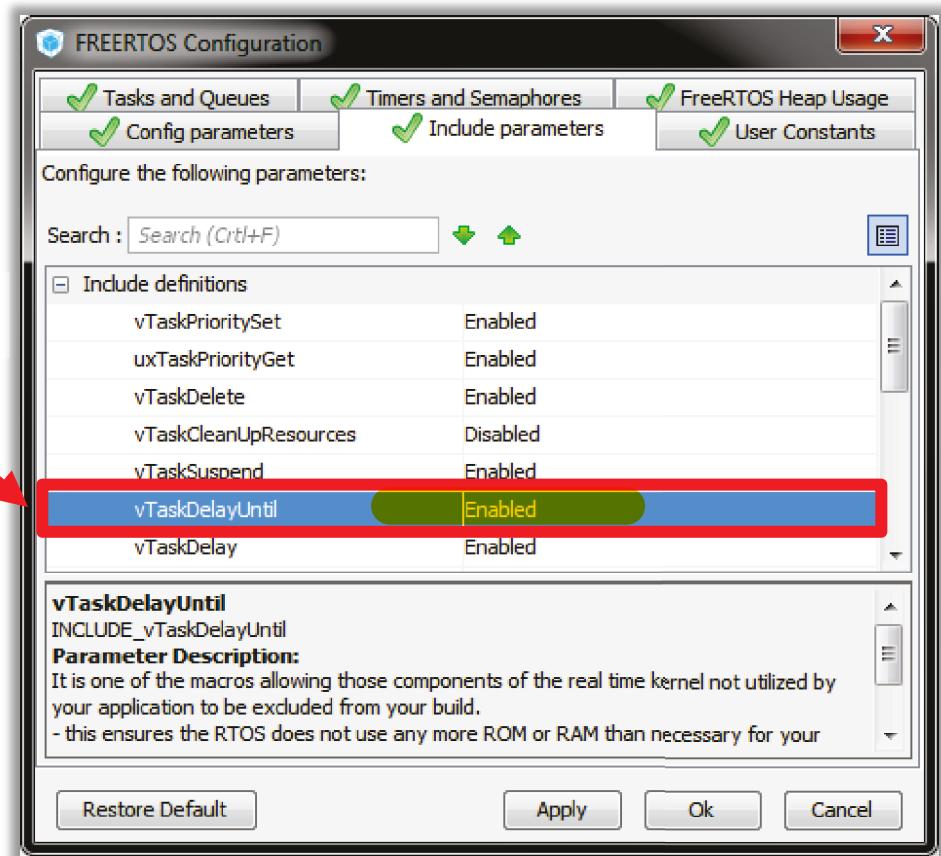
- osDelay start measure time from osDelay call
- osDelayUntil measure time from point which we selected
- This allow us to call task in regular intervals



osDelay and osDelayUntil

- Enable “vTaskDelayUntil” in “**Include parameters**”

Enable “vTaskDelayUntil”





osDelay and osDelayUntil

56

- Only 2 tasks will be used.,.
- Regenerate project, modify tasks to:

```
void StartTask1(void const * argument)
{
    /* USER CODE BEGIN StartTask1 */
    uint32_t lastwakeuptime = osKernelSysTick();
    /* Infinite loop */
    for (;;) {
        printf("Task 1 %d\n", lastwakeuptime);
        HAL_Delay(1000);
        osDelayUntil(&lastwakeuptime, 2000);
    }
    /* USER CODE END StartTask1 */
}
```

**For “osDelayUntil” function,
We need mark startup time**

Time from which is delay measured

... Function will be executed every 2s.

osDelay and osDelayUntil

- Only 2 tasks will used,.
- Regenerate project, modify tasks to:

```
void StartTask2(void const * argument)
{
    /* USER CODE BEGIN StartTask2 */
    uint32_t lastwakeupt;
    /* Infinite loop */
    for (;;) {
        lastwakeupt = osKernelSysTick();
        printf("Task 2 %d\n", lastwakeupt );
        HAL_Delay(1000);
        osDelay(2000);
    }
    /* USER CODE END StartTask2 */

}
```

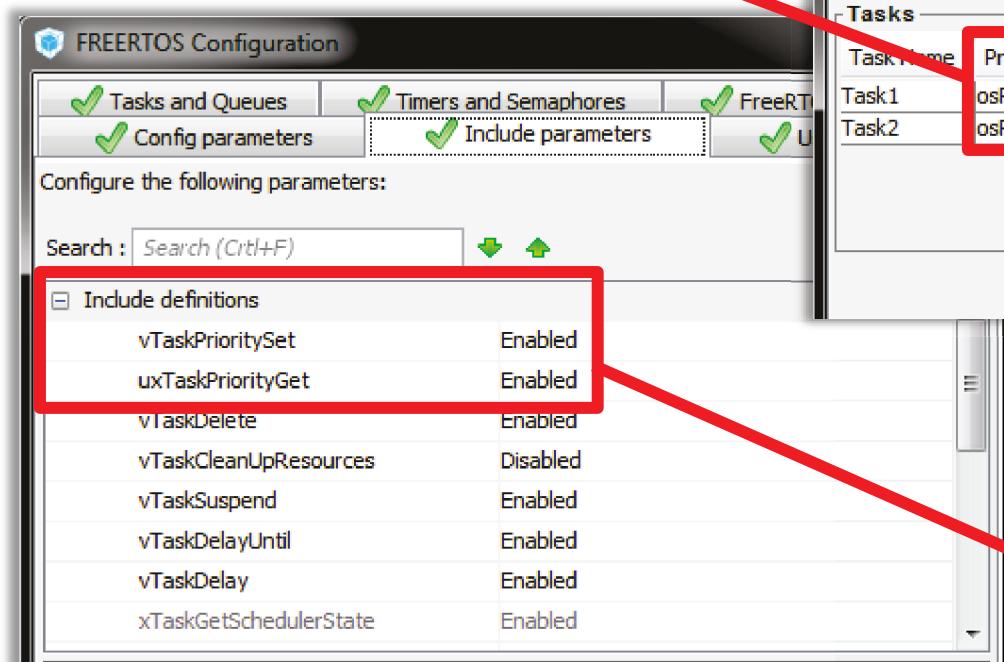
...How long does this function execute per loop?

Priority change lab

58

- Task1 have higher priority than Task2
- Enable 'vTaskPriorityGet' and 'uxTaskPrioritySet' in '**Include Parameters**'

Task1: osPriorityRealtime
Task2: osPriorityNormal



The screenshot shows the 'Tasks' configuration window from the ST-Link Freertos Configuration tool. It lists Task1 with 'osPriorityRealtime' and Task2 with 'osPriorityNormal'. A red box highlights the 'Priority' column for both tasks. The window also includes tabs for 'User Constants', 'Tasks and Queues', and 'Timers and Semaphores'.

Task Name	Priority	Stack Size...	Entry Function	Code Gener...
Task1	osPriorityRealtime	128	StartTask1	Default
Task2	osPriorityNormal	128	StartTask2	Default

vTaskPriorityGet
uxTaskPrioritySet

Priority change lab

59

- Modify Task1 to:

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

Read Task2 priority

Increase Task2 priority

Priority change lab

60

- Modify Task2 to:

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

Read priority of current task



Decrease task priority

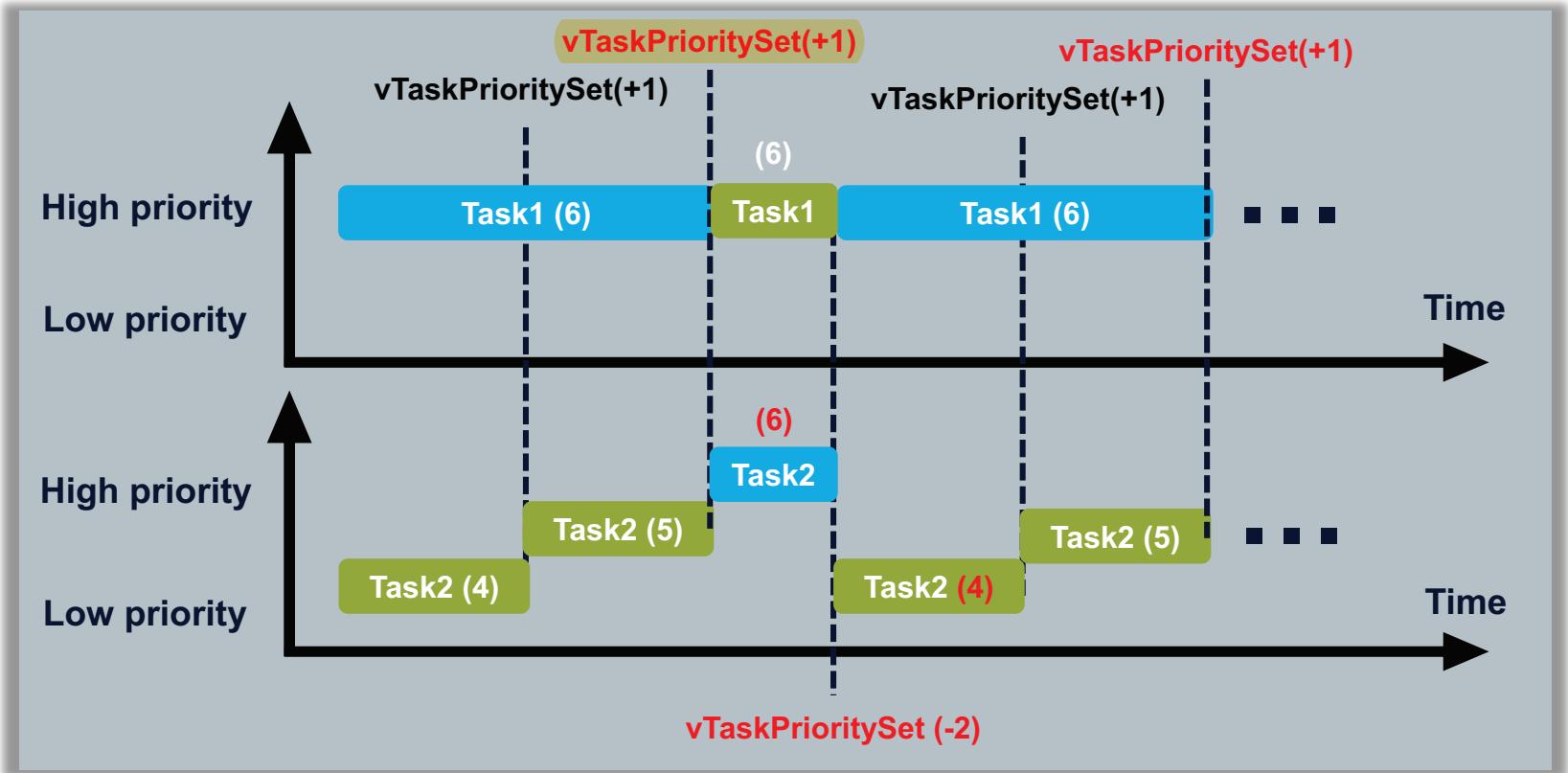


Priority change lab

61

- How priorities are changed?

Running Ready



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



Task2 creation

Creating and deleting tasks lab

63

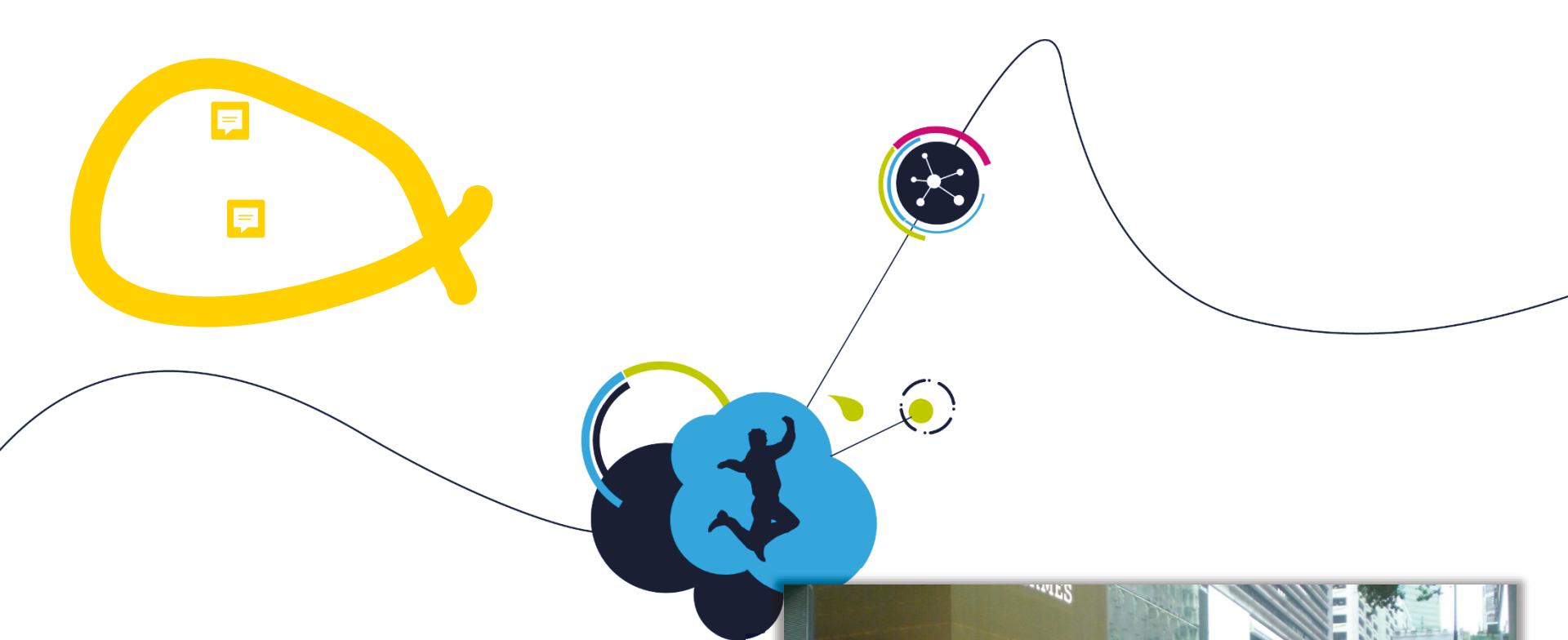
- Example how to create and delete tasks
- Modify Task2 to delete him-self:

```
/* 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

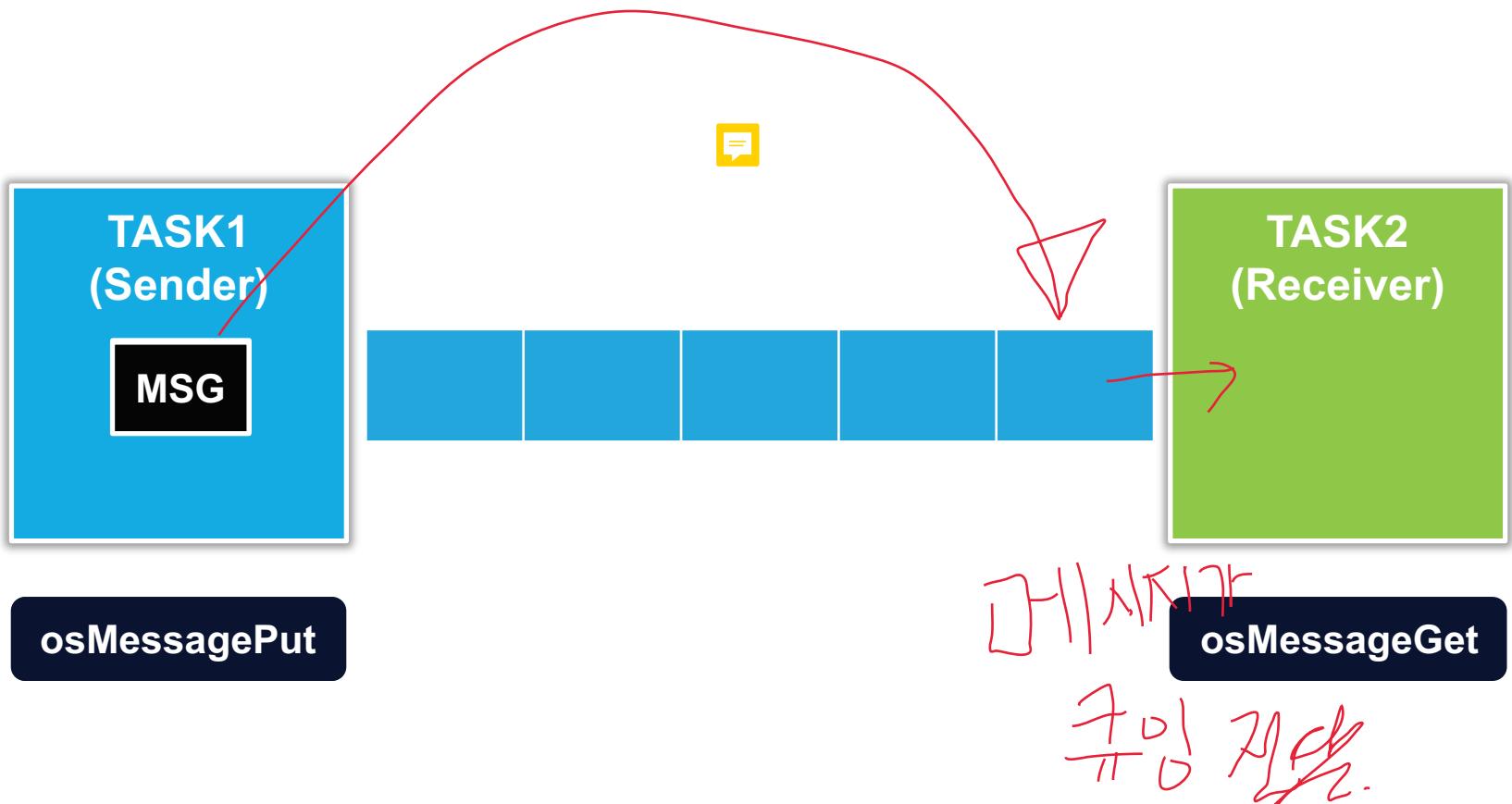


FreeRTOS Queues



Queue

65



- Create Queue:

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

- Put data into Queue

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

- Receive data from Queue

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

Item to send



Structure with status and with received item

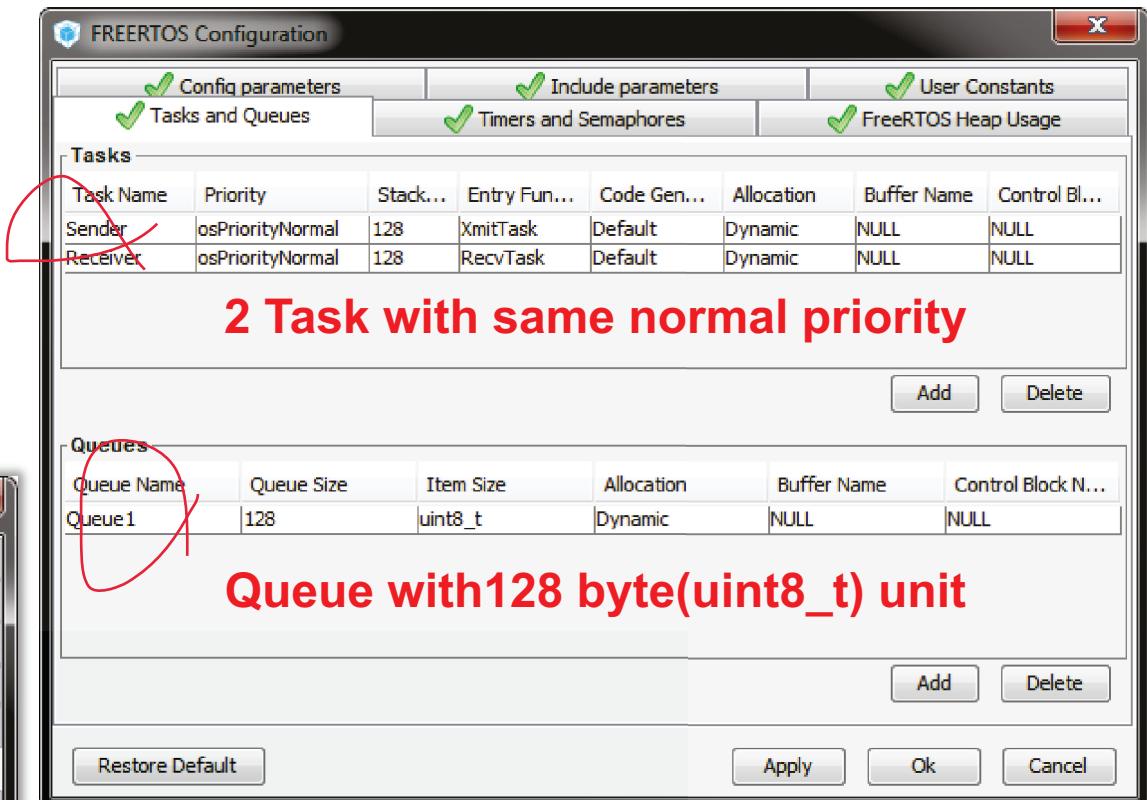
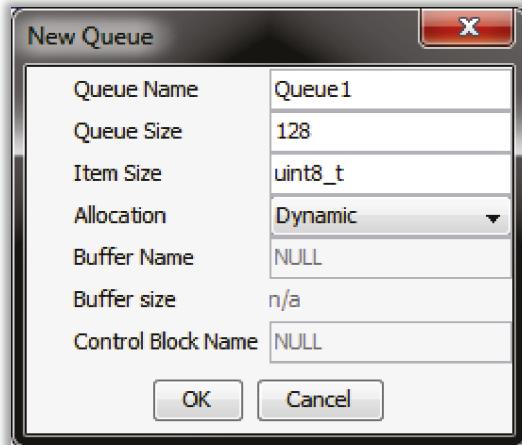
- osEvent structure

```
typedef struct {  
    osStatus           status;      ///    information  
    union {  
        uint32_t         v;          ///        void             *p;          ///        int32_t          signals;    ///    } value;  
    union {  
        osMailQId       mail_id;    ///        osMessageQId    message_id; ///        osMessageCreate  
    } def;                      ///} 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 data-type

- Set both tasks (Sender, Receiver) to normal priority
 - XmitTask / RecvTask
- Queue part
 - Button 'Add'
 - Set queue size to '128'
 - Queue type to 'uint8_t'
- Button 'OK'



- Queue handler is now defined

```
/* Private variables -----  
--- */  
osThreadId SenderHandle;  
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, 128, uint8_t);  
Queue1Handle = osMessageCreate(osMessageQ(Queue1), NULL);
```

- Sender task

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

Put value '0xAA' into queue

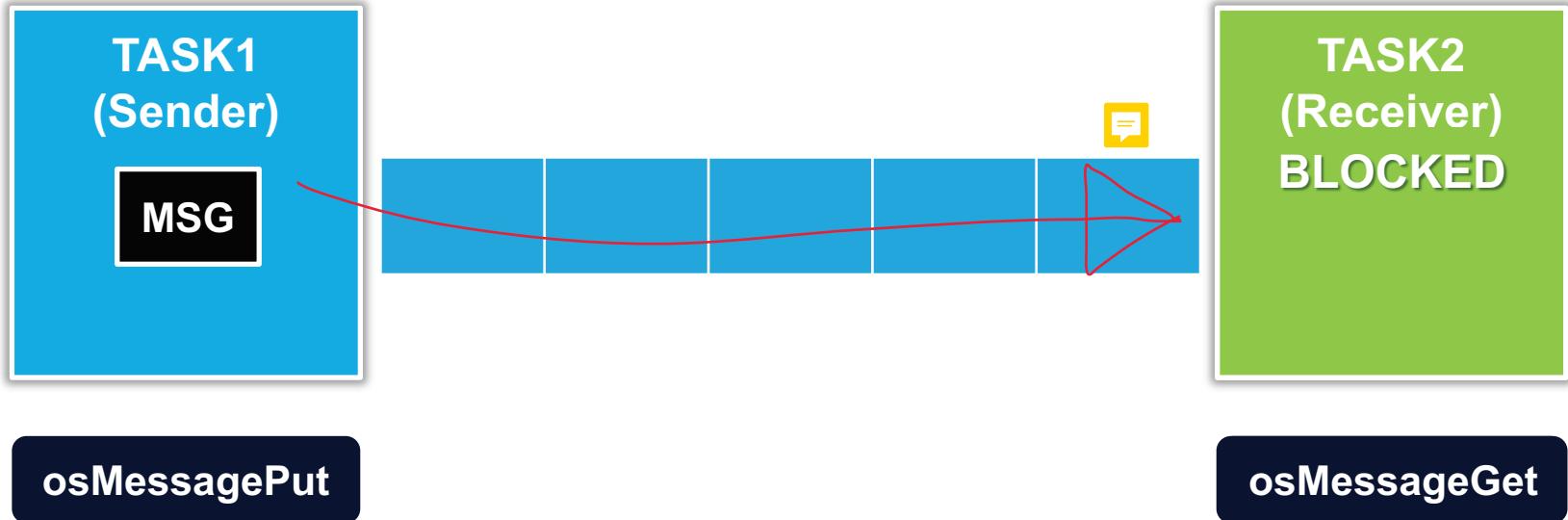
- Receiver task

```
/* RcvTask function */
void RcvTask(void const * argument)
{
    /* USER CODE BEGIN RcvTask */
    osEvent retval;
    /* Infinite loop */
    for (;;) {
        printf("Receiver\n");
        retval = osMessageGet(Queue1Handle, 1000);
        printf("%d\n", retval.value.v);
    }
    /* USER CODE END RcvTask */
}
```

Queue Blocking

72

- After calling ‘osMessageGet’
- If any data are not in queue, the task is blocked for configured time
- If the data are in queue, the task will continue



Two senders lab

73

- Two sending tasks
- One receivers tasks
- Same priorities

The screenshot shows the FreeRTOS Configuration tool interface. At the top, there are three tabs: 'Config parameters' (selected), 'Tasks and Queues' (highlighted with a red dashed box), and 'Timers and Semaphores'. Below the tabs, there are two main sections: 'Tasks' and 'Queues'. The 'Tasks' section contains a table with four columns: Task Name, Priority, Stack ..., and Entry Fun... . It lists four tasks: Sender1, Receiver, Sender2, and another Sender1. All tasks have 'osPriorityNormal' priority and a stack size of 128. Their entry functions are StartSender1, StartReceiver, StartSender2, and StartSender1 respectively. The 'Queues' section contains a table with five columns: Queue Name, Queue Size, Item Size, Allocation, Buffer Name, and Control Block N... . It lists one queue named 'Queue1' with a size of 128, item type uint8_t, dynamic allocation, and no buffer or control block name specified. At the bottom of the window are buttons for 'Restore Default', 'Apply', 'Ok', and 'Cancel'.

Task Name	Priority	Stack ...	Entry Fun...
Sender1	osPriorityNormal	128	StartSender1
Receiver	osPriorityNormal	128	StartReceiver
Sender2	osPriorityNormal	128	StartSender2

Queue Name	Queue Size	Item Size	Allocation	Buffer Name	Control Block N...
Queue1	128	uint8_t	Dynamic	NULL	NULL

Two senders lab

74

- Two sending tasks
- They are same no change necessary

```
void StartSender1(void const *  
argument)  
{  
    /* USER CODE BEGIN StartSender1 */  
    /* Infinite loop */  
    for (;;) {  
        printf("Sender1\n");  
  
        osMessagePut(Queue1Handle, 0x1, 200);  
        printf("Sender1 delay\n");  
        osDelay(2000);  
    }  
    /* USER CODE END StartSender1 */  
}
```

```
void StartSender2(void const *  
argument)  
{  
    /* USER CODE BEGIN StartSender2 */  
    /* Infinite loop */  
    for (;;) {  
        printf("Sender2\n");  
  
        osMessagePut(Queue1Handle, 0x2, 200);  
        printf("Sender2 delay\n");  
        osDelay(2000);  
    }  
    /* USER CODE END StartSender2 */  
}
```

Two senders lab

75

- Simple receiver

```
/* StartReceiver function */
void StartReceiver(void const * argument)
{
    /* USER CODE BEGIN StartReceiver */
    osEvent retval;
    /* Infinite loop */
    for (;;) {
        retval = osMessageGet(Queue1Handle,
1000);
        printf("Receiver %d\n", retval.value.v);
    }
    /* USER CODE END StartReceiver */
}
```

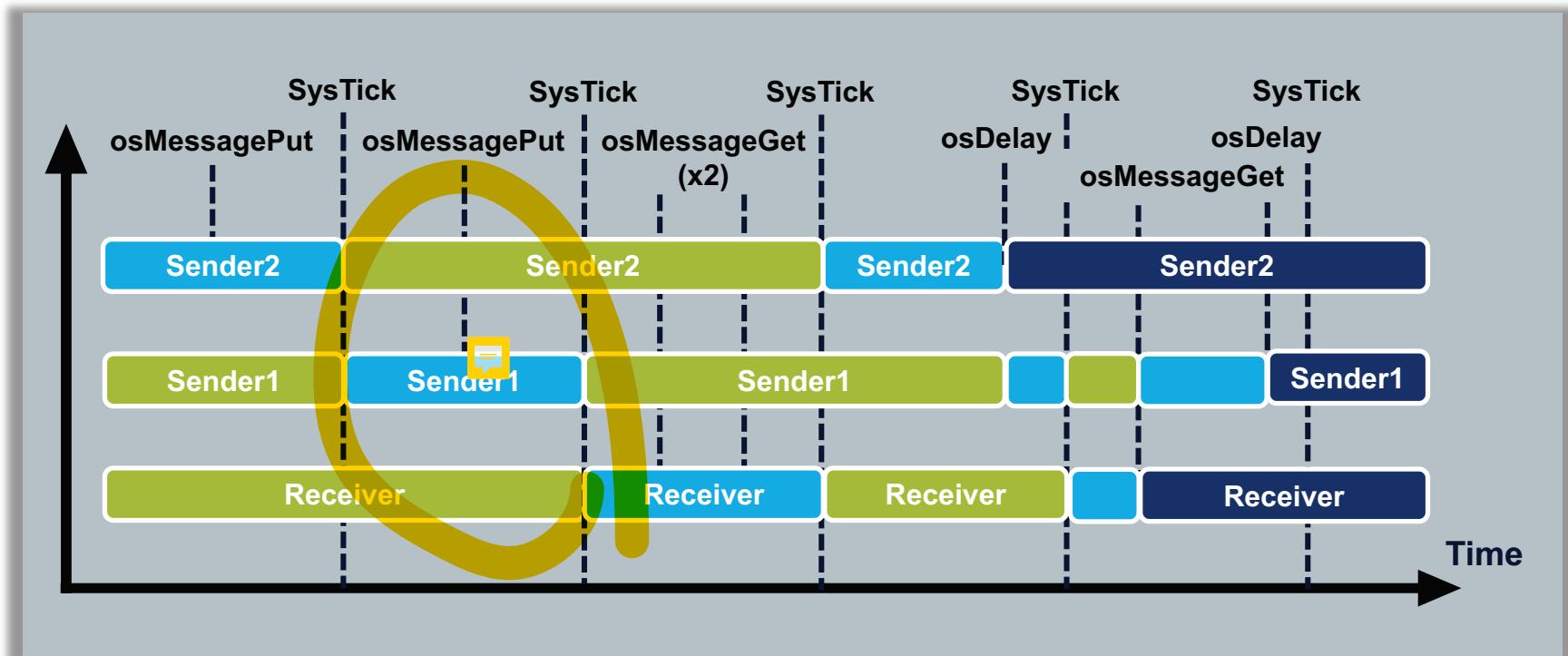


Queue Blocking

76

- 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 tasks?

■ Running ■ Blocked ■ Ready



Receiver with higher priority lab

77

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

The screenshot shows the FreeRTOS Configuration tool interface. The main window has tabs for 'Config parameters', 'Include parameters', 'User Constants', 'Tasks and Queues' (selected), 'Timers and Semaphores', and 'FreeRTOS Heap Usage'. The 'Tasks' section displays three tasks:

Task Name	Priority	Stac...	Entry Function	Code...	Allocation	Buff...	Control Bl...
Sender1	osPriorityNormal	128	StartSender1	Default	Dynamic	NULL	NULL
Receiver	osPriorityAboveNormal	128	StartReceiver	Default	Dynamic	NULL	NULL
Sender2	osPriorityNormal	128	StartSender2	Default	Dynamic	NULL	NULL

At the bottom, there are 'Add' and 'Delete' buttons. The 'Queues' section shows one queue:

Queue Name	Queue Size	Item Size
Queue1	128	uint8_t

A red box highlights the 'Tasks' section of the main window. A red dashed line connects this box to a secondary 'Tasks' window at the bottom. This secondary window also lists three tasks:

Task Name	Priority	Stac...	Entry Function
Sender1	osPriorityNormal	128	StartSender1
Receiver	osPriorityAboveNormal	128	StartReceiver
Sender2	osPriorityNormal	128	StartSender2

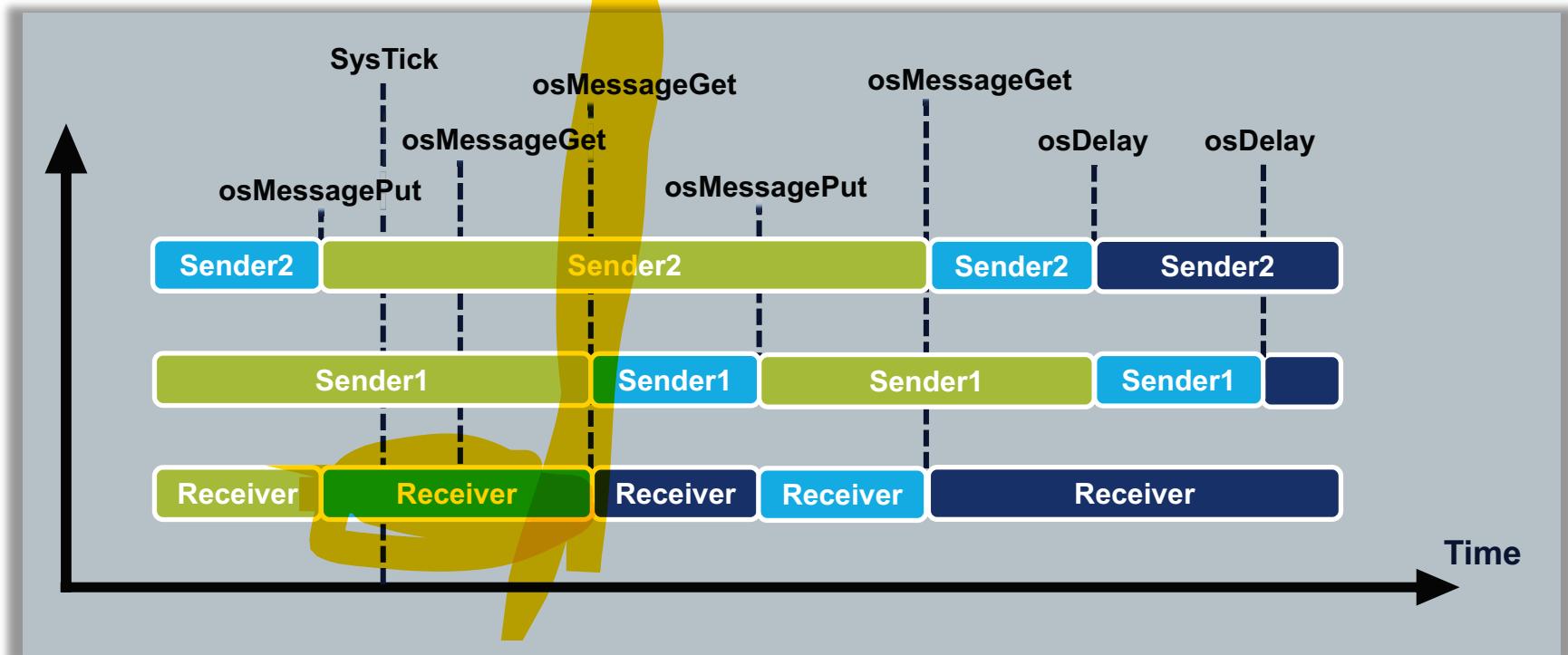
Buttons at the bottom of the secondary window include 'Restore Default', 'Apply', 'OK', and 'Cancel'.

Queue Blocking

78

- 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 tasks?

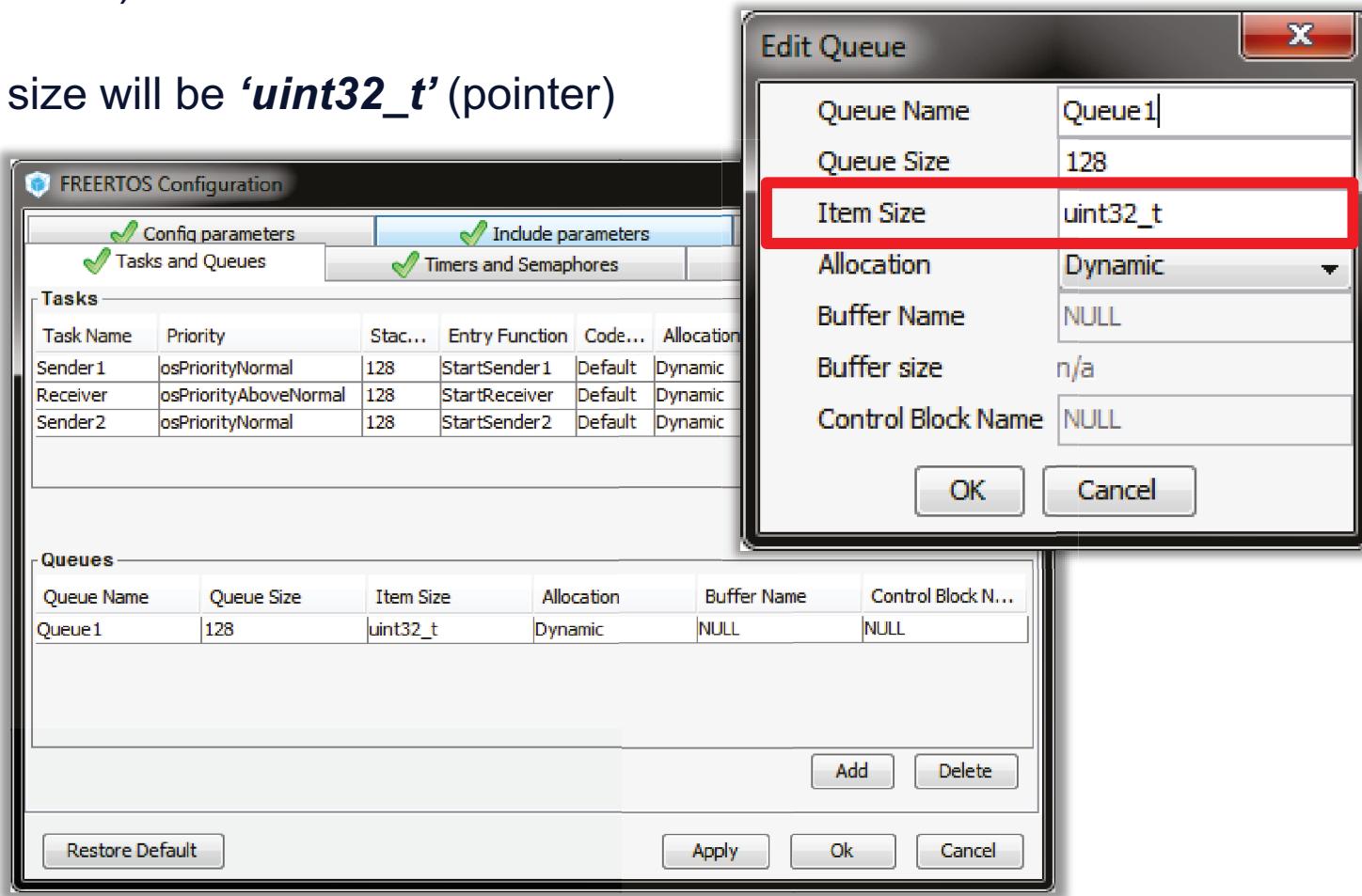
■ Running ■ Blocked ■ Ready



Queue items lab

79

- Queues allow to define ‘type’ (different variables or structures which the queue use)
- Item size will be ‘*uint32_t*’ (pointer)



Queue items lab

80

- Create new structure type for data

```
/* Define string literal type that will be passed on the queue.  
 */  
const char *type1str = "Hello, World!";
```

- Sent data from Sender task

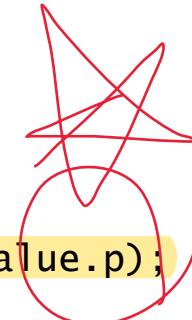
```
void StartSender1(void const * argument)  
{  
    /* USER CODE BEGIN 5 */  
    /* Infinite loop */  
    for (;;) {  
        printf("Task1\n");  
        osMessagePut(Queue1Handle, (uint32_t)&type1str ,200);  
        printf("Task1 delay\n");  
        osDelay(2000);  
    }  
    /* USER CODE END 5 */  
}
```

Queue items lab

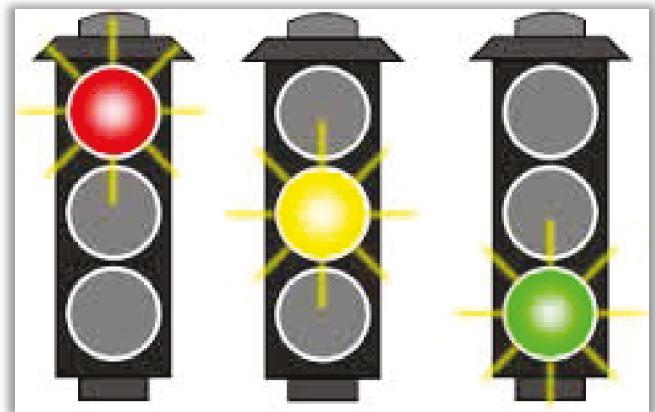
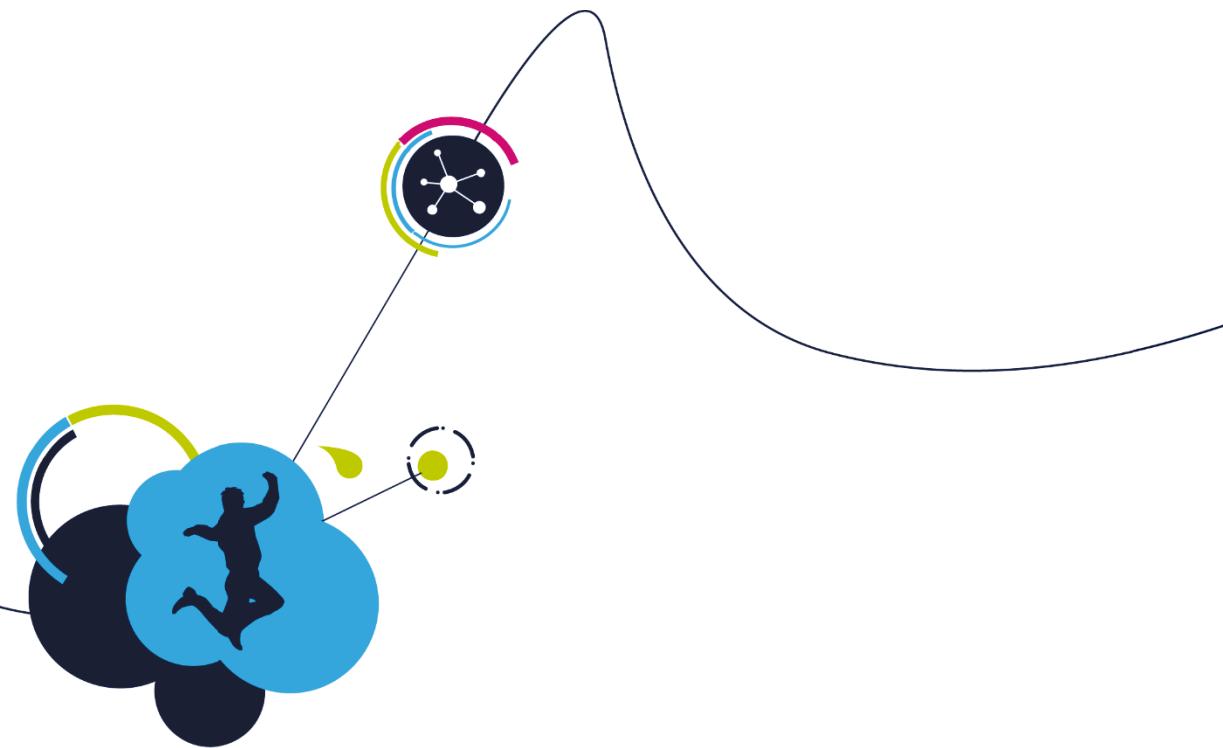
81

- Receiver data from sender task

```
/* StartReceiver function */
void StartReceiver(void const * argument)
{
    /* USER CODE BEGIN StartReceiver */
    osEvent retval;
    /* Infinite loop */
    for (;;) {
        retval = osMessageGet(Queue1Handle, 4000);
        if (retval.value.p) {
            printf("Receive: %s\n", (const char *)retval.value.p);
        } else {
            printf("Receive nothing\n");
        }
    }
    /* USER CODE END StartReceiver */
}
```



FreeRTOS Semaphores



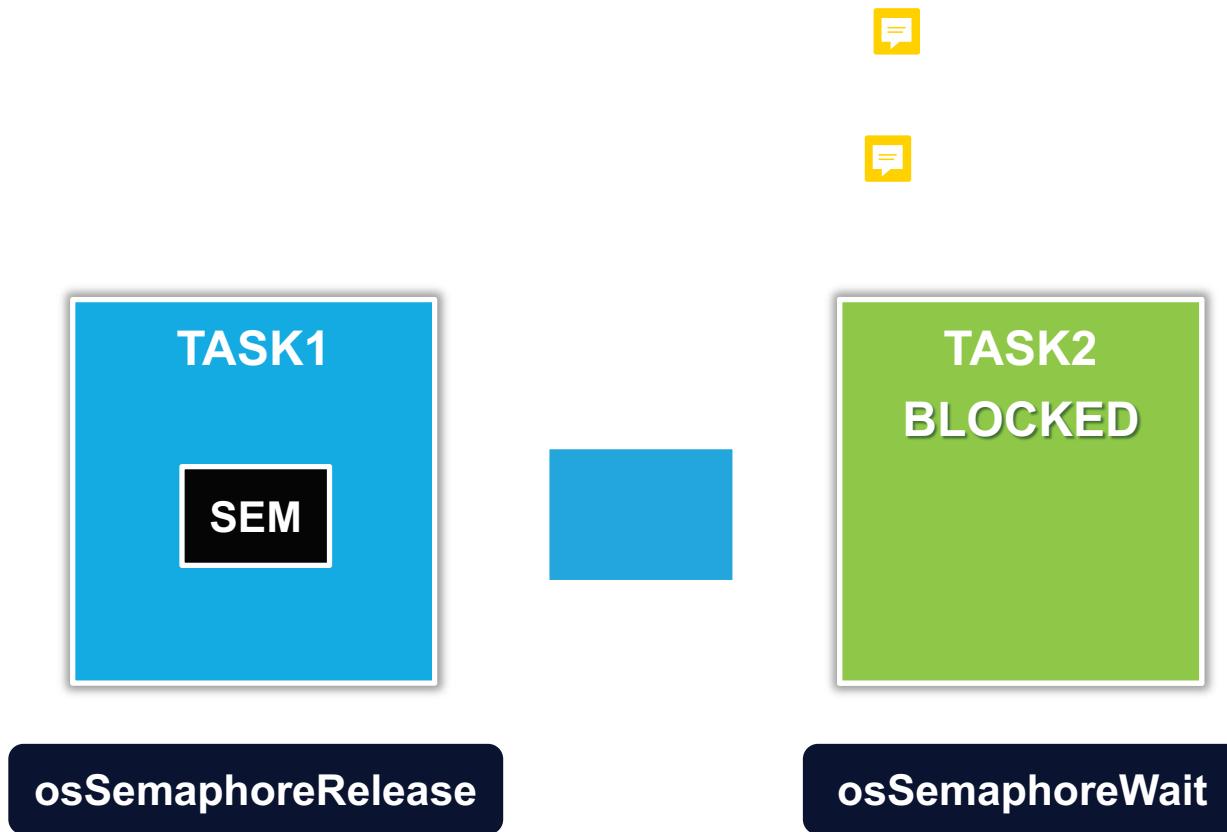
- Used for synchronization between
 - Tasks
 - Interrupt and task
- Two types
 - Binary semaphores
 - Counting semaphores

- **Binary semaphore**
 - Have only one ‘token’
 - Using to synchronize one action
- **Counting semaphore**
 - Have multiple ‘tokens’
 - Synchronize multiple actions



Binary Semaphore

84



Binary Semaphore

85

- Semaphore creation

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

**Semaphore ‘tokens’
for binary semaphore is 1.**

- Wait for Semaphore release

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

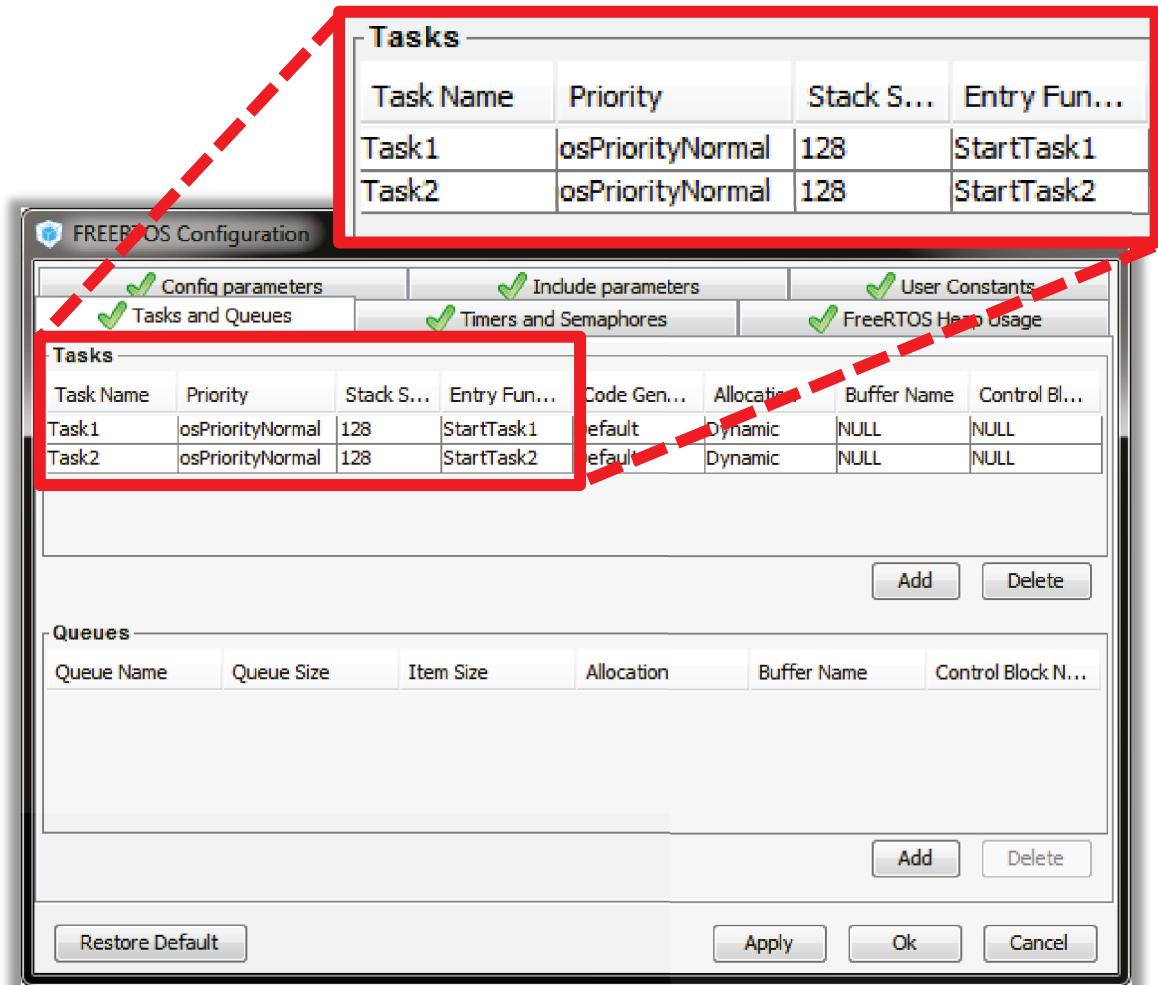
Number of ‘tokens’ in semaphore

- Semaphore release

```
osStatus osSemaphoreRelease(osSemaphoreId sema_id)
```

Binary Semaphore lab

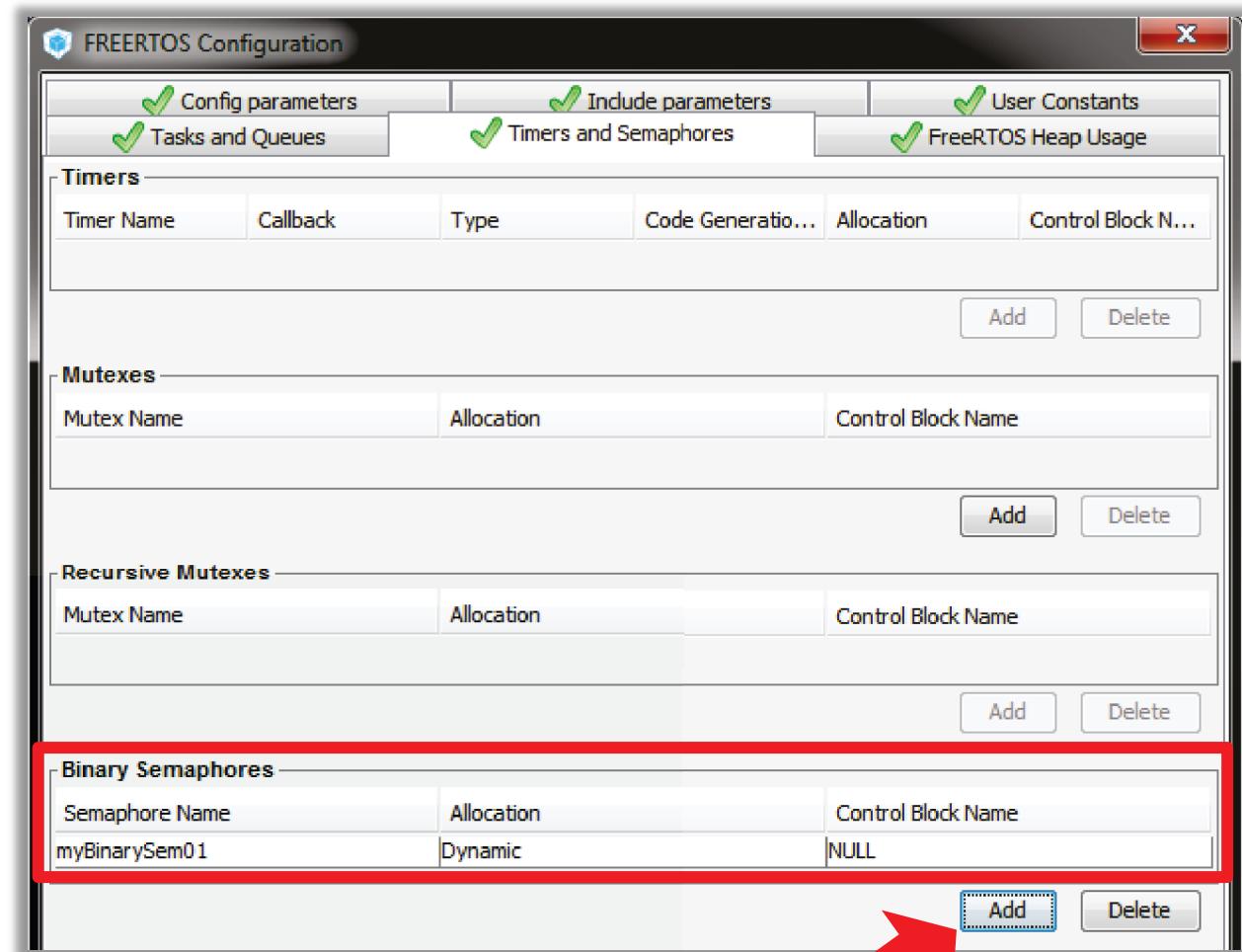
- Create two tasks
- With same priority
- Button Add
- Set parameters
- Button OK



Binary Semaphore lab

- Create binary binary semaphore (in Timers and Semaphores)

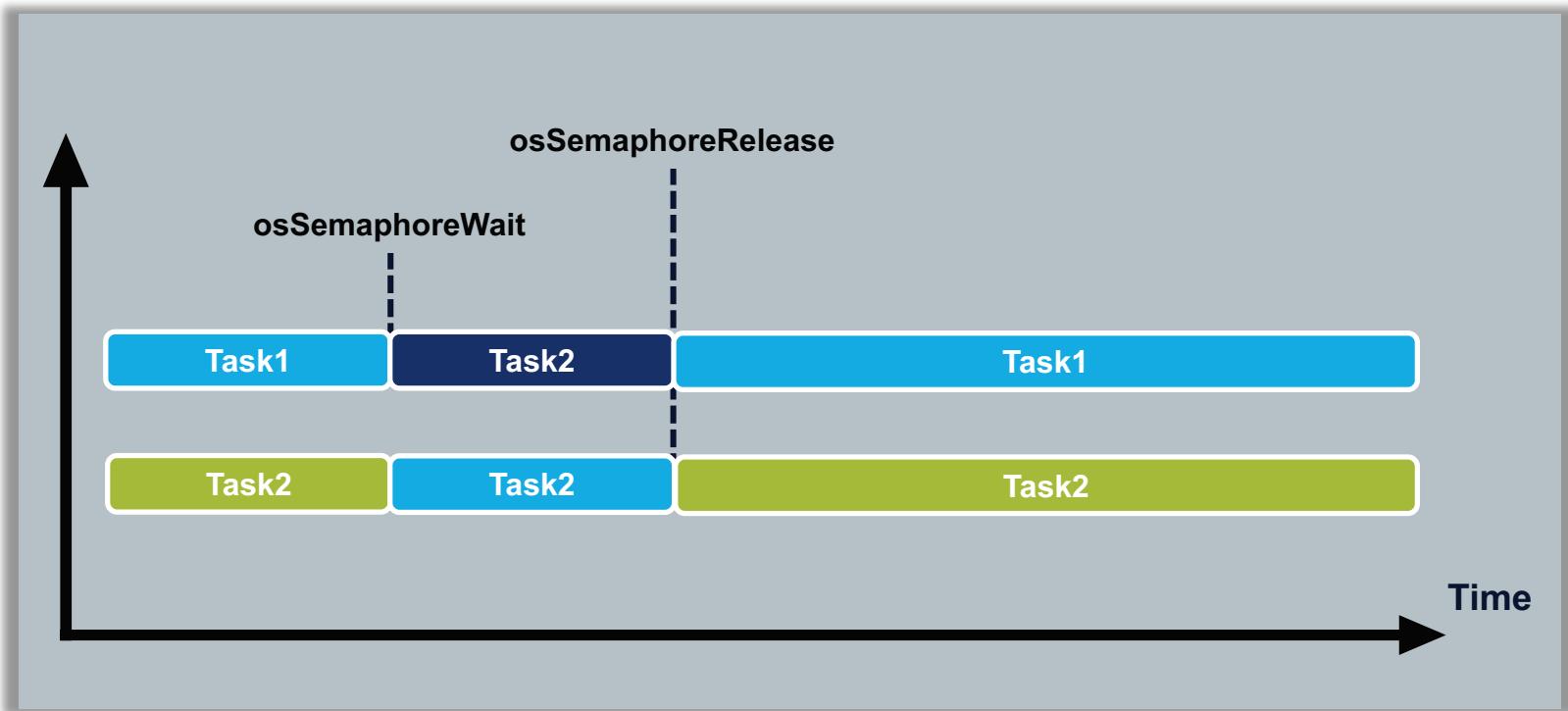
- Button Add
- Set name
- Button OK



Binary Semaphore lab

- Task1 is synchronized with Task2

■ Running ■ Blocked ■ Ready



Binary Semaphore lab

89

- 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

90

- Semaphore use
- If tasks/interrupt is done the semaphore is released

```
void StartTask1(void const * argument)
{
    /* USER CODE BEGIN StartTask1 */
    /* Infinite loop */
    for (;;) {
        osDelay(2000);
        printf("Task1 Release semaphore\n");
        osSemaphoreRelease(myBinarySem01Handle);
    }
    /* USER CODE END StartTask1 */
}
```

Binary Semaphore lab

- Semaphore Wait
- Second task waits on semaphore release
After release task is unblocked and continue in work

```
void StartTask2(void const * argument)
{
    /* USER CODE BEGIN StartTask2 */
    /* Infinite loop */
    for (;;) {
        if (osSemaphoreWait(myBinarySem01Handle, 1000) == osOK) {
            printf("Task2 synchronized\n");
        }
    }
    /* USER CODE END StartTask2 */
}
```

Counting semaphore

92

- OS API same as Binary semaphore
- Semaphore creation

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

- Wait for Semaphore release

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

- Semaphore release

```
osStatus osSemaphoreRelease(osSemaphoreId sema_id)
```

Counting Semaphore

93

osSemaphoreRelease

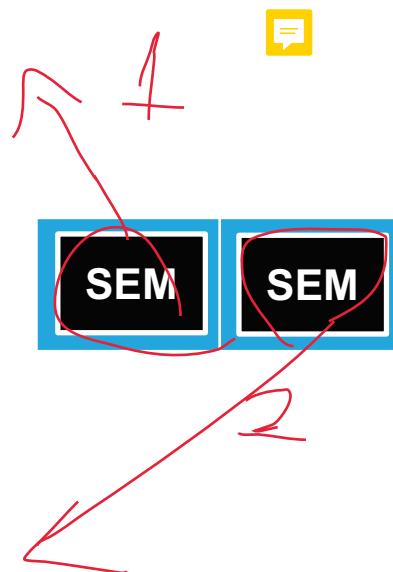
TASK1

TASK3

**TASK2
BLOCKED**

osSemaphoreWait

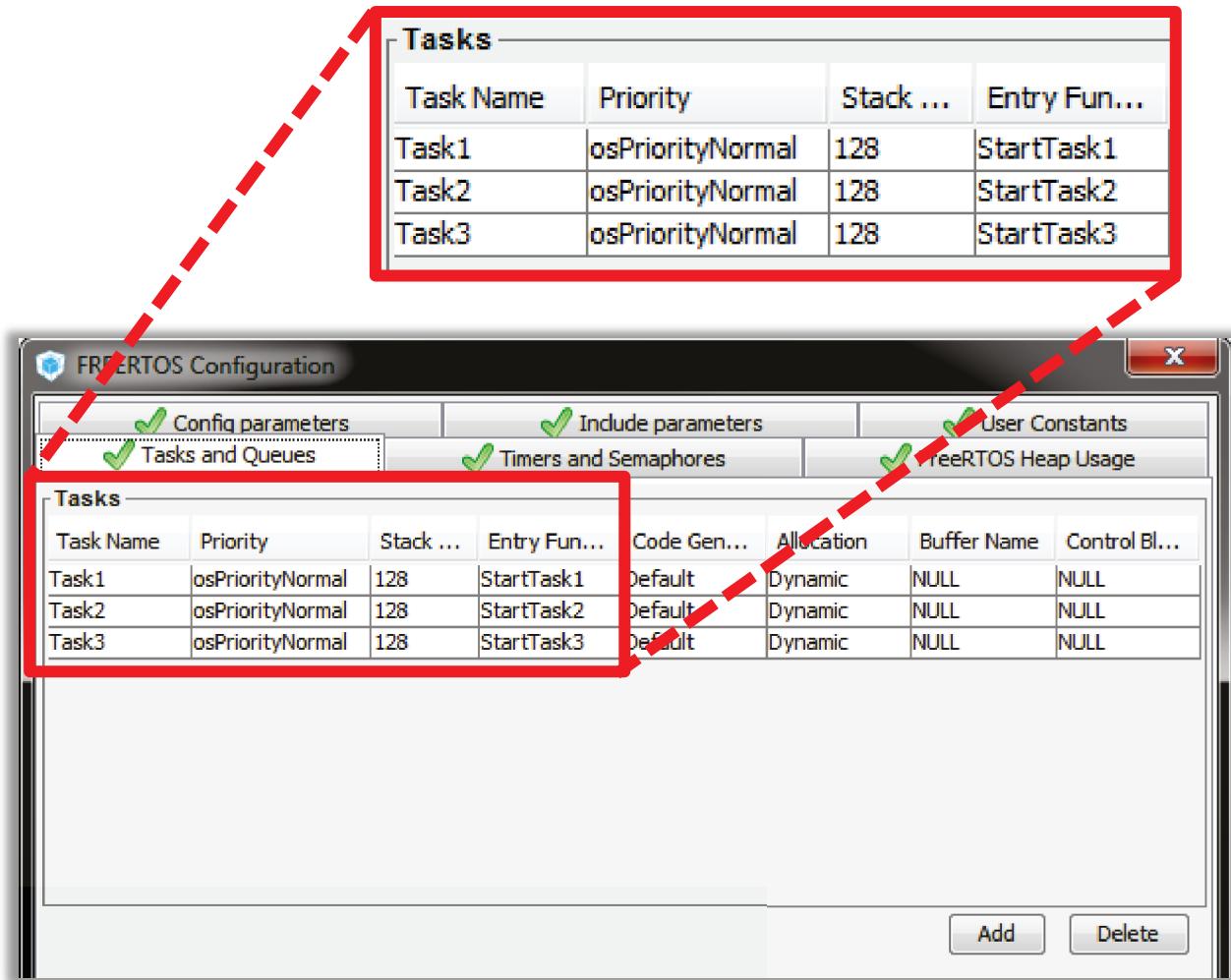
osSemaphoreRelease



Counting Semaphore lab

94

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

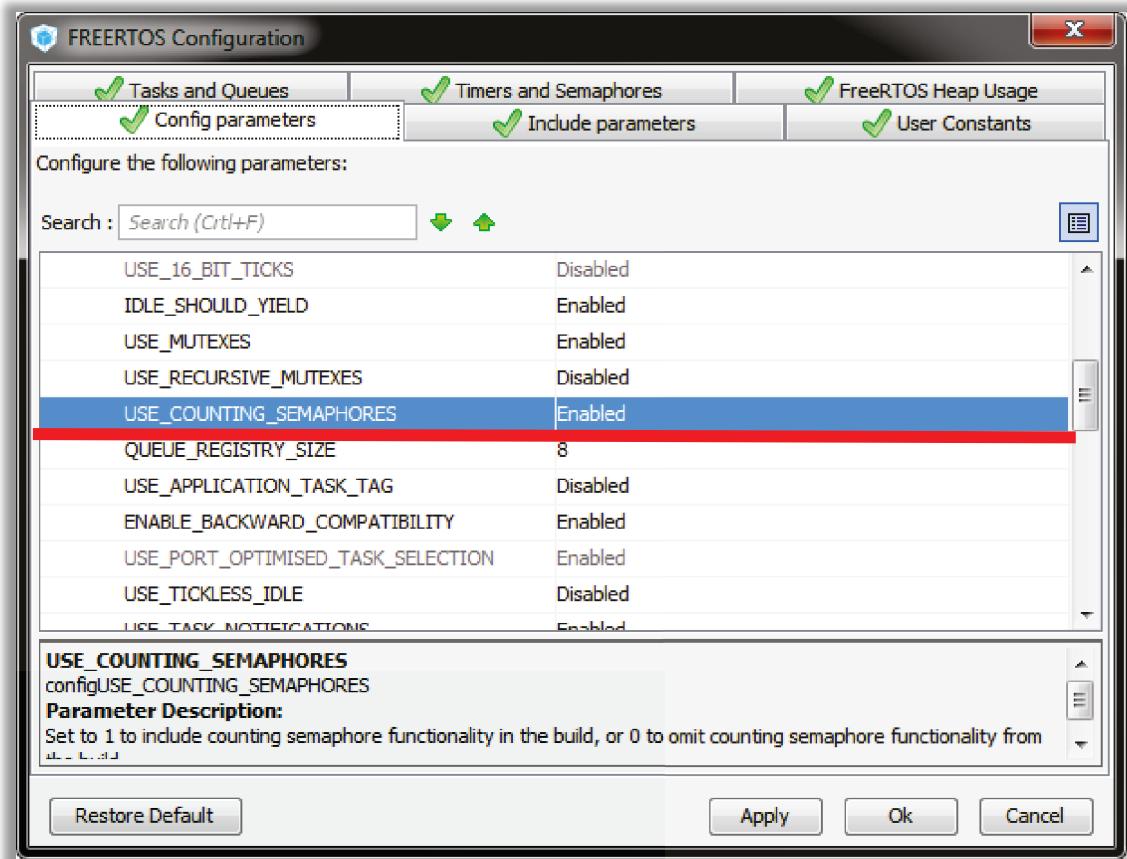


Counting Semaphore lab

95

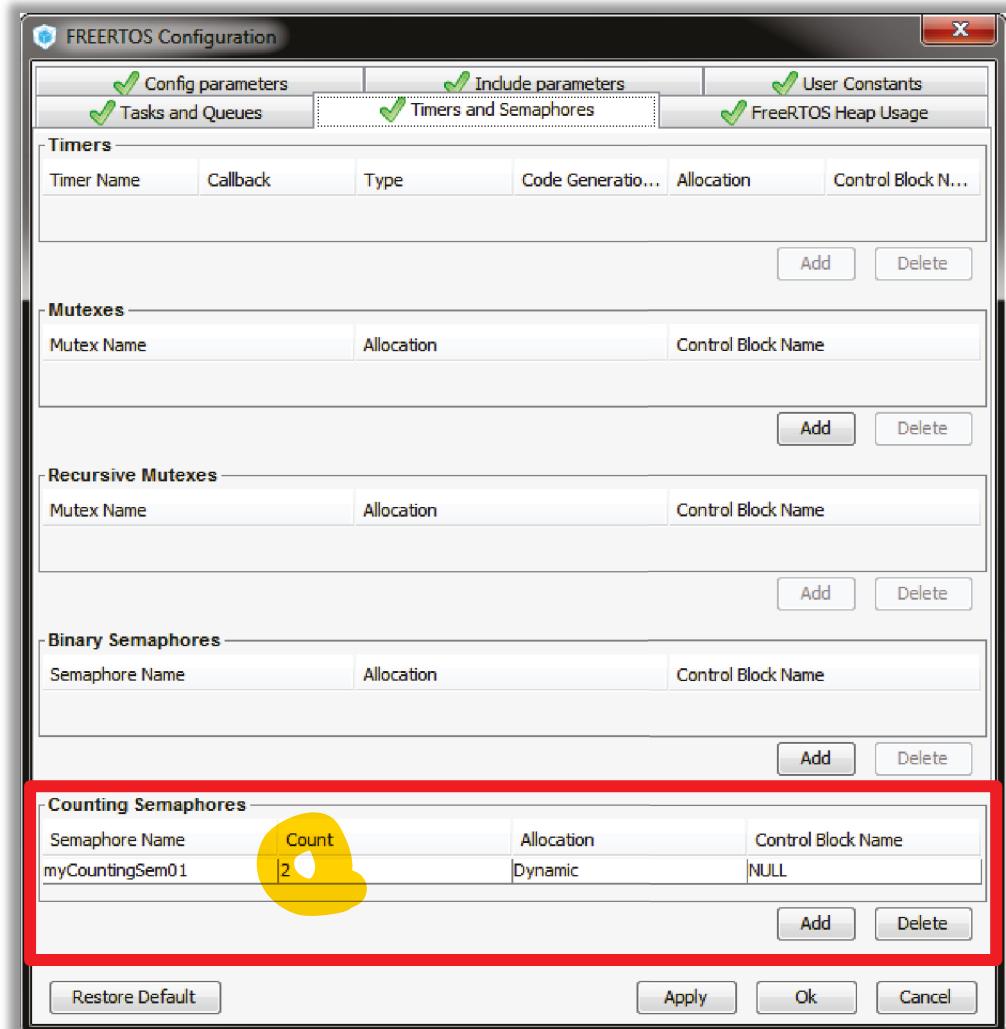
- Config parameters
- Enable kernel settings
- Button OK

USE_COUNTING_SEMAPHORES, Enabled



Counting Semaphore lab

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



Counting Semaphore lab

- Create Counting semaphore

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

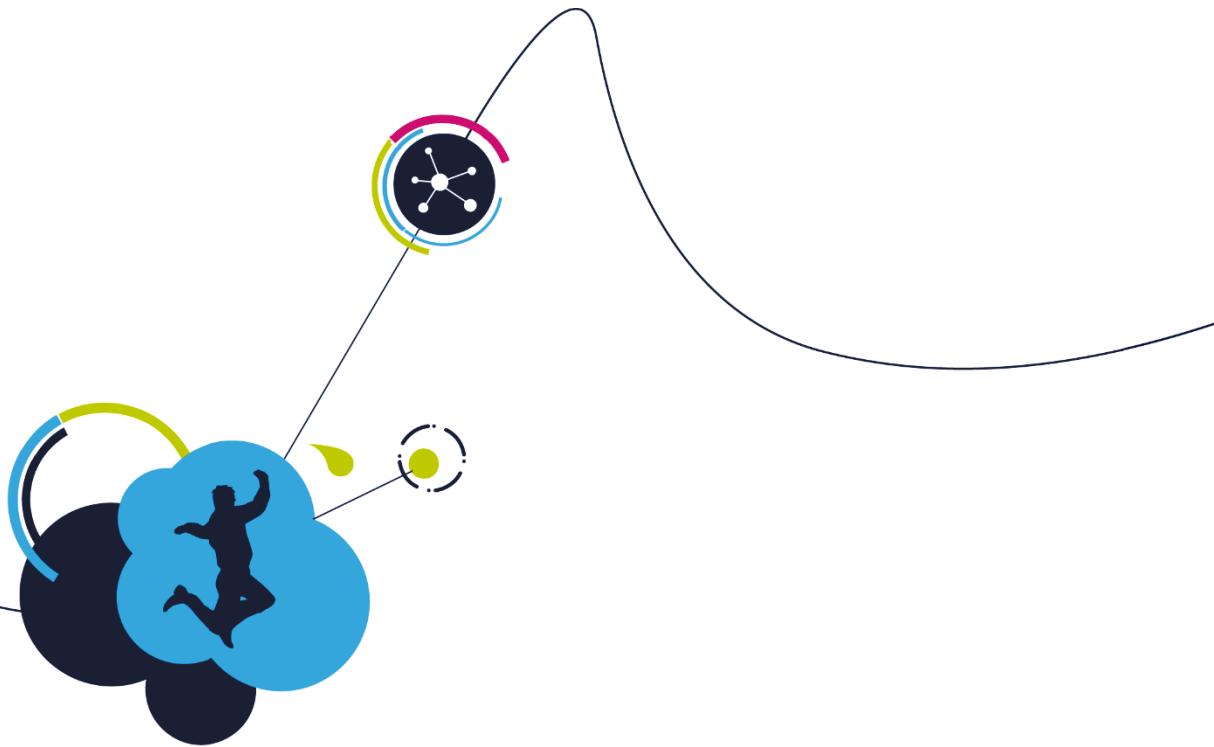
- Task1 and Task2 will be same

```
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 */
}
```

Counting Semaphore lab

- Task3 will wait until semaphore will be 2 times released

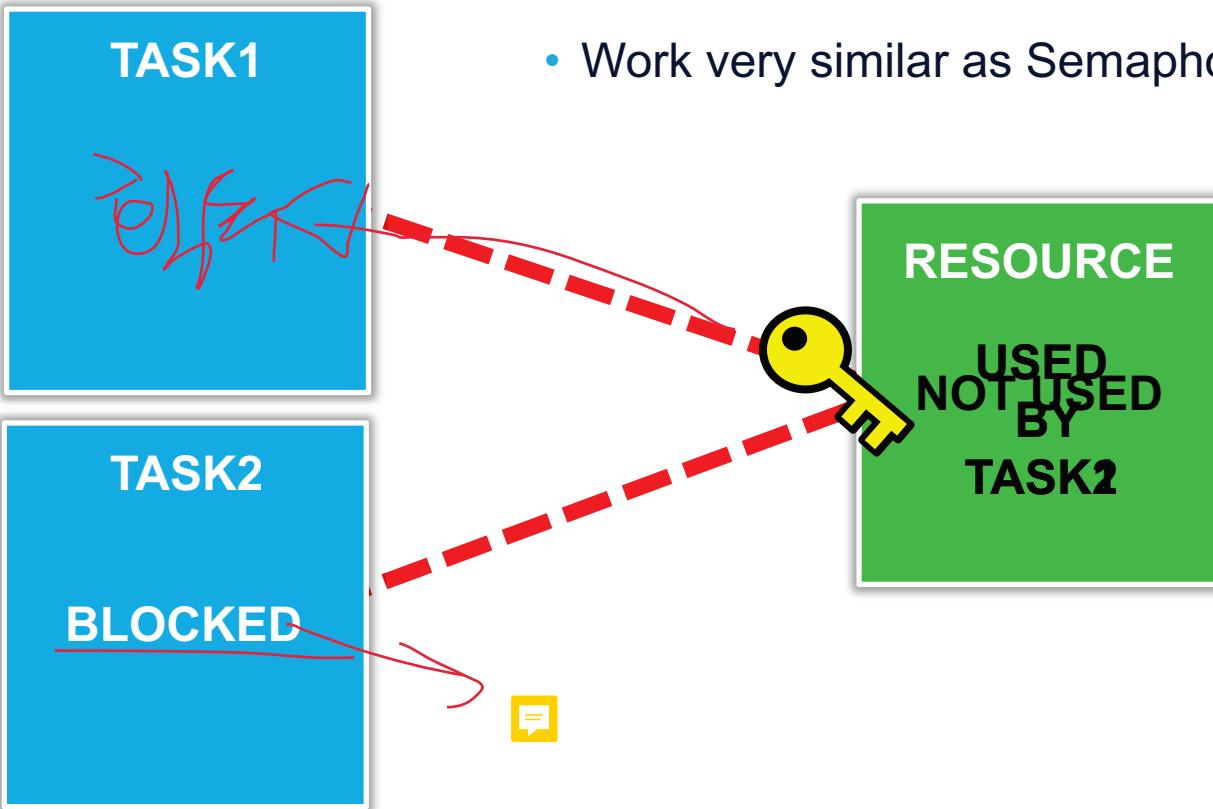
```
void StartTask3(void const * argument)
{
    /* USER CODE BEGIN StartTask3 */
    /* Infinite loop */
    for(;;)
    {
        osSemaphoreWait(myCountingSem01Handle, 4000);
        osSemaphoreWait(myCountingSem01Handle, 4000);
        printf("Task3 synchronized\n");
    }
    /* USER CODE END StartTask3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



FreeRTOS Mutex

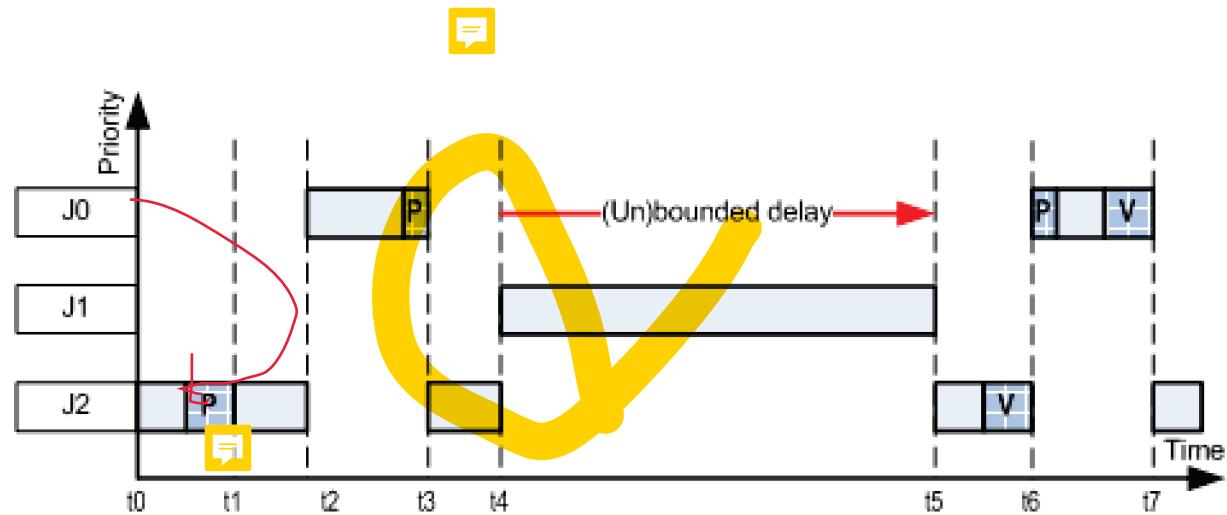
osMutexRelease

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

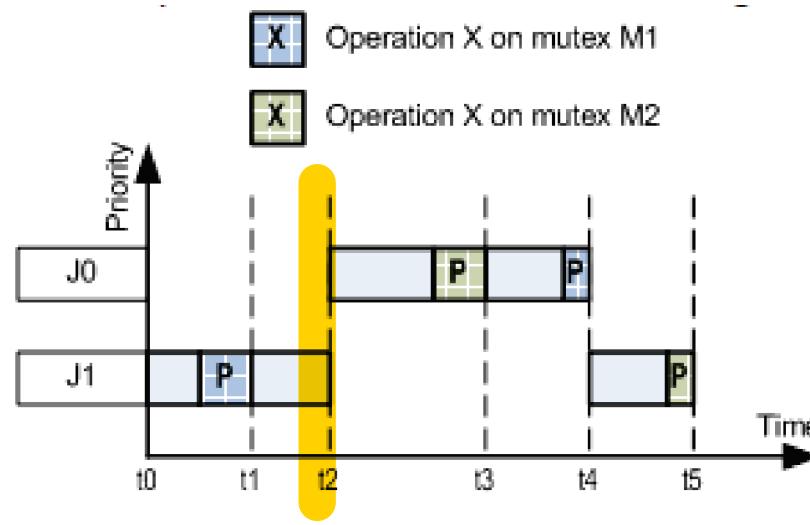


osMutexWait

- Priority Inversion



- Deadlock



- Mutex creation

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

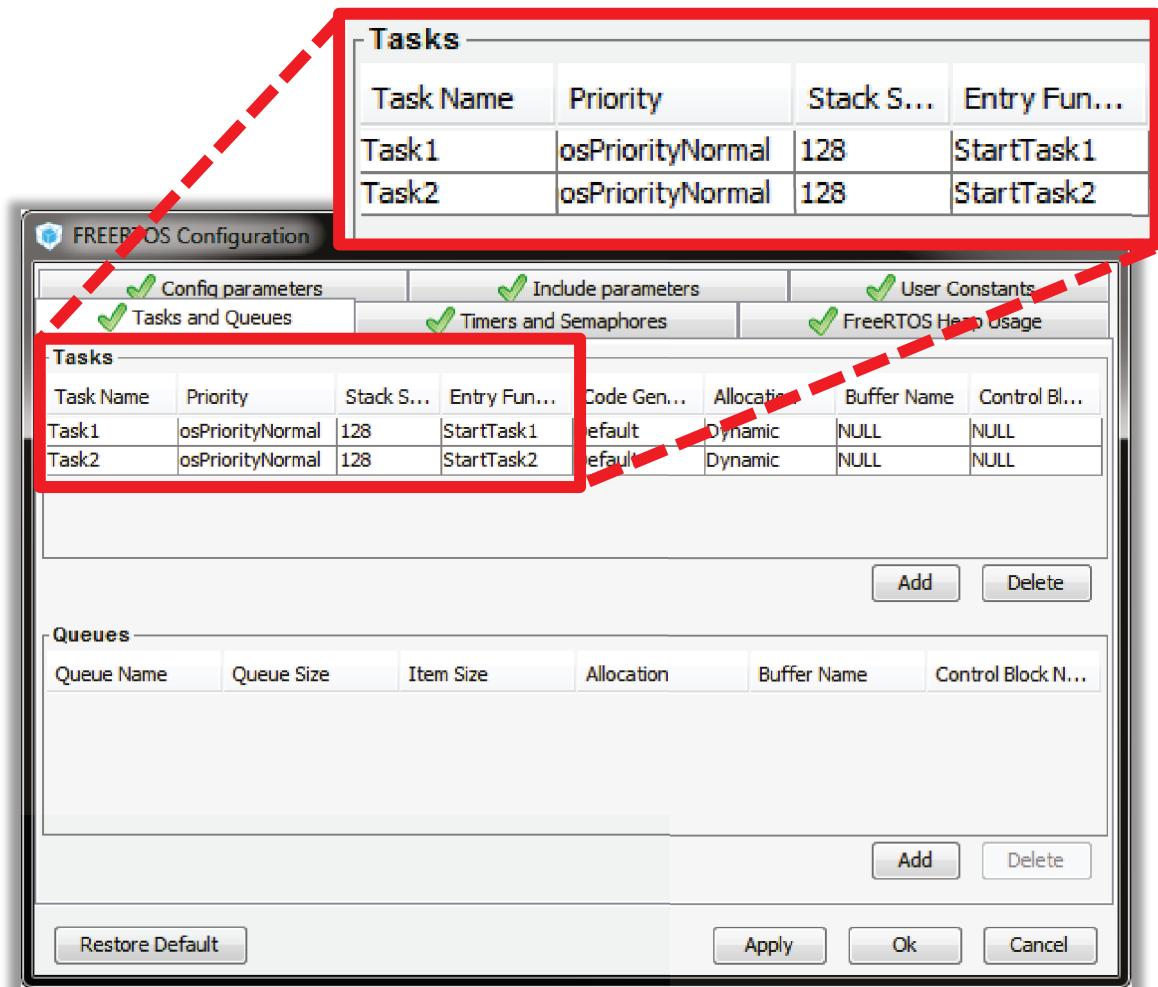
- Wait for Mutex release

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

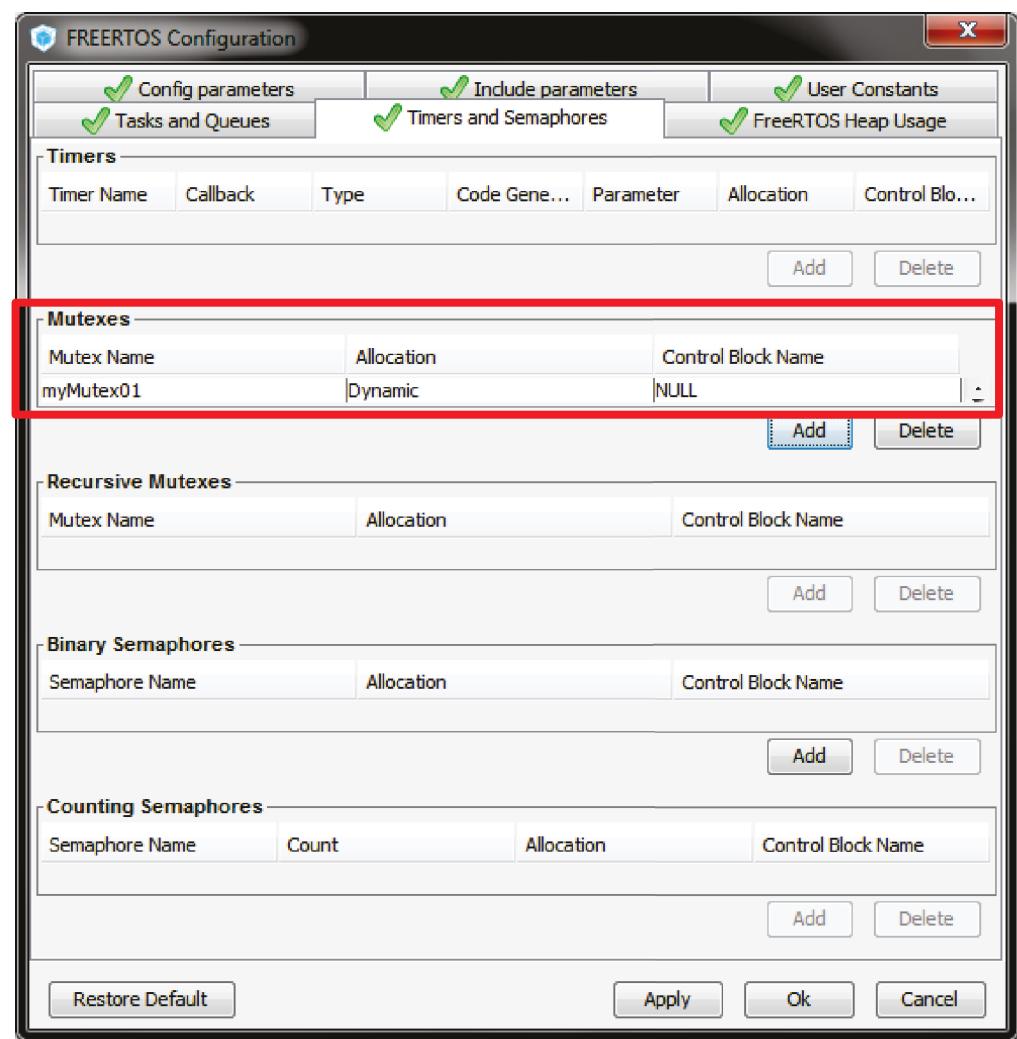
- Mutex release

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

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



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



- Mutex handle definition

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

- Mutex creation

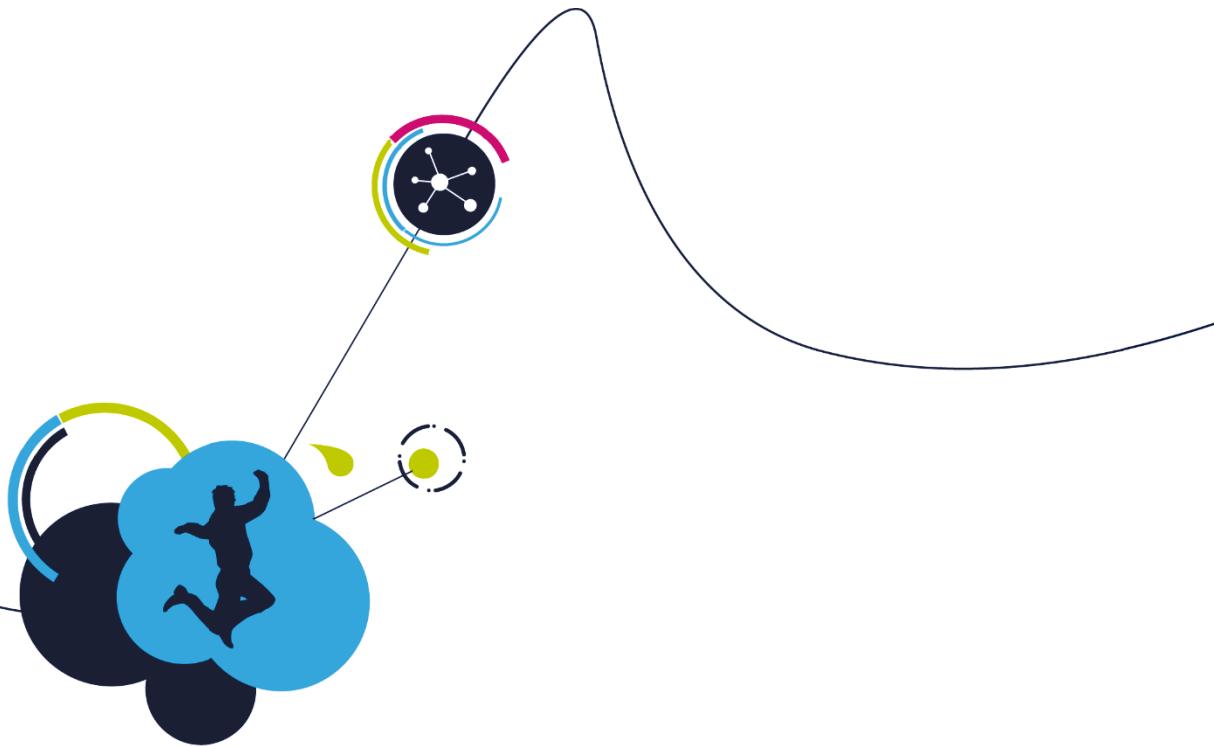
```
/* Create the mutex(es) */  
/* definition and creation of myMutex01 */  
osMutexDef(myMutex01);  
myMutex01Handle = osMutexCreate(osMutex(myMutex01));
```

- 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 */  
}
```



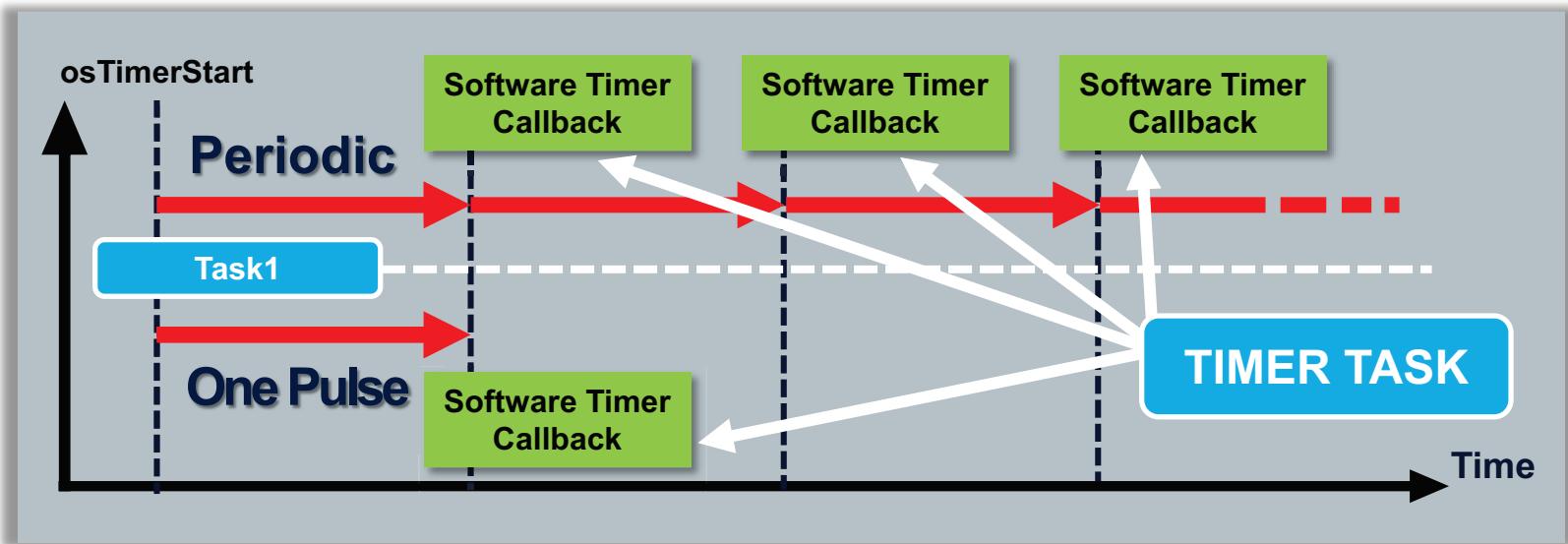


FreeRTOS Software Timers

Software Timers

108

- 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 and One Pulse)



- Software timer creation

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

- Software timer start

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

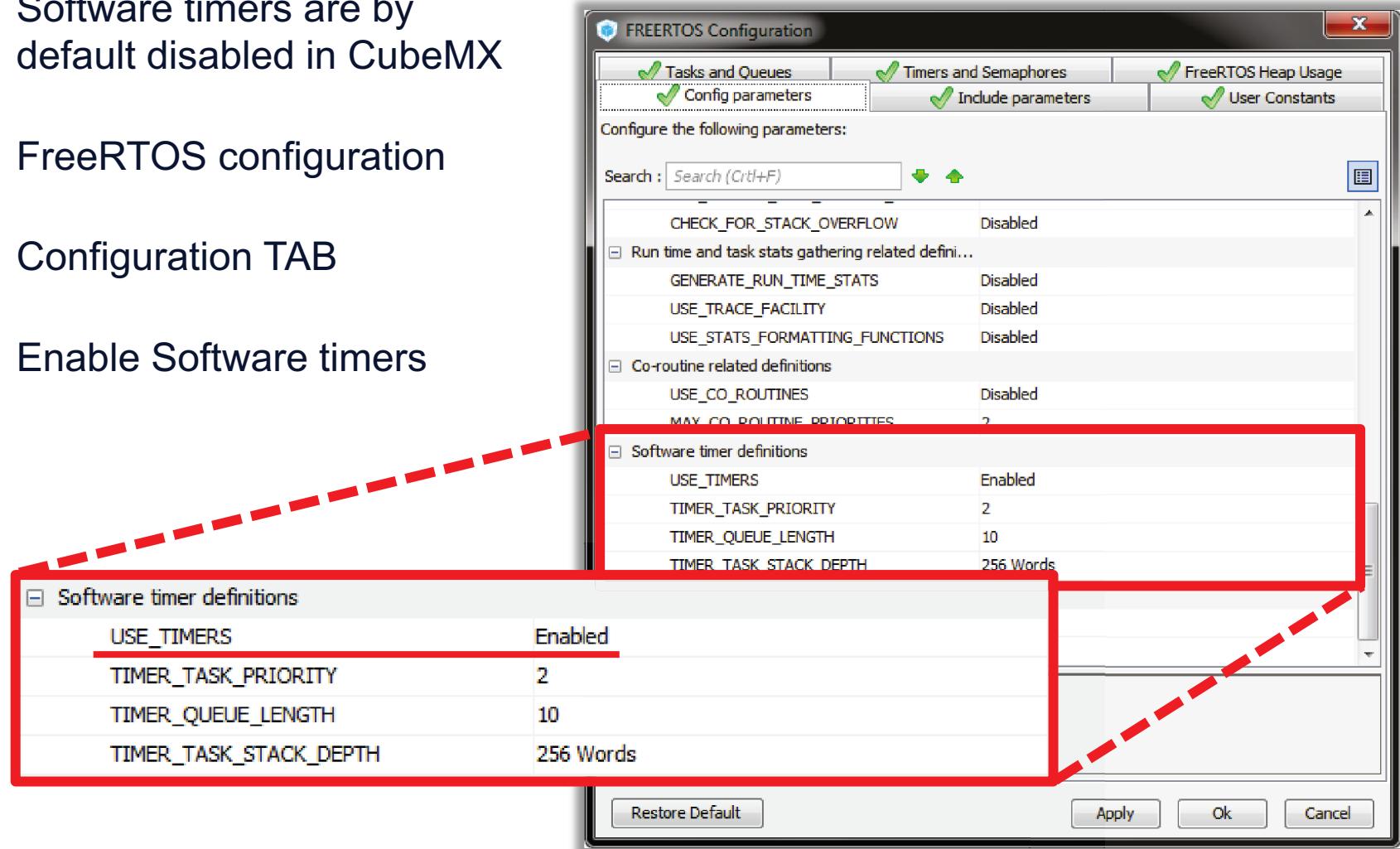
- Software timer stop

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

Software Timers lab

110

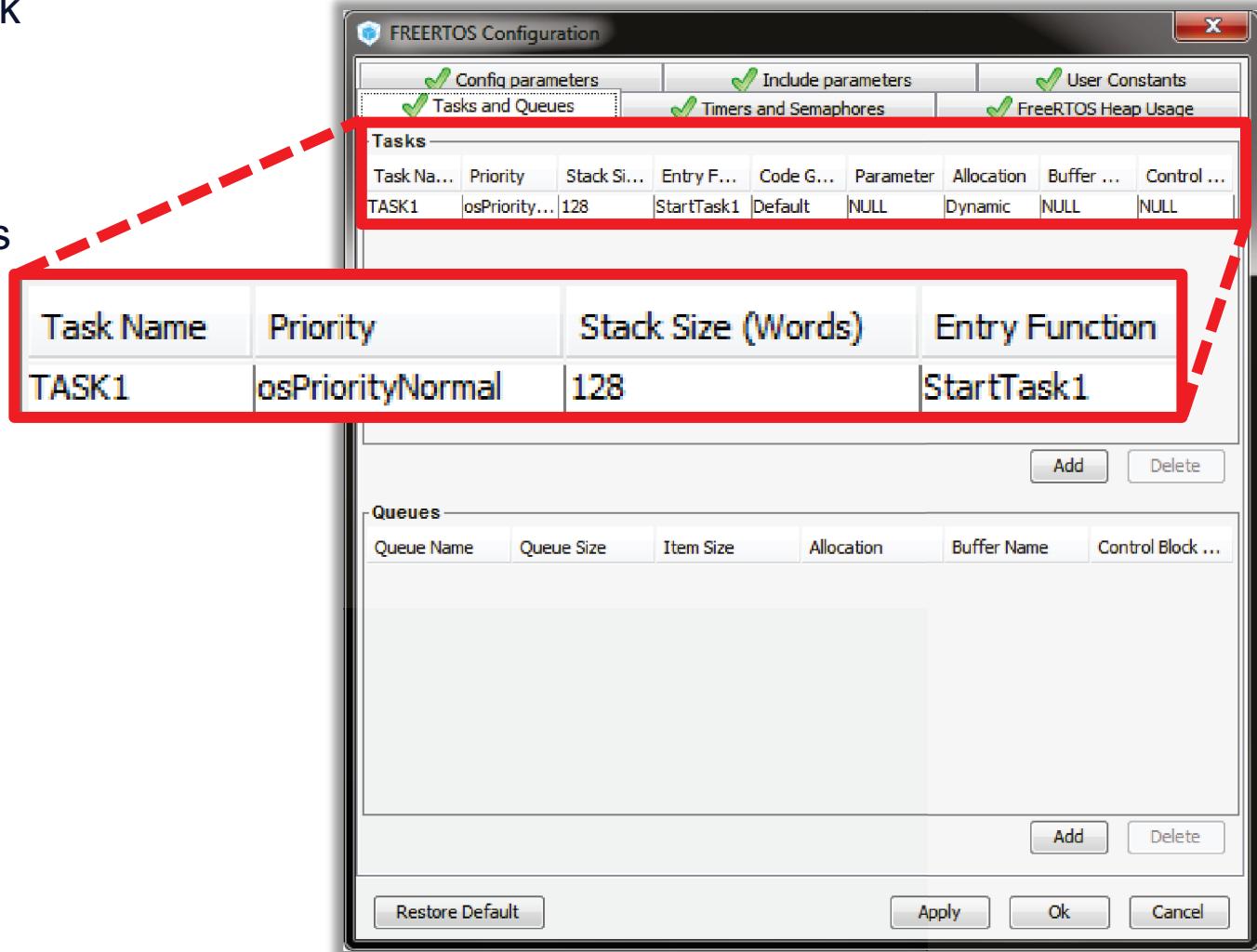
- Software timers are by default disabled in CubeMX
- FreeRTOS configuration
- Configuration TAB
- Enable Software timers



Software Timers lab

111

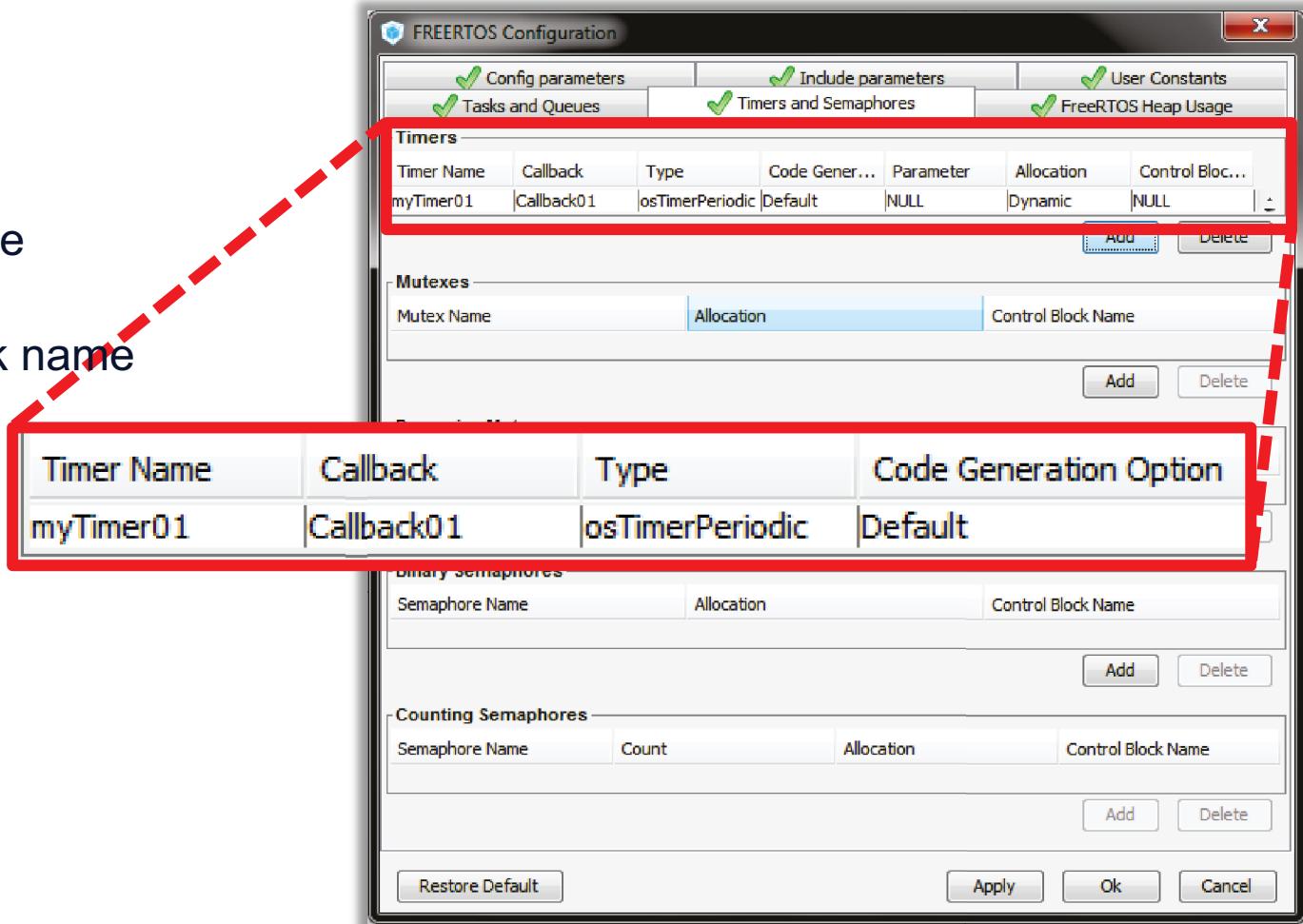
- Create one task
- Button Add
- Set parameters
- Set name
- Button OK



Software Timers lab

112

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



- 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 Timers lab

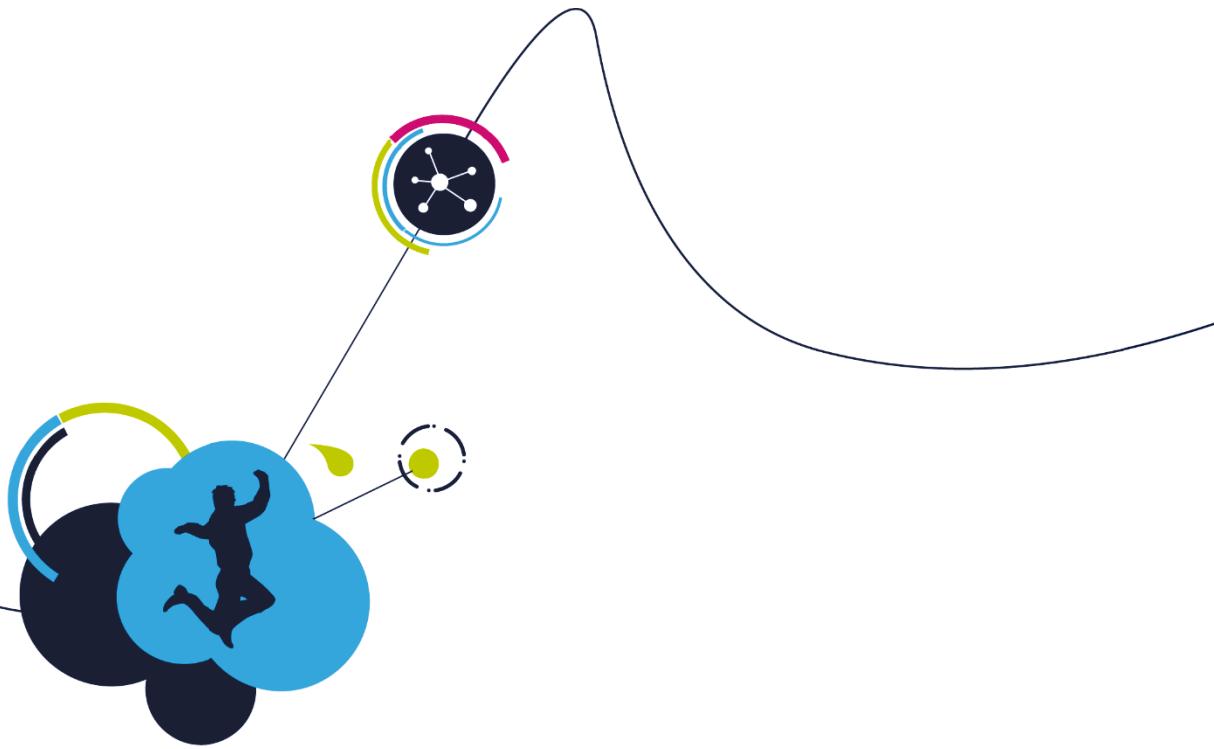
114

- Software timer start

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

- Software timer callback

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



End
Thanks for your attention