TRAINING MANUAL

EMBEDDED SYSTEMS DESIGN AND IOT APPLICATIONS USING CLOUD COMPUTING

Unit 10.1



Embedded System Programming

Time allocation: Week 10

Objectives

The aim of this module is to get immersed into embedded programming on a real hardware. To complete the basic workflow, simple applications are developed, implemented, and demonstrated in an Embedded System work environment. Experiment(s) in this module are conducted using Real-Time OS (RTOS) to demonstrate some of the most common practical applications.

Resources

- Desktop PC / Laptop
- Software development Tools
- Embedded Kit (ARM Cortex Series)
- Jumper Wires / Breadboard / LEDs, Switches

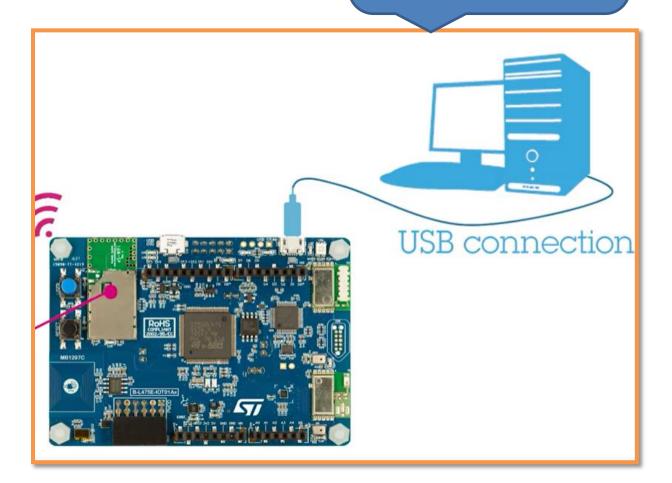
Topics to be covered:

- 1. Getting Started a Tutorial Project
- 2. ARM Cortex M4 I/O Programming
- 3. GPIO (General Purpose I/O) Programming and Interfacing
- 4. Reading Switches and Displaying the same on LEDs
- 5. Standard Application(s) Interfacing and Programming
- 6. Internet-of-Things (IOT) Application(s) Interfacing and Programming



Embedded System Setup

STM32 (ARM Cortex M4) Starter Kit - Development and Education Board



(STM32 µController)

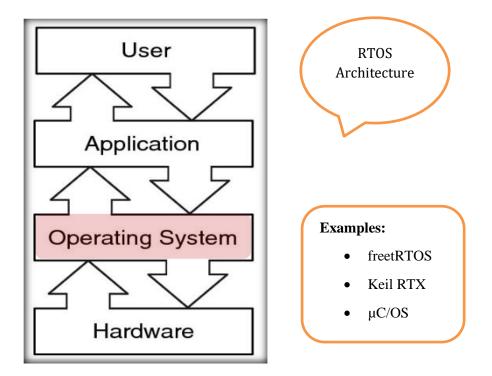
Document: Datasheet (stm3214s5) and Reference manual (stm3214s5)



Getting started with embedded RTOS (freeRTOS)

What is an RTOS and Multitasking?

A RTOS is a real-time operating system which manages **software and hardware resources** on a computing system and provides services to application software which are **not** possible with bare metal.



A RTOS is basically a software component that rapidly switches between tasks, giving the impression that multiple programs are being executed at the same time on a single processing core.

In actual fact the processing core can only execute one program at any one time, and what the RTOS is actually doing is rapidly switching between individual programming threads (or Tasks) to give the impression that multiple programs are executing simultaneously.

When switching between **Tasks** the RTOS has to choose the most appropriate task to load next. There are several scheduling algorithms available. However, to provide a responsive system most RTOS use a pre-emptive scheduling algorithm.

In a pre-emptive system each Task is given an individual priority value. The faster the required response, the higher the priority level assigned. When working in pre-emptive mode, the task chosen to execute is the highest priority task that is able to execute. This results in a highly responsive system.



While selecting a RTOS, one of the most important considerations is what type of response is desired – Is a hard real time response required? This means that there are precisely defined deadlines that, if not met, will cause the system to fail. Alternatively, would a non-deterministic, soft real time response be appropriate? In which case there are no guarantees as to when each task will complete.

The choice of RTOS can greatly affect the development of the design.

By selecting an appropriate RTOS the developer gains:

- A Task based design that enhances modularity, simplifies testing and encourages code reuse;
- An environment that makes it easier for engineering teams to develop together;
- Abstraction of timing behaviour from functional behaviour, which should result in smaller code size and more efficient use of available resources.

Peripheral support, memory usage and real-time capability are key features that govern the suitability of the RTOS. Using the wrong RTOS, particularly one that does not provide sufficient real time capability, will severely compromise the design and viability of the final product.

The RTOS needs to be of high quality and easy to use. Developing embedded projects is difficult and time consuming – the developer does not want to be struggling with RTOS related problems as well. The RTOS must be a trusted component that the developer can rely on, supported by in-depth training and good, responsive support.

What is FreeRTOS?

FreeRTOS is a class of RTOS that is designed to be small enough to run on a microcontroller (μ C). A microcontroller is a small and resource constrained processor that incorporates, on a single chip, the processor itself, read only memory (ROM / Flash) to hold the program to be executed, and the random access memory (RAM) needed by the programs it executes. Typically the program is executed from the read only memory. One of the main attractions in freeRTOS is its free of cost licensing model.

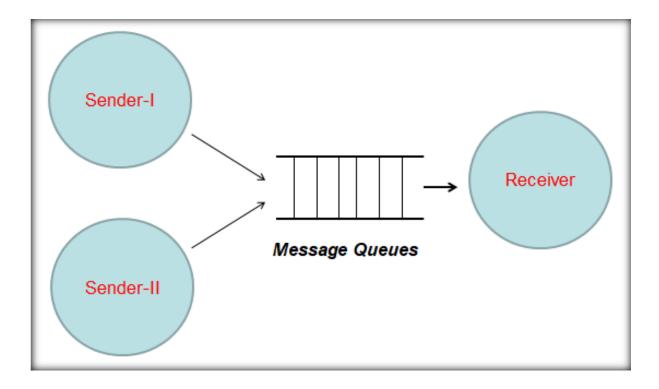
Microcontrollers are a central piece of the embedded systems that normally have a very specific job to do. The size constraints, and dedicated end application nature, rarely warrant the use of a full package implementation.

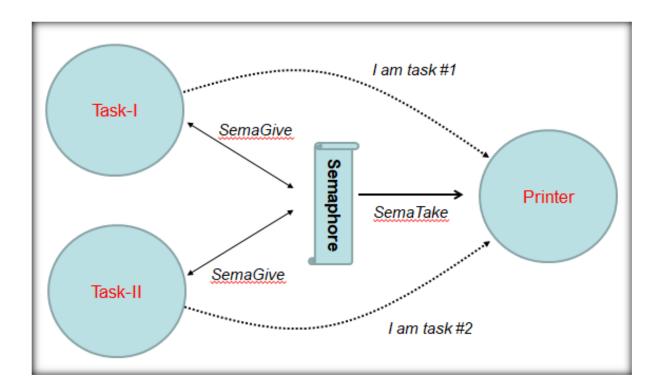
Applications - few to mention:

Command and control systems, heart pacemaker, industrial automation, and modern robotics systems



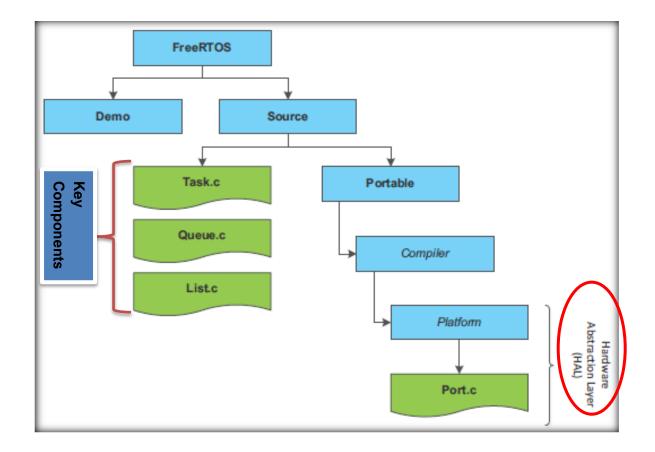
Key Features - Tasks Synchronization through Semaphores / Queues







FreeRTOS architecture

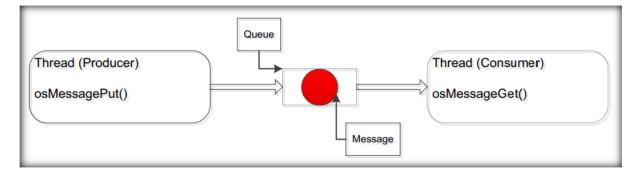


TRAINING MANUAL

Unit 10: Getting started with SPI Bus



Queue process

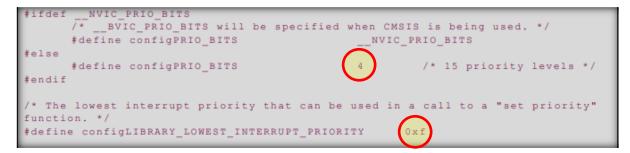


Semaphore from ISR

	Thread (blocked) osSemaphoreWait()
Interrupt	Thread (running)
osSemaphoreRelease()	osSemaphoreWait()

FreeRTOS configuration

FreeRTOSConfig.h



Reference:

<u>https://www.st.com/resource/en/user_manual/dm00105262-developing-applications-on-stm32cube-with-rtos-stmicroelectronics.pdf</u>



Free RTOS APIs

APIs Categories	API
Task Creation	– xTaskCreate 🥚 – vTaskDelete
Task Control	 vTaskDelay vTaskDelayUntil uxTaskPriorityGet vTaskPrioritySet vTaskSuspend vTaskResume xTaskResumeFromISR vTaskSetApplicationTag xTaskCallApplicationTaskHook
Task Utilities	 xTaskGetCurrentTaskHandle xTaskGetSchedulerState uxTaskGetNumberOfTasks vTaskList vTaskStartTrace ulTaskEndTrace vTaskGetRunTimeStats
Kernel Control	 vTaskStartScheduler vTaskEndScheduler vTaskSuspendAll xTaskResumeAll
Queue Management	 xQueueCreate xQueueSend xQueueReceive xQueuePeek xQueueSendFromISR xQueueSendToBackFromISR xQueueSendToFrontFromISR xQueueReceiveFromISR vQueueReceiveFromISR vQueueAddToRegistry vQueueUnregisterQueue
Semaphores	 vSemaphoreCreateBinary vSemaphoreCreateCounting xSemaphoreCreateMutex xSemaphoreTake xSemaphoreGiveFromISR

Task-0



This task demonstrates:

• Simple working of a freeRTOS on STM32L4S5 device

Objective

• Learn bare metal set up dealing with the hardware

Learn how to set up the RTOS with DMA and Interrupt in STM32CubeMX.

- Generate code in STM32CubeMX and using HAL functions.
- Create applications to start the **freeRTOS** and learn how to set **freeRTOS** in different modes.

On the target board,

You will use GPIOs (LEDs) and/or USART (Tera-Term) to demonstrate the working of RTOS;

• Using the IOT board, based on STM32L4S5 µController from ST-Microelectronics Ltd.



Launch Keil µVision Development Tools



🔀 μVision								
File Edit View	Project	Flash	Debug	Peripherals	Tools	SVCS	Window	Help
🗋 💕 🗔 🥥	Nev	v µVisio	n Project					
	Nev	v Multi-	Project W	orkspace				
Project	Clos Exp Mar Sele	nage	ct se for Targ	et				

- ✓ In the New μ Vision Project window, browse to the folder "**bbb**" you just created
- ✓ Enter a name for the project file. We will call it "hello" and click Save

😨 Create New Project	
\leftarrow \rightarrow \checkmark \Uparrow \blacksquare \Rightarrow This PC \Rightarrow Desktop \Rightarrow bbb	✓ ♂ Search bbb
Organise 🔻 New folder	8=≡ ▼
Downloads ^ Name	Date modified Type
Music No	items match your search.
E Pictures	icens materi you searem
Videos	
L Windows (C:) V <	
File name: hello	
Save as type: Project Files (*.uvproj; *.uvprojx)	
∧ Hide Folders	Save Cancel



Select Device for Target

✓ STMicroelectronics STM32L4S5VITx

Select Devic	ce for Target 'Target 1'	
Device		
	Software Packs	•
Vendor:	<unknown></unknown>	
Device:	<unknown></unknown>	
Toolset:	<unknown></unknown>	
Search:		
		Description:
	RM	
	TMicroelectronics	
	\$ STM32F4 Series	
.	\$ STM32F7 Series	

Select Device for Target 'Target 1'	×
Device	
Software Packs	•
Vendor: STMicroelectronics	
Device: STM32L4S5VITx	
Toolset: ARM	
Search:	
,	Description:
STM32L4R5 STM32L4R7 STM32L4R9 STM32L4S5 STM32L4S5 STM32L4S5Qlix STM32L4S5Qlix STM32L4S5ZlTx STM32L4S5ZlYx	ST has built a new architecture to reach best-in-class ultra-low-power figures thanks to its high flexibility. STM32L4 MCUs have scored 176.7 in the standardized EEMBC ULPBench tests that compare the efficiency of ultra-low-power microcontrollers. Moreover, the STM32L4 series shatters performance limits in the ultra-low-power world. It delivers 100 DMIPS based on its ARM Cortex-M4 core with FPU and ST ART Accelerator at 80 MHz. STM32L4 microcontrollers offer dynamic voltage scaling to balance power consumption with processing demand, low-power peripherals (LP UART, LP timers) available in Stop mode, safety and security features, smart and numerous peripherals, advanced and low-power analog peripherals such as op



Alternatively: Try,

Project -> Options for Target 'Target 1'-> Device,

Scroll down, and select "STM32L4S5VITx"

Configure default Initialization files

Software Component	Sel.	Variant	Version	Description
🗉 🚸 Board Support		STM32L496G-Discovery	1.1.0	STMicroelectronics STM32L496G-Discovery Board
CMSIS				Cortex Microcontroller Software Interface Components
CORE			5.6.0	CMSIS-CORE for Cortex-M, SC000, SC300, Star-MC1, ARMv8-
DSP		Source	1.14.3	CMSIS-DSP Library for Cortex-M and Cortex-A
NN Lib			4.0.0	CMSIS Neural Network(NN) Library
🗈 💠 DSP				
🗉 🚸 RTOS (API)			1.0.0	CMSIS-RTOS API for Cortex-M, SC000, and SC300
🗉 🚸 RTOS2 (API)			2.1.3	CMSIS-RTOS API for Cortex-M, SC000, and SC300
CMSIS Driver				Unified Device Drivers compliant to CMSIS-Driver Specification
🛛 💠 Compiler		ARM Compiler	1.7.2	Compiler Extensions for ARM Compiler 5 and ARM Compiler
🗐 🚸 Device				Startup, System Setup
Startup			1.3.1	System Startup for STMicroelectronics STM32L4 Series
🗉 💠 STM32Cube Framework (AP			1.1.0	STM32Cube Framework
🗉 💠 STM32Cube HAL				STM32L4xx Hardware Abstraction Layer (HAL) Drivers
/alidation Output		Description		
Resolve Select Packs Details			ок	Cancel
			\smile	

Project -> Manage -> Manage Run-Time Environment,



Project Outlook

File Edit View Project Flash Debug Periphera	ils Tool	s SVC	5 Win	dow	Help
🚔 🖼 🥔 👗 🛍 🛍 🧐 (~ + +	$\mathbb{I}^{n-1/2}$	8.8			//= // _%
🔌 🔛 🕮 🧼 🕶 🧮 🙀 Target 1	~ 🔊	📥 🤊	a 🔶	<	<u> </u>
Project 📮 🗴					
 Project: hello Target 1 Source Group 1 CMSIS Device startup_stm32l4s5vx.s (Startup) system_stm32l4xx.t (Startup) cmsis_armclang.n cmsis_version.h core_cm4.h mpu_armv7.h stdint.h stm32l4s5xx.h stm32l4xx.h system_stm32l4xx.h 					



Compiler & Debugger Setting (Project → Options for Target)

Options for Target 'Target 1'		×
Device Target Output Listing User C/C++ (AC6) As	sm Linker Debug	Utilities
Texas Instruments TM4C123GH6PM	Code Generation ARM Compiler:	Use default compiler version 6 <
Xtal (MHz): <a> 		· · · · · · · · · · · · · · · · · · ·

🔣 Options for Target 'Target 1'	×
Device Target Output Listing User C/C++ (AC6) Asm Linker	Debug Utilities
Preprocessor Symbols	· · · ·
Define:	
Undefine:	
Language / Code Generation Execute-only Code Warnings: MISRA Compatible	✓ Language C: c11
Optimization: -00 Tum Warnings into Errors	Language C++: c++11
Link-Time Optimization	Short enums/wchar

🕼 Options for Target 'Target 1'	×
Device Target Output Listing User C/C++ (AC6) A	sm Linker Debug Utilities
O Use Simulator with restrictions Settings	O Use: ST-Link Debugger
Limit Speed to Real-Time	☆
✓ Load Application at Startup ✓ Run to main()	I Load Application at Startup I Run to main()

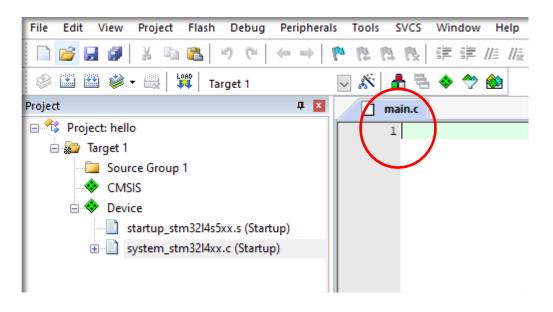
V	Options for Target 'Target 1'	×
C	Device Target Output Listing User C/C++ (AC6) Asm Linker Debug Utilities	
	Configure Flash Menu Command	
	Use Target Driver for Flash Programming Use Target Driver	
	Use Debug Driver Settings 🔽 Update Target before Debugging	

<0K>

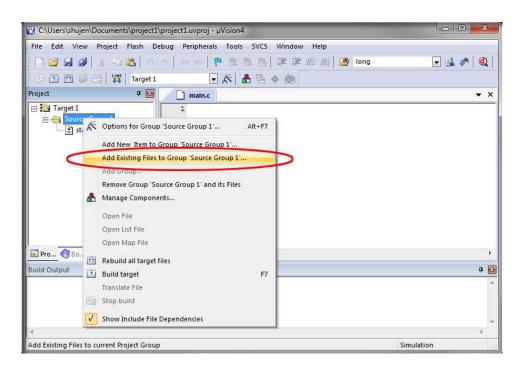


Add a Source File to the Project

- ✓ Click the **File->New** button to add a new text file to the display with the default name **Text1**.
- ✓ From the menu, select File > Save As... to open the Save As dialog box. Browse to the project folder if it is not already there. Type in the file name "main.c" and click Save.



✓ The new file needs be added to the project. Right click on the folder Source Group 1 in the Project window and select Add Existing Files to Group 'Source Group 1'...



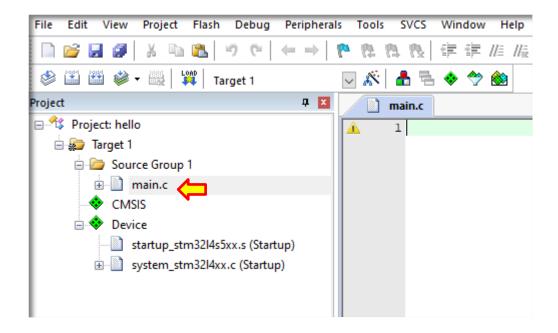
 \checkmark In the dialog box, browse to the project folder if it is not already there.



Click select main.c then click Add.

C:\Users\shujen\Documents\pro	ect1\project1.uvproj - µVision4		
ile Edit View Project Flash	Debug Peripherals Tools SVCS Wi	ndow Help	
D 🗳 🖬 🕼 🔺 🕹 🛍 🕯	のでしゃや「やなれた」達	: 🎼 //:: //:: 🖄 long	💌 🗟 🥐 🛛 🍭
🕸 🏝 🕮 🧼 🔙 🙀 Tar	get 1 💽 🌾 🛔 🗟 🚸 (
oject 4	🖸 🗋 main.c		~ 3
Target 1	1		
	Add Files to Group 'Source Group 1'	 ▼ ← È 🗳 💷 -	1
	Name main.c	Date modified T ₃ 11/16/2013 9:52 AM C	
	main.c	11/10/2015 9:52 AM C	
Pro 🚷 Bo {} Fu 0.			
ild Output	٠ m		Ф.
	A Loss		
	File name:	Add	
	File <u>n</u> ame: main.c	Add	
	File name: main.c Files of type: C Source file (*.c)	Add Close	

✓ The file should appear in the project window under Source Group 1 folder.
 Click Close to close the dialog box.





Create RTOS based Project Configuration

🔀 C:\Users\Liaqat\Desktop\bbb\hello.uvprojx - µVision													
File	Edit View	Proj	ect Fl	ash D	Debug	Peripherals	Tools	SVCS	Window	Help			
h	ii 🛛 🖉		New µVision Project									(A)	🍭 • 🌢 🔿 🔗 🕀 • 🖻 •
٢	🕮 🞬 🧼		New Multi-Project Workspace										
Project		Open Project Close Project									h		
.	Project: hell										8		
Ē	ᇶ Target 1	Export											
	📜 Soui	r	Manag	e 🔶								8	Project Items
			Select Device for Target						5	Multi-Project Workspace			
			Remov	e Item							🛁	•	Run-Time Environment
			Optior	is for Ta	arget 'Ta	rget 1'					Alt+F7	*	Select Software Packs
					- s	Reload Software Packs							
		2000	Clean 1	-							_		Pack Installer
			-							Migrate to Version 5 Format			
			Rebuil	d all tar	get files	;							

There are two (2) versions of **freeRTOS** implementation.

- **Standard** (Traditional original implementation)
- **CMSIS-RTOS2** (An ARM version which is platform independent)

NOTE:		
Make sure there i	s <u>one and only one</u> c	onfiguration produced in any working folder.
If necessary, mak	e the working folder	an empty folder by deleting everything from there.

Standard freeRTOS Configuration



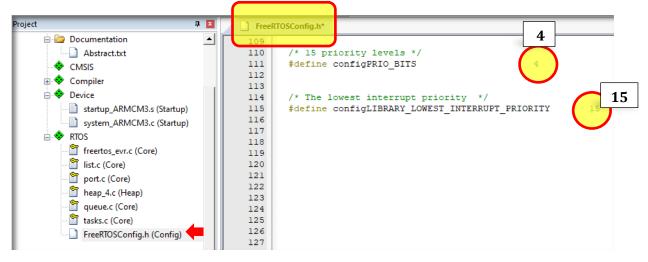
<u>1 of 2 options</u>

Software Component	Sel.	Variant		Version
🖯 🚸 CMSIS				
CORE				5.6.0
DSP		Source		1.14.3
NN Lib				4.0.0
🕀 💠 DSP				
🕀 💠 RTOS (API)				1.0.0
🗄 🚸 RTOS2 (API)				2.1.3
🗄 💠 CMSIS Driver				
🗄 🚸 Compiler		ARM Compiler		1.7.2
🖯 🔶 Device				
Startup				1.0.1
🗄 🚸 File System		MDK-Plus	~	6.15.3
🗄 💠 Graphics		MDK-Plus	~	6.30.1
🗄 🚸 IoT Client				
🗄 💠 loT Service				
🗉 💠 IoT Utility				
🗉 💠 Network		MDK-Plus	\sim	7.18.0
🖶 🚸 RTOS		FreeRTOS		10.5.1
Config	V	FreeRTOS	~	10.5.1
Core	v	Cortex-M	~	10.5.1
Coroutines	V			10.5.1
Event Groups	v			10.5.1
Heap	v	Heap_4	~	10.5.1
Message Buffer	v			10.5.1
Stream Buffer	V			10.5.1
Timers				10.5.1
🗄 🚸 USB		MDK-Plus	\sim	6.16.1
		,		
Resolve Select Packs De	etails			ОК

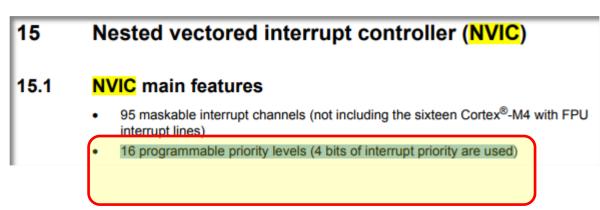


Correction!!

Update the priority levels

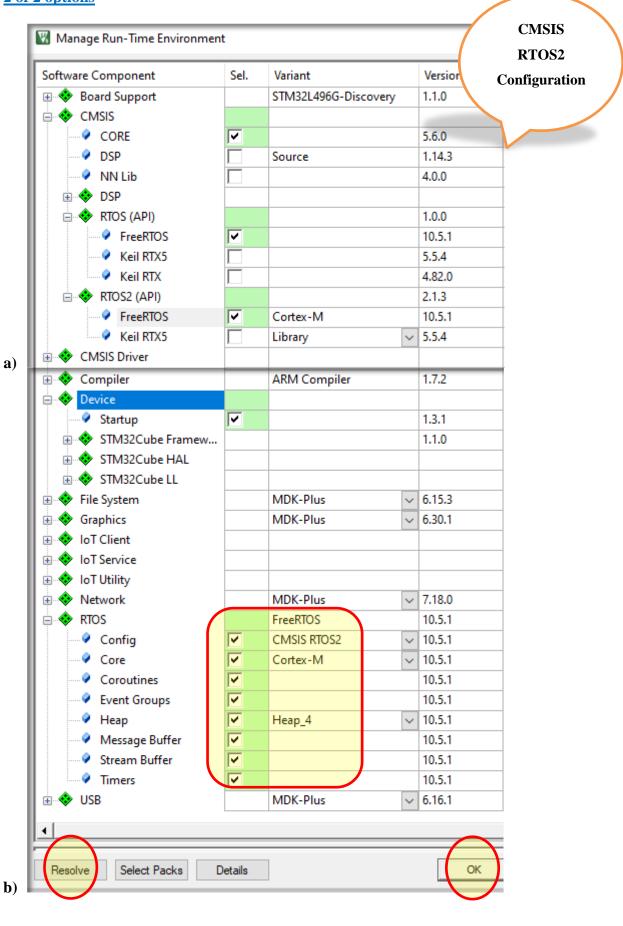


Cross checked (Reference Manual - STM32L4S5)





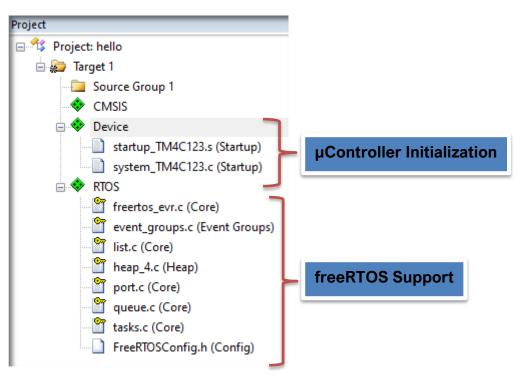
2 of 2 options



TRAINING MANUAL Unit 10: Getting started with SPI Bus



c)



d)

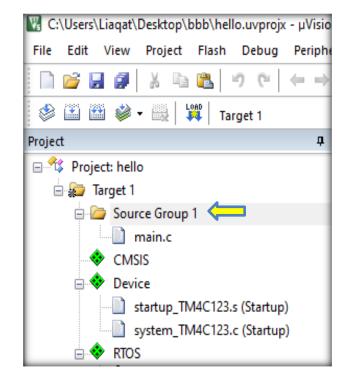
Ignore this step, if "*main.c*" file is already in the project otherwise complete the below listed steps.

✓ Open a new blank file
 File → New

File → Save As:

"main.c"

✓ Add this file into the project

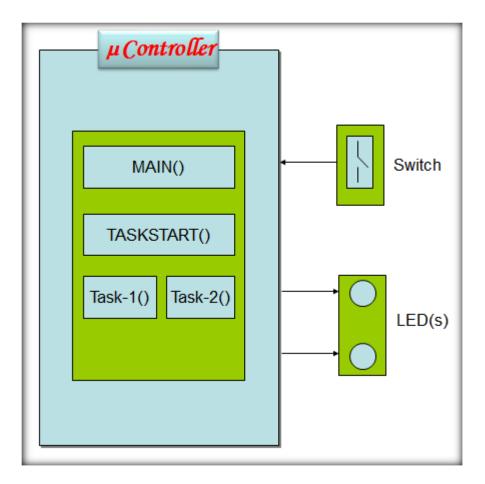




Task-1

This task demonstrates how to:

- Configure **GPIO** ports
- Create multiple Tasks in RTOS (FreeRTOS)
- Toggles a set of LEDs (PA5 & PB14) of PORTA & PORTB through Tasks- 1 & 2



- C Programming Code for this task is given next
- Copy & Paste the same in "main.c" file
- Build, Download, and Run on the Embedded Kit / Board
- Monitor the LEDs toggling patterns (sequences)



Copy and paste the sample code into the "main.c" editor window.

```
// Example-1: RTOS based multitasking
//
#include <stdio.h>
#include "RTE_Components.h"
                              // Component selection
#include CMSIS_device_header
#include "FreeRTOS.h"
                          // Keil::RTOS:FreeRTOS:Core
#include "task.h"
                       // Keil::RTOS:FreeRTOS:Core
#include "stm32l4s5xx.h"
/*-----*/
void vTask1( void *pvParameters ) {
       /* As per most tasks, this task is implemented in an infinite loop. */
       for(;;) {
                      GPIOA->ODR ^= (0x1 << 5); //PA5 ON
                      vTaskDelay(100); // freeRTOS function
       }
}
/*-----*/
void vTask2( void *pvParameters ) {
       /* As per most tasks, this task is implemented in an infinite loop. */
       for(;;) {
                      GPIOB->ODR ^= (0x1 << 14); //PB14 ON
                      vTaskDelay(500); // freeRTOS function
       }
}
// Initialize FreeRTOS and start the initial set of tasks.
int main(void){
/* Create required number of task(s) */
       xTaskCreate( vTask1,
                              /* Pointer to the function that implements the task. */
                                      "Task 1", /* Text name for the task. */
                                                     /* Stack depth in words. */
                                      200,
                                      NULL,
                                                     /* We are not using the task parameter. */
                                                    /* This task will run at priority 1. */
                                      1,
                                      NULL);
                                                    /* We are not using the task handle. */
       xTaskCreate( vTask2, "Task 1", 200, NULL, 1, NULL );
```



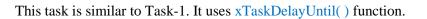
```
/* GPIO Port configuration */
 // Enable the clock to GPIO Port B
 RCC->AHB2ENR |= 0x2; //RCC_AHB2ENR_GPIOB_EN;
         // MODE: 00: Input mode, 01: General purpose output mode
      10: Alternate function mode, 11: Analog mode (reset state)
 //
 GPIOB->MODER &= (unsigned int)~(0x3 << 14*2); // Clear bit PB14
 GPIOB->MODER |= (0x1 << 14*2); // Set bit PB14 Output
 // Enable the clock to GPIO Port A
 RCC->AHB2ENR |= 0x1; //RCC_AHB2ENR_GPIOA_EN;
         // MODE: 00: Input mode, 01: General purpose output mode
      10: Alternate function mode, 11: Analog mode (reset state)
 //
 GPIOA->MODER &= (unsigned int)~(0x3 << 5*2); // Clear bit PA5
 GPIOA->MODER |= (0x1 << 5*2); // Set bit PA5 Output
/* Start the scheduler so our tasks start executing. */
         vTaskStartScheduler();
/* If all is well we will never reach here as the scheduler will now be
         running. If we do reach here then it is likely that there was insufficient
         heap available for the idle task to be created. */
         while(1) {
         }
} // Program ends here
```

Exercise:

✓ Change priorities of;

- o *task1 to 1*,
- o *task2 to 2*
- What is the impact and difference?
- ✓ Change priorities of;
 - o *task1 to 2,*
 - o *task2 to* **1**
 - What is the impact and difference?

Task-2



The given activities are synchronized through "Semaphore" functions.

```
// Example-1: RTOS based multitasking
// This task uses "Semaphore" function
#include <stdio.h>
#include "RTE_Components.h"
                                 // Component selection
#include CMSIS_device_header
#include "FreeRTOS.h"
                             // Keil::RTOS:FreeRTOS:Core
#include "task.h"
                          // Keil::RTOS:FreeRTOS:Core
#include <semphr.h> // Keil::RTOS:FreeRTOS:Core
#include "stm32l4s5xx.h"
// declare semaphore
SemaphoreHandle_t xSemaphore;
/*-----*/
// This thread generates semaphore
void vTask1( void *pvParameters ) {
        portTickType xLastWakeTime;
xLastWakeTime = xTaskGetTickCount(); // Initialized with the current tick
        // Infinite loop.
        for(;;) {
                xSemaphoreGive( xSemaphore ); // Post Sema
                GPIOA->ODR ^= (0x1 << 5); //PA5 ON
                vTaskDelayUntil( &xLastWakeTime, ( 500 ) ); // Absolute delay of 500 ticks
        }
}
```





/*-----*/ // This thread picks up semaphore void vTask2(void *pvParameters) { // Infinite loop. for(;;) { xSemaphoreTake(xSemaphore, portMAX_DELAY); // Pend Sema (0xffff, max. ticks) GPIOB->ODR ^= (0x1 << 14); //PB14 ON } } // Initialize FreeRTOS and start the initial set of tasks. int main(void){ /* Create required number of task(s) */ xTaskCreate(vTask1, /* Pointer to the function that implements the task. */ "Task 1", /* Text name for the task. */ 200. /* Stack depth in words. */ NULL, /* We are not using the task parameter. */ 1, /* This task will run at priority 1. */ /* We are not using the task handle. */ NULL); xTaskCreate(vTask2, "Task 2", 200, NULL, 2, NULL); /* GPIO Port configuration */ // Enable the clock to GPIO Port B RCC->AHB2ENR |= 0x2; //RCC_AHB2ENR_GPIOB_EN; // MODE: 00: Input mode, 01: General purpose output mode 10: Alternate function mode, 11: Analog mode (reset state) // GPIOB->MODER &= (unsigned int)~(0x3 << 14*2); // Clear bit PB14 GPIOB->MODER |= (0x1 << 14*2); // Set bit PB14 Output



```
// Enable the clock to GPIO Port A
 RCC->AHB2ENR |= 0x1; //RCC_AHB2ENR_GPIOA_EN;
// MODE: 00: Input mode, 01: General purpose output mode
 // 10: Alternate function mode, 11: Analog mode (reset state)
 GPIOA->MODER &= (unsigned int)~(0x3 << 5*2); // Clear bit PA5
 GPIOA->MODER |= (0x1 << 5*2); // Set bit PA5 Output
/* Attempt to create a semaphore. */
         xSemaphore = xSemaphoreCreateBinary();
         if( xSemaphore != NULL ) {
                  /* The semaphore was created successfully. */
                  /* Start the scheduler so our tasks start executing. */
                  vTaskStartScheduler();
         }
         else {
                  // Sema creation failed, do something else
         }
/* If all is well we will never reach here as the scheduler will now be
         running. If we do reach here then it is likely that there was insufficient
         heap available for the idle task to be created. */
         while(1) {
         }
} // Program ends here
```



Task-3

This task is similar to Task-1.

The given activities are synchronized through "Queue" functions.

Monitor Tera-Term Window to see the messages movement from sender to receiver blocks.

```
// Example-1: RTOS based multitasking
// This lab uses "QUEUE" to synchronize the tasks.
// Monitor Tera-Term Window for the data movement
//
// Task 1 (PRODUCER) - Generates messages
// Task 2 (CONSUMER) - Displays data on "Tera-Term" window
//
#include <stdio.h>
#include "RTE_Components.h"
                               // Component selection
#include CMSIS_device_header
#include "FreeRTOS.h"
                            // Keil::RTOS:FreeRTOS:Core
#include "task.h"
                        // Keil::RTOS:FreeRTOS:Core
#include <queue.h>
#include "stm32l4s5xx.h"
// Define the data type that will be queued
typedef struct {
              // object data type
uint8_t msgID;
uint8_t msgData[5];
} MSGQUEUE_OBJ_t;
// Define the queue parameters
#define QUEUE_LENGTH 16
#define QUEUE_ITEM_SIZE sizeof( MSGQUEUE_OBJ_t )
static void vSenderTask( void *pvParameters );
static void vReceiverTask( void *pvParameters );
// Initialize FreeRTOS and start the initial set of tasks.
int main(void) {
        // Enable the clock to GPIO Port A
        RCC->AHB2ENR |= 1;
                                     /* enable GPIOA clock */
        // MODE: 00: Input mode, 01: General purpose output mode
        //
            10: Alternate function mode, 11: Analog mode (reset state)
        GPIOA->MODER &= (unsigned int)~(0x3 << 5*2); // Clear bit PA5
        GPIOA->MODER |= (0x1 << 5*2); // Set bit PA5 Output
```

ALL CLARKER CONTRACTOR

```
// Enable the clock to GPIO Port B
        RCC->AHB2ENR |= 0x2; //RCC_AHB2ENR_GPIOB_EN;
        // MODE: 00: Input mode, 01: General purpose output mode
        // 10: Alternate function mode, 11: Analog mode (reset state)
        GPIOB->MODER &= (unsigned int)~(0x3 << 14*2); // Clear bit PB14
        GPIOB->MODER |= (0x1 << 14*2); // Set bit PB14 Output
// -----
// Configure USART1
                // Enable the clock to GPIO Port B
                RCC->AHB2ENR |= 0x2; //RCC_AHB2ENR_GPIOB_EN;
                GPIOB->AFR[0] &= ~0x0F000000;
                GPIOB->AFR[0] |= 0x07000000; /* PB6 for USART1 TX */
                GPIOB->AFR[0] &= ~0xF0000000;
                GPIOB->AFR[0] |= 0x70000000; /* PB7 for USART1 RX */
                // MODE: 00: Input mode, 01: General purpose output mode
                    10: Alternate function mode, 11: Analog mode (reset state)
                11
                GPIOB->MODER &= (unsigned int)~(0x3 << 6*2); // Clear bit PB6
                GPIOB->MODER |= (0x2 << 6*2); // Set bit PB6 Alternate
                GPIOB->MODER &= (unsigned int)~(0x3 << 7*2); // Clear bit PB7
                GPIOB->MODER |= (0x2 << 7*2); // Set bit PB7 Alternate
 RCC->APB2ENR |= 0x4000;
                                /* enable USART1 clock */
 USART1->CR1 = 0x000C;
                          /* enable Tx, Rx, 8-bit data */
 USART1->CR2 = 0x0000; /* 1 stop bit */
 USART1->CR3 = 0x0000; /* no flow control */
 USART1->BRR = 0x0023;
                                 /* 115200 baud @ 16 MHz */
 USART1->CR1 |= 0x0001; /* enable USART1 */
// -----
// Create the queue, storing the returned handle in the xQueue variable.
        OueueHandle t xOueue:
        xQueue = xQueueCreate( QUEUE_LENGTH, QUEUE_ITEM_SIZE );
        if( xQueue != NULL ) {
                // Create two instances of the PRODUCER task
                // Tasks pass on the queue handle as the task parameter
                // Both instances of tasks are created at priority 1
                xTaskCreate( vSenderTask,
                                 "Sender1",
                                 100.
                                 (void *) xQueue, // The queue handle is used as the task parameter.
                                 1,
                                 NULL
                        );
```



```
xTaskCreate( vSenderTask, "Sender2", 100, ( void * ) xQueue, 1, NULL );
                 // Create a CONSUMER task at priority 2
                 xTaskCreate( vReceiverTask, "Receiver", 100, ( void * ) xQueue, 2, NULL );
                  // Start the task executing
                 vTaskStartScheduler();
        }
// Execution will only reach here if there was not enough FreeRTOS heap memory
// remaining for the idle task to be created
         while(1) {
        }
} // main() ends here
/*-----*/
// Generate data stream
static void vSenderTask( void *pvParameters ) {
         portBASE_TYPE xStatus;
         MSGQUEUE_OBJ_t xMessage [2] = {
                           {'A', 11,2,3,4,5},
                           {'B', 16,7,8,9,0xa}
                 };
         int i=0;
         // The queue handle is passed into this task as the task parameter.
         // Cast the parameter back to a queue handle.
         QueueHandle_t xQueue;
         xQueue = ( QueueHandle_t ) pvParameters;
         for(;;) {
                  GPIOA->ODR ^= (0x1 << 5); //PA5 ON
                 vTaskDelay(100);
         // Send the message to the queue, waiting for 10 ticks for space to become
                 // available if the queue is already full.
                 i ^= 0x1; // update #msg index
                 do {
                           xStatus = xQueueSendToBack( xQueue, &xMessage[ i ], 10 );
                 } while ( xStatus != pdPASS );
// Allow the other sender task to execute.
                  taskYIELD();
        }
} // Thread ends here
```

TRAINING MANUAL

Unit 10: Getting started with SPI Bus



```
/*-----*/
// Pick up received messages
static void vReceiverTask( void *pvParameters ) {
        portBASE_TYPE xStatus;
        MSGQUEUE_OBJ_t xMessage;
        int i, mm;
        uint8_t data;
// The queue handle is passed into this task as the task parameter
        QueueHandle_t xQueue;
        xQueue = ( QueueHandle_t ) pvParameters;
        for(;;) {
                 GPIOB->ODR ^= (0x1 << 14); //PB14 ON
                 vTaskDelay(100);
                 // Wait for the maximum period for data to become available on the queue.
                 xStatus = xQueueReceive( xQueue, &xMessage, 1000 ); // xTicksToWait = 1000
                 if( xStatus == pdPASS ) {
                                  // xMessage now contains the received data.
                                  mm = sizeof(xMessage.msgData) / sizeof(uint8_t);
                                  for (i=0; i< mm; i++) {
                                          data = xMessage.msgData[i];
                                          printf("%d ", data & 0xff); // // write to monitor
                                  }
                 }
        }
} // Thread ends here
// -----
// The code below is the interface to the C standard I/O library.
// All the I/O are directed to the console, which is UART1.
FILE __stdout = {1};
/* Called by C library console/file output */
                 int fputc(int ch, FILE *f) {
                 while (!(USART1->ISR & 0x0080)) {} // Wait until Tx buffer empty
                 USART1->TDR = ch;
                 return ch;
}
```



Task-3/b

This task is similar to Task-1.

This task uses "freeRTOS_V2" variant

The given activities are synchronized through "Queue" functions.

Monitor Tera-Term Window to see the messages movement from sender to receiver blocks.

Procedure

- Create an empty folder
- Open a New Project
- Walk-through the necessary steps
- Create "CMSIS freeRTOS-V2" based Project Configuration
- Edit "main.c"
- Insert the sample code (listed below)
- Build / Flash download code / Run, and Monitor LEDs on the h/w board
- Monitor "Tera-Term" Window, the below shown messages should be there

💻 COM3 - Tera T	ferm VT	-	0 X
File Edit Setup	Control Window Help		
A 83 85 A 83 85 A 83 4 A 83 85 A 83 85 A 83 85 A 83 4	85 A 83 85 A 85 A 83 85 A 83 85 A 83 85 A 83 85 A 85 A 83 85 A 83 85 A 83 85 A 83 85 A 85 A 83 85 A 83 85 A 83 85 A 83 85 A 85 A 83 85 A 83 85 A 83 85 A 83 85 A 85 A 83 85 A 83 85 A 83 85 A 83 85 A 85 A 83 85 A 83 85 A 83 85 A 83 85 A 85 A 83 85 A 83 85 A 83 85 A 83 85 A 85 A 83 85 A 83 85 A 83 85 A 83 85 A 85 A 83 85 A 83 85 A 83 85 A 83 85 A 85 A 83 85 A 83 85 A 83 85 A 83 85 A 85 A 83 85 A 83 85 A 83 85 A 83 85 A 85 A 83 85 A 83 85 A 83 85 A 83 85 A 85 A 83 85 A 83 85 A 83 85 A 83 85 A 85 A 83 85 A 83 85 A 83 85 A 83 85 A 85 A 83 85 A 83 85 A 83 85 A 83 85 A 85 A 83 85 A 83 85 A 83 85 A 83 85 A 85 A 83 85 A 83 85 A 83 85 A 83 85 A 85 A 83 85 A 83 85 A 83 85 A 83 85 A 85 A 83 85 A 85 A 83 85 A 85 A 83 85 A	1 83 85 A 83 1 83 85 A 83	85 A 83 85 85 A 83 85

// Example-1: RTOS based multitasking // This lab uses "QUEUE" to synchronize the tasks. // Monitor Tera-Term Window for the data movement // // Task 1 (PRODUCER) - Generates messages // Task 2 (CONSUMER) - Displays data on "Tera-Term" window // #include "RTE_Components.h" // Component selection

#include CMSIS_device_header

TRAINING MANUAL

Unit 10: Getting started with SPI Bus



```
#include "FreeRTOS.h"
                            // Keil::RTOS:FreeRTOS:Core
                         // Keil::RTOS:FreeRTOS:Core
#include "task.h"
#include <queue.h>
#include "stm32l4s5xx.h"
#include "cmsis_os2.h"
                               // CMSIS RTOS header file
#include <stdio.h>
typedef struct {
                  // object data type
                  uint8_t Idx;
                  char Buf[5];
} MSGQUEUE_OBI_t;
    _____
   Message Queue creation & usage
*_____*/
#define MSGQUEUE_OBJECTS 16
                                     // number of Message Queue Objects
osMessageQueueId_t mid_MsgQueue;
                                       // message queue id
                                      // thread id 1
osThreadId_t tid_Thread_MsgQueue1;
osThreadId_t tid_Thread_MsgQueue2;
                                      // thread id 2
void Thread_MsgQueue1 (void *argument);
                                         // thread function 1
void Thread_MsgQueue2 (void *argument);
                                         // thread function 2
int Init_MsgQueue (void) {
mid_MsgQueue = osMessageQueueNew(MSGQUEUE_OBJECTS, sizeof(MSGQUEUE_OBJ_t), NULL);
if (mid_MsgQueue == NULL) {
 ; // Message Queue object not created, handle failure
}
 tid_Thread_MsgQueue1 = osThreadNew(Thread_MsgQueue1, NULL, NULL);
if (tid_Thread_MsgQueue1 == NULL) {
 return(-1);
}
 tid_Thread_MsgQueue2 = osThreadNew(Thread_MsgQueue2, NULL, NULL);
if (tid_Thread_MsgQueue2 == NULL) {
 return(-1);
}
return(0);
}
void Thread_MsgQueue1 (void *argument) {
         MSGQUEUE_OBJ_t msg = {0};
         msg.Idx = 'A';
         msg.Buf[0] = 83;
         msg.Buf[1] = 85;
 while (1) {
                  GPIOA->ODR ^= (0x1 << 5); //PA5 ON
                  vTaskDelay(100);
```



```
osMessageQueuePut(mid_MsgQueue, &msg, 0U, 0U);
                  osThreadYield();
                                   // Suspend thread for a system tick
}
}
void Thread_MsgQueue2 (void *argument) {
         MSGQUEUE_OBJ_t msg = {0};
         osStatus_t status;
         char str_tmp[100] = {0};
 while (1) {
                  GPIOB->ODR ^= (0x1 << 14); //PB14 ON
                  vTaskDelay(100);
 status = osMessageQueueGet(mid_MsgQueue, &msg, NULL, 0U); // wait for message
 if (status == osOK) {
                                    snprintf(str_tmp, sizeof(str_tmp), "%c %d %d \n", msg.Idx, msg.Buf[0], msg.Buf[1]);
                                    int i=0;
                                    while (str_tmp[i] != '\n') 
                                                      while (!(USART1->ISR & 0x0080)) {} // Wait until Tx buffer empty
                                                                USART1->TDR = str_tmp[i];
                                                      i++; if (i > sizeof(str_tmp)) {i=0; break;}
                                    }
 }
}
}
// Initialize FreeRTOS and start the initial set of tasks.
int main(void) {
         // Enable the clock to GPIO Port A
         RCC->AHB2ENR |= 1;
                                         /* enable GPIOA clock */
         // MODE: 00: Input mode, 01: General purpose output mode
         // 10: Alternate function mode, 11: Analog mode (reset state)
         GPIOA->MODER &= (unsigned int)~(0x3 << 5*2); // Clear bit PA5
         GPIOA->MODER |= (0x1 << 5*2); // Set bit PA5 Output
         // Enable the clock to GPIO Port B
         RCC->AHB2ENR |= 0x2; //RCC_AHB2ENR_GPIOB_EN;
         // MODE: 00: Input mode, 01: General purpose output mode
              10: Alternate function mode, 11: Analog mode (reset state)
         11
         GPIOB->MODER &= (unsigned int)~(0x3 << 14*2); // Clear bit PB14
         GPIOB->MODER |= (0x1 << 14*2); // Set bit PB14 Output
```

Unit 10: Getting started with SPI Bus



```
// -----
// Configure USART1
                  // Enable the clock to GPIO Port B
                  RCC->AHB2ENR |= 0x2; //RCC_AHB2ENR_GPIOB_EN;
                  GPIOB->AFR[0] &= ~0x0F000000;
                  GPIOB->AFR[0] |= 0x07000000; /* PB6 for USART1 TX */
                  GPIOB->AFR[0] &= ~0xF0000000;
                  GPIOB->AFR[0] |= 0x7000000; /* PB7 for USART1 RX */
                  // MODE: 00: Input mode, 01: General purpose output mode
                  // 10: Alternate function mode, 11: Analog mode (reset state)
                  GPIOB->MODER &= (unsigned int)~(0x3 << 6*2); // Clear bit PB6
                  GPIOB->MODER |= (0x2 << 6*2); // Set bit PB6 Alternate
                  GPIOB->MODER &= (unsigned int)~(0x3 << 7*2); // Clear bit PB7
                  GPIOB->MODER |= (0x2 << 7*2); // Set bit PB7 Alternate
  RCC->APB2ENR |= 0x4000;
                                /* enable USART1 clock */
  USART1->CR1 = 0x000C;
                           /* enable Tx, Rx, 8-bit data */
 USART1->CR2 = 0x0000;
                           /* 1 stop bit */
 USART1->CR3 = 0x0000;
                          /* no flow control */
 USART1->BRR = 0x0023:
                                      /* 115200 baud @ 16 MHz */
 USART1->CR1 |= 0x0001; /* enable USART1 */
                  // Start the task executing
                  Init_MsgQueue();
                  vTaskStartScheduler();
// Execution will only reach here if there was not enough FreeRTOS heap memory
// remaining for the idle task to be created
         while(1) {
         }
} // main() ends here
```



Launch STM32CubeMX Development Tools

Double Click the Icon



This graphical tool is to assist the embedded system hardware initialization and setting of the relevant parameters without a direct engagement at the Registers level. Basically, the low-level initialization remains hidden for a simple, clean, and hassle free task implementations (i.e., application specific configurations).

File → New Project

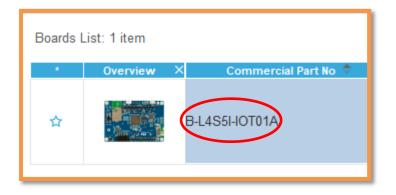
Select "Hardware Platform"

Board Selector

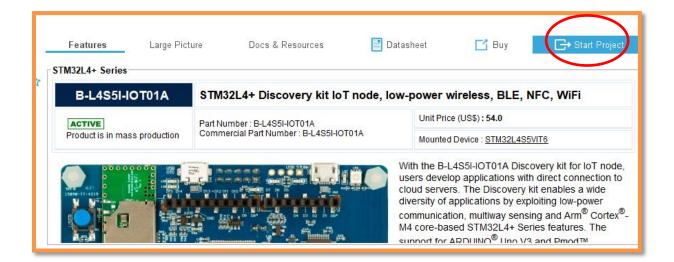
MX New Project	
MCU/MPU Selector	Board Selector Example Selector
Board Filters	
*	B
Commercial Part Number	
	B-L475E-IOT01A2
Q	B-L4S5I-IOT01A 🗘
	B-U585I-IOT02A

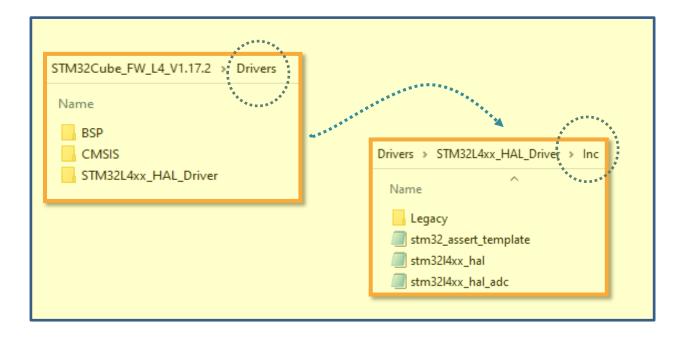


Select the "h/w board",



Click "Start Project"







Reset "Preconfigured Pinout"

- Select "Pinout & Configuration",
- Click "Pinout"
- Right Click, "Clear Pinouts"

	F	File	Window	Help		
Home >	STM32L4S5VITx - B-L4	S51-10T01A >	Untitled - Pinout 8	Configuration		
	Pinout & Configuratio	on	Clock Co	nfiguration	Project Manag	er
· · · ·			✓ Software P	acks	▲ Pinout	
	~ () A->Z	*			Undo Mode and pinout Rede mode and pinout	Ctrl-Z Ctrl-Y
Categories System C				lan 'anna Ian 'anna Ian 'anna	Keep Current Signals Placement Show User Label	Ctrl-K
			No.	land land lat	Disable All Modes	Ctrl-D
Analog	>		 An and a second s	Clear Pinouts	Ctrl-P	
Timers	>				Clear Single Mapped Signals Pins/Signals Options	set the pinout
Connectivi	ity >	Extense, P. Jones J. Hall, H., Y. (1996), 1171 Status, P. Status, P.	₫		List Pinout Compatible MCUs	Alt-L
Multimedia	a >	WHAL, W., BE, DA'N, DF, NAMA, UY, BE BATTRE, DF - AND - BATTRE, DEF - AND - AN		100, 201, 201, 201 100, 201, 201 100, 201, 201 100, 201, 201 100, 201, 201 100, 2	Export pinout with Alt. Functions Export pinout without Alt. Function	IS CtrI-U
Security	>	NUMBER OF STREET		C C C C C C C C C C C C C C C C C C C	Reset used GPIOs	Alt-G
Security		100, 10 (100) 100, 10 (100) 100, 10 (100)		and a second sec	Set unused GPIOs	Ctrl-G
Computing	g >		STM32L4S5VIT:	K Billion Allower and Allower	Pinout View Colors	
	e and Software Pac >	ente, ara partecha parte ante, ara partecha partecha partecha partecha parte antecha partecha	LQFP100		Layout reset	
			A consense Sector of the sector of the secto			



TRAINING MANUAL

Unit 10: Getting started with SPI Bus



Configure FreeRTOS

Pinout & Configuratio	Pinout & Configuration		Clock Configuration			Project Manager		
		✓ Soft	ware Packs		🗸 Pin	nout		
۵ 🗸 🖉			FREERTO	S Mode and	Configuration			
Categories A->Z Multimedia / Security >	Interface	CMSIS_V2		Mode				~
Computing >	Poset C	onfiguration	-	Configuration	1	-	_	-
¢ FATFS		Vlutexes er Constants	⊗ Events ⊘ Tasks	and Queues	0 1	FreeRTOS H	eap Usage and Semap	phores
FP-ATR-SIGFOX1 FREERTOS FREERTOS FI-CUBE-Cesium FI-CUBE-UNISONRTOS FI-CUBE-embOS FI-CUBE-embOS FI-CUBE-wolfSSL FI-Cube-SoM-uGOAL TOUCHSENSING	Tasks Tasks defaultTask			nclude param <mark>Code Gen</mark> Default		S ,	Advanced se	
USB_DEVICE USB_HOST	 _ Queues						Add	Delete

Configure / Edit default Tasks

Edit Task					
Task Name	Task1				
Priority	osPriorityNorm	al			
Stack Size (Words)	128				
Entry Function	StartTask1				
Code Generation Optic	on Default				
Parameter	NULL	New Task	c		×
Allocation	Dynamic	Tas	k Name	Task2	
Buffer Name	NULL	Prio	rity	osPriorityBelowNormal	\sim
Control Block Name	NULL	Sta	ck Size (Words)	128	
ок	Cancel	Entr	ry Function	StartTask2	
		Cod	le Generation Option	Default	\sim
		Para	ameter	NULL	
		Allo	cation	Dynamic	~
Add a 2nd Task		Buff	er Name	NULL	
• Click "Add"		Con	trol Block Name	NULL	
			ОК	Cancel	



Optional – Enable library (new version)

	Config	guration			
Reset Configuratio	in				
Mutexes	🥺 Events	0	Free	eRTOS Heap Usage	
🥝 User Constants	🧭 Tasks and	Queues	0	Timers and Semaphores	
😔 Config paramete	ers 🥺 Include	parameter	rs	🥝 Advanced settings	
Configure the below pa	arameters :				
Q Search (Ctrl+F)	0 0			0	
\sim Newlib settings (se	ee parameter des				
USE_NEW	LIB_REENTRANT	Enabled			
 Project settings (see parameter des 					
Use FW pa	ick heap file	Enabled			
	-				

Choose Timer as the HAL Timebase Source (Instead of Systick)

Q ~	٥	SYS Mode and Configuration
Categories A->Z		Mode
System Core 🛛 🗸 🗸		Debug Disable
÷		System Wake-Up 1
DMA		System Wake-Up 2
GPIO IWDG		□ System Wake-Up 3
NVIC		□ System Wake-Up 4
		System Wake-Up 5
		Power Voltage Detector In Disable
WWDG		VREFBUF Mode Disable
		Timebase Source TIM2
Analog >		

NOTE:

The **SysTick** is a special timer in most ARM processors that's generally reserved for operating system purposes. By default, SysTick will be used for things like HAL_Delay() and HAL_GetTick(). As a result, the STM32 HAL framework gives SysTick a very high priority. However, **FreeRTOS** needs SysTick for its scheduler, and it requires SysTick to be a much lower priority. **Therefore**, a quick work around is to use a **Timer** as a Time-base source in the cases of freeRTOS.

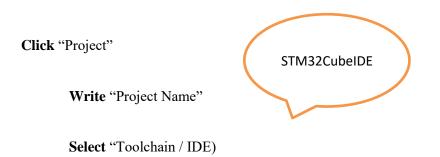


Click "Pinout and Configuration"

Right Click on "**PA5**", and declare it as a "**GPIO Output**" signal **Right Click** on "**PB14**", and declare it as a "**GPIO Output**" signal

PC3 STM32L4S5VITx VSSA LQFP100 VREF. LQFP100		PD11 PD10 PD9 PD8 PB15
PEID Output PAG VBS VBB VBB PAG VBS VBS VBB VBB PAG VBS PAG	PB14 TSC_G1_IO3 USART3_DE USART3_RTS GPIO_Input GPIO_Output GPIO_Output GPIO_Analog EVENTOUT GPIO_EXTI14	B13 B12 GPIO_Output

Click "Project Manager"



Click "Generate Code"



Open "Project" in "STM32CubeIDE"

Update "main.c"

```
void StartTask1(void *argument)
{
    /* USER CODE BEGIN 5 */
    /* Infinite loop */
    for(;;)
    {
        GPIOB->ODR ^= (0x1 << 14); //PB14 ON
        osDelay(100);
    }
    /* USER CODE END 5 */
}</pre>
```

```
void StartTask2(void *argument)
{
   /* USER CODE BEGIN StartTask2 */
   /* Infinite loop */
   for(;;)
   {
        GPIOA->ODR ^= (0x1 << 5); //PA5 ON
        osDelay(50);
   }
   /* USER CODE END StartTask2 */
}</pre>
```

- Build "Project" (In STM32CubeIDE)
- Flash "binary code" on the h/w board (In STM32CubeProgrammer)
- > Reset h/w board (By pressing switch/button on the board)
- Monitor "LEDS" toggling on the h/w board

Reference

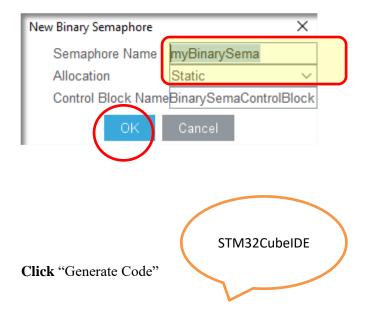
https://www.keil.com/pack/doc/CMSIS/RTOS2/html/group_CMSIS_RTOS_SemaphoreMgmt.html

Task-4/b



Add "Semaphore" feature (Continuation to previous task)

Pinout & C	Configuration	Cloc	k Configuration		Project Manager	
		✓ Softw	vare Packs	✓ Pinout		
۹	0	\frown	FREERTOS Mode and	I Configuration		
Categories A->Z			Mode			
USAR12 USART3		Interface CMSIS_V2			\sim	
USB_OTG_FS						
	- IX		Configuratio	on		
Multimedia	<u> </u>	Reset Configuration				
Security	>	Mutexes	🤡 Events	📀 FreeR	TOS Heap Usage	
O		User Constants	Tasks and Queues		Timers and Semaphores	
Computing		Config parameters	🥝 Include parar	meters	Advanced settings	
Middleware and Software		mers Fimer Name Callback	Type Code Gener	at Parameter	Allocation Control Block	
÷						
FATFS					Add Delete	
FP-ATR-SIGFOX1	- Bit	nary Semaphores				
I-CUBE-Cesium		Semaphore Name	Allocation	n	Control Block Name	
I-CUBE-UNISONRTO	os 🚺					
I-CUBE-embOS					Add Delete	
I-CUBE-wolfSSL						





Open "Project" in "STM32CubeIDE"

Update "main.c"

```
void StartTask1(void *argument)
{
    /* USER CODE BEGIN 5 */
    /* Infinite loop */
    for(;;)
    {
        osSemaphoreRelease(myBinarySemaHandle);
        GPIOB->ODR ^= (0x1 << 14); //PB14 ON
        osDelay(100);
    }
    /* USER CODE END 5 */
}</pre>
```

```
void StartTask2(void *argument)
{
   /* USER CODE BEGIN StartTask2 */
   /* Infinite loop */
   for(;;)
   {
      osSemaphoreAcquire(myBinarySemaHandle, osWaitForever);
      GPIOA->ODR ^= (0x1 << 5); //PA5 ON
   }
   /* USER CODE END StartTask2 */
}</pre>
```

- Build "Project" (In STM32CubeIDE)
- Flash "binary code" on the h/w board (In STM32CubeProgrammer)
- Reset h/w board (By pressing switch/button on the board)
- Monitor "LEDS" toggling on the h/w board

Task-4/c



Add "Queue" feature (Continuation to previous task)

Pinout & Co	Clock	Configuration		Project I	Manager	
		✓ Softwar	e Packs	🗸 Pinot	ut	
Q	٥	\frown	FREERTOS Mode and	d Configuration		
Categories A->Z			Mode			
USAR12 USART3 USB_OTG_FS	Inte	fade CMSIS_V2				\checkmark
			Configurati	ion		
Multimedia	> Res	set Configuration				
Security	>	⊗ Mutexes	Sevents	🤗 Fre	eRTOS Heap Usage	
Computing	>	User Constants Config parameters	✓ Tasks and Queue ✓ Include para		⊘ Timers and Sema ⊘ Advanced s	
Middleware and Software I	Packs Y	ask Name Stack Siz.	Entry FunCode Ger	n Parameter 7	Allocation Buffer Nan	ne Control Bl
\$	Task1	128	StartTask1 Default		ynamic NULL	NULL
FATFS	Task2	128	StartTask2 Default	NULL D	ynamic NULL	NULL
FP-ATR-SIGFOX1 FREERTOS I-CUBE-Cesium I-CUBE-UNISONRTOS	_ Queue	8			Add	Delete
I-CUBE-embOS	Que	ue Name Queue Size	Item Size	Allocation	Buffer Name Con	trol Block N
I-CUDE-WOIRSSL I-CUDE-SoM-uGOAL TOUCHSENSING USB_DEVICE USB_HOST X-CUBE-AI					Add	Delete

Edit Queue	X
Queue Name	myQueue
Queue Size	16
Item Size	uint16_t
Allocation	Static 🗸
Buffer Name	myQueueBuffer
Buffer size	32
Control Block Nam	nemyQueueControlBlock
ОК	Cancel

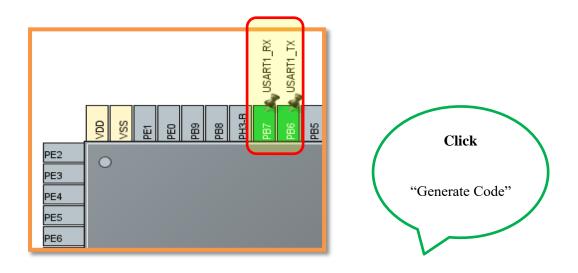


Add "UART1" feature



Configure USART Pins

۹	٢	USART1 Mode and Configuration
Categories A->Z		Mode
I2C3 I2C4 IRTIM LPUART1 OCTOSPI1 OCTOSPI2		Mode Asynchronous
SDMMC1 SPI1 SPI2 SPI3 UART4 UART5 VISART1		Configuration Reset Configuration ◇ NVIC Settings ◇ DMA Settings ◇ GPIO Settings ◇ Parameter Settings ◇ User Constants
USART2 USART3 USB_OTG_FS Multimedia		Search Signals Search (Ctrl+F) Pin Na Signal on GPIO outp GPIO mode GPIO Pull Maximum Fast Mode
Security >	_ (PB6 USART1_TX n/a Alternate No pull-up Very High Disable PB7 USART1_RX n/a Alternate No pull-up Very High Disable





Open "Project" in "STM32CubeIDE"

Update "main.c"

```
#include <stdio.h>
```

{

}

```
typedef struct { // object data type
uint8_t Buf[32];
uint8_t Idx;
} MSGQUEUE_OBJ_t;
```

```
void StartTask1(void *argument)
```

```
MSGQUEUE_OBJ_t msg;

while (1) {

GPIOB->ODR ^= (0x1 << 14); //PB14 ON

osDelay(100);

msg.Buf[0] = 0x55U; // do some work...

msg.Idx = 0U;

osMessageQueuePut(myQueueHandle, &msg, 0U, 0U);

osThreadYield(); // suspend thread

}
```

```
void StartTask2(void *argument)
{
        char str_tmp[100] = "";
                                 // To display formatted messages
        MSGQUEUE_OBJ_t msg;
        osStatus_t status;
        while (1) {
        status = osMessageQueueGet(myQueueHandle, &msg, NULL, 0U); // wait for message
                 if (status == osOK) {
                          GPIOA->ODR ^= (0x1 << 5); //PA5 ON
                          osDelay(50);
                          snprintf(str_tmp,100," %d \n\r", msg.Buf[0]);
                          HAL_UART_Transmit(&huart1,( uint8_t * )str_tmp,sizeof(str_tmp),1000);
                 }
        }
}
```



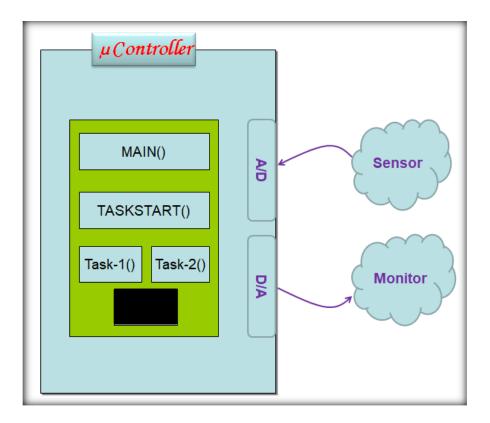
- Build "Project" (In STM32CubeIDE)
- Flash "binary code" on the h/w board (In STM32CubeProgrammer)
- **Reset h/w board** (By pressing switch/button on the board)
- Monitor "LEDS" toggling on the h/w board and "Tera-Term" Console window for the messages



Exercise

This task demonstrates how to:

- Configure GPIO ports
- Create multiple Tasks in RTOS (Free RTOS)
- Generate A/D data (Task 1)
- Generate D/A data (Task 2)



Construct a C program to demonstrate and verify the design behaviour.



Review Questions

Q1.

Q2

Q3