Somaiya Vidyavihar University
K J Somaiya School of Engineering

Batch: A2	Roll No.: 16010322014
Experiment / assignment / tutorial No. 07	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

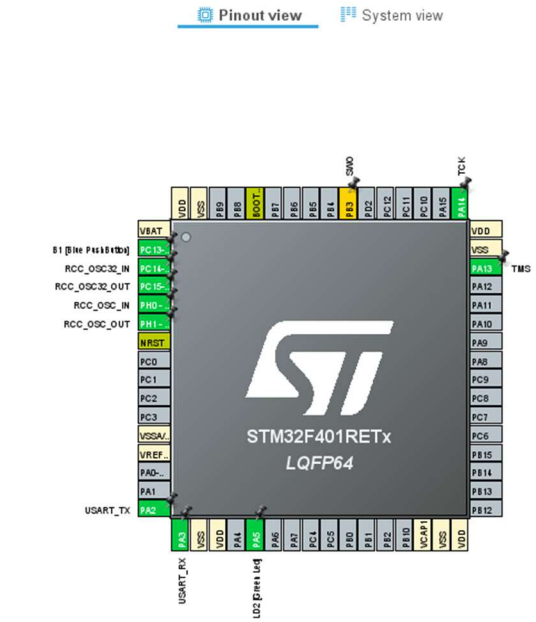
TITLE: Integration of FreeRTOS on STM32 platform
AIM: Understand and explore RTOS functionalities in STM32 environment.

1. **OUTCOME:** Implement Open Source RTOS for resource sharing using inter task communication and explore advances and recent trends in embedded systems

Hardware tools required:
Microcontroller Development Board (e.g., STM32, NXP, or any Cortex-M based board)
Debug Probe/Debugger (e.g., SEGGER J-Link) for real-time debugging and tracing
PC/Workstation to run IDE and debugging tools
USB Cable for programming and debugging connection

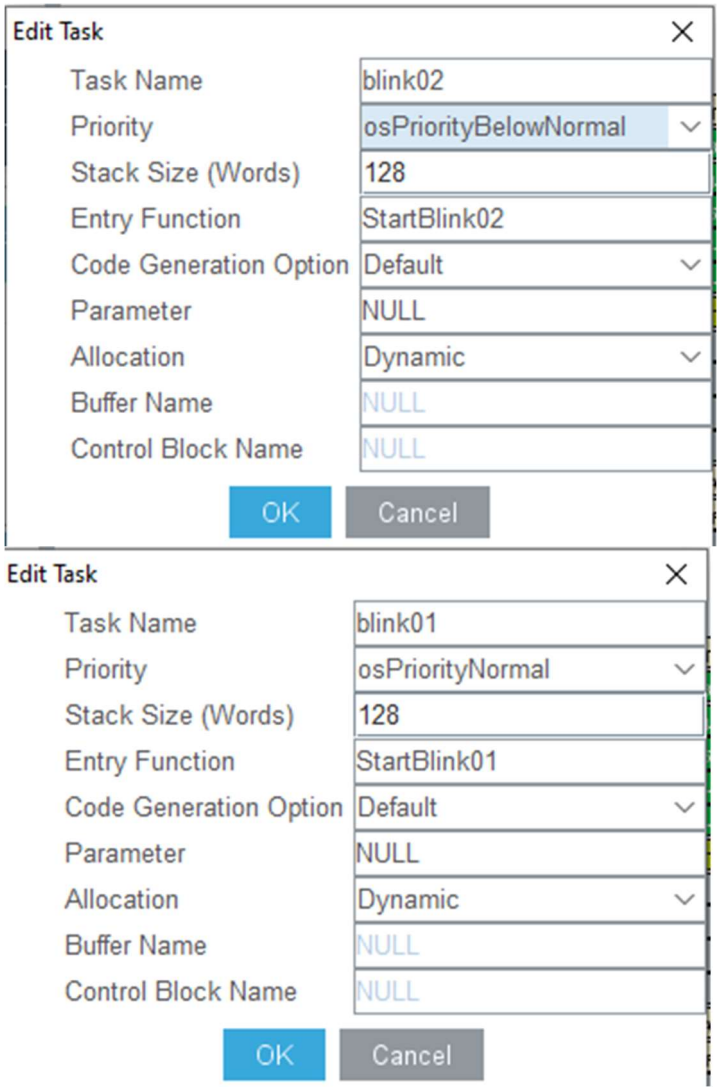
Software tools required:
IDE (e.g., STM32CubeIDE, Keil uVision, IAR Embedded Workbench)
FreeRTOS Kernel (usually included or imported into the IDE)
SEGGER SystemView Software for real-time tracing and analysis
SEGGER J-Link Software and Drivers for debugger interface
STM32CubeMX (optional, for configuring peripherals and middleware)

Task1: Create two task blink1 and blink2 using FreeRTOs
Assign different delays to toggle it
Take the screen shots and paste here


Pinout view
System view
STM32F401RETx
LQFP64
USART_TX
USART_RX
TMS
TCK
SWO


Pinout & Configuration
Clock Configuration
Project Manager
Tools
Software Packs
Pinout
SYS Mode and Configuration
Mode
Debug Serial Wire
System Wake-Up
Configuration
Warning: This peripheral has no parameters to be configured.
Categories
A->Z
Analog
Timers
RTC
TIM1
TIM2
TIM3
TIM4
TIM5
TIM9
TIM10
TIM11
Connectivity
Multimedia
Computing
Middleware
FATFS
FREERTOS
LIBJPEG
MBEDTLS
Pinout view
System view
STM32F401RETx
LQFP64

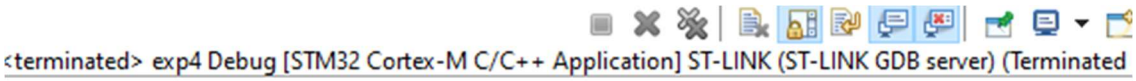
FREERTOS Mode and Configuration
Mode
Interface
CMSIS_V2



Task2: Explore the usage of Timers and NVIC using STM32cube paste the screenshots here

```
/* USER CODE END Header_StartBlink02 */
void StartBlink02(void *argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
      HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_5);
    osDelay(500);
  }
  osThreadTerminate(NULL);
  /* USER CODE END 5 */
}

/* USER CODE BEGIN Header_StartBlink01 */
/**
* @brief Function implementing the blink01 thread.
* @param argument: Not used
* @retval None
*/
/* USER CODE END Header_StartBlink01 */
void StartBlink01(void *argument)
{
  /* USER CODE BEGIN StartBlink01 */
  /* Infinite loop */
  for(;;)
  {
      HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_5);
    osDelay(600);
  }
  osThreadTerminate(NULL);
  /* USER CODE END StartBlink01 */
}
```



```
<terminated> exp4 Debug [STM32 Cortex-M C/C++ Application] ST-LINK (ST-LINK GDB server) (Terminated

STMicroelectronics ST-LINK GDB server. Version 6.0.0
Copyright (c) 2021, STMicroelectronics. All rights reserved.

Starting server with the following options:
        Persistent Mode            : Disabled
        Logging Level              : 1
        Listen Port Number         : 61234
        Status Refresh Delay       : 15s
        Verbose Mode               : Disabled
        SWD Debug                  : Enabled
        InitWhile                  : Enabled

Waiting for debugger connection...
Debugger connected
Waiting for debugger connection...
Debugger connected
Waiting for debugger connection...
      -------------------------------------------------------------------
                      STM32CubeProgrammer v2.9.0-RC01
      -------------------------------------------------------------------

ST-LINK SN  : 0669FF535455886687144022
ST-LINK FW  : V2J39M27
Board       : NUCLEO-F401RE
Voltage     : 3.25V
SWD freq    : 4000 KHz
Connect mode: Under Reset
Reset mode  : Hardware reset
Device ID   : 0x433
Revision ID : Rev Z
Device name : STM32F401xD/E
Flash size  : 512 KBytes
Device type : MCU
Device CPU  : Cortex-M4
BL Version  : --
```

Source code:

```
41
42  /* Private variables ---------------------------------------
43  UART_HandleTypeDef huart2;
44
45  /* Definitions for blink02 */
46  osThreadId_t blink02Handle;
47  const osThreadAttr_t blink02_attributes = {
48    .name = "blink02",
49    .stack_size = 128 * 4,
50    .priority = (osPriority_t) osPriorityBelowNormal,
51  };
52  /* Definitions for blink01 */
53  osThreadId_t blink01Handle;
54  const osThreadAttr_t blink01_attributes = {
55    .name = "blink01",
56    .stack_size = 128 * 4,
57    .priority = (osPriority_t) osPriorityNormal,
58  };
59  /* USER CODE BEGIN PV */
60
61  /* USER CODE END PV */
62
63  /* Private function prototypes -----------------------------
64  void SystemClock_Config(void);
65  static void MX_GPIO_Init(void);
66  static void MX_USART2_UART_Init(void);
67  void StartBlink02(void *argument);
68  void StartBlink01(void *argument);
69
70  /* USER CODE BEGIN PFP */
71
72  /* USER CODE END PFP */
73
74  /* Private user code ---------------------------------------
75  /* USER CODE BEGIN 0 */
76
77  /* USER CODE END 0 */
78
79  /**
80    * @brief  The application entry point.
81    * @retval int
82    */
83  int main(void)
84  {
85    /* USER CODE BEGIN 1 */
86
87    /* USER CODE END 1 */
88
```

```
161 /**
162   * @brief System Clock Configuration
163   * @retval None
164   */
165 void SystemClock_Config(void)
166 {
167   RCC_OscInitTypeDef RCC_OscInitStruct = {0};
168   RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
169
170   /** Configure the main internal regulator output voltage
171   */
172   __HAL_RCC_PWR_CLK_ENABLE();
173   __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);
174   /** Initializes the RCC Oscillators according to the specified parameters
175   * in the RCC_OscInitTypeDef structure.
176   */
177   RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
178   RCC_OscInitStruct.HSIState = RCC_HSI_ON;
179   RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
180   RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
181   RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
182   RCC_OscInitStruct.PLL.PLLM = 16;
183   RCC_OscInitStruct.PLL.PLLN = 336;
184   RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
185   RCC_OscInitStruct.PLL.PLLQ = 7;
186   if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
187   {
188     Error_Handler();
189   }
190   /** Initializes the CPU, AHB and APB buses clocks
191   */
192   RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
193                                 |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
194   RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
195   RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
196   RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
197   RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
198
199   if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
200   {
201     Error_Handler();
202   }
203 }
204
205 /**
206   * @brief USART2 Initialization Function
207   * @param None
208   * @retval None
```

Post lab questions

1. **What is CMSIS-RTOS?**
   CMSIS-RTOS (Cortex Microcontroller Software Interface Standard - Real-Time Operating System) is a standardized API layer defined by ARM for real-time operating systems on Cortex-M microcontrollers. It provides a consistent interface for RTOS features like threads, timers, semaphores, and message queues, allowing easier portability and code reuse across different RTOS implementations.

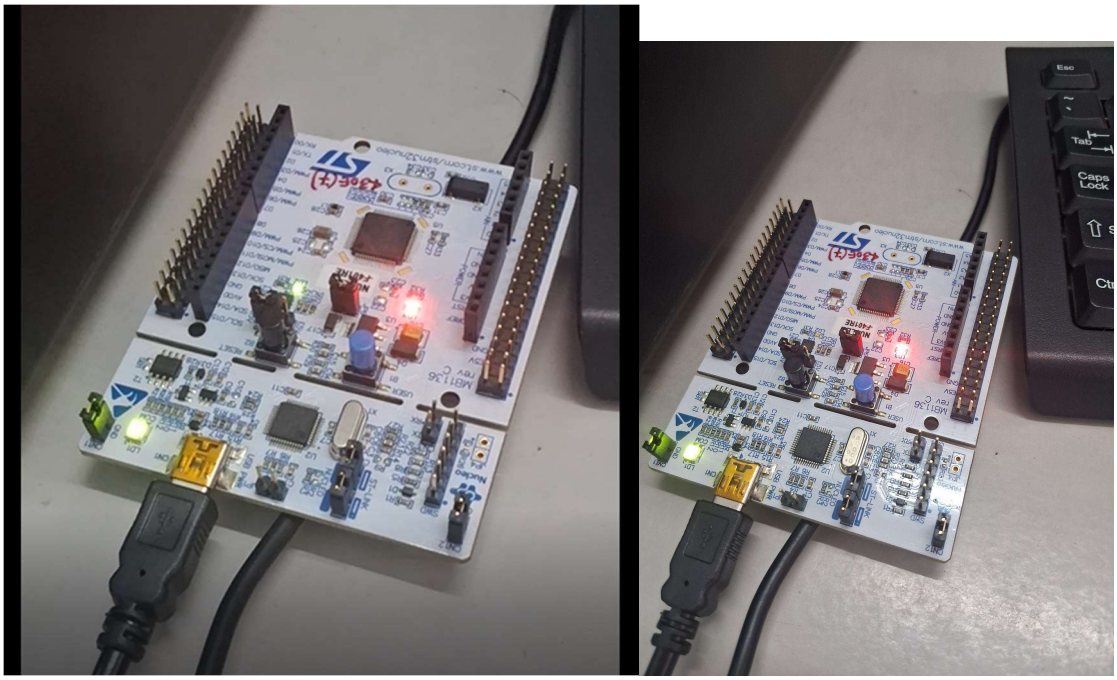
2. **List the kernel objects present in Cube MX IDE.**
   When you configure an RTOS in STM32CubeMX, the following kernel objects are commonly available:

- Threads (Tasks)
- Mutexes
- Semaphores
- Timers
- Message Queues
- Event Flags (Event Groups)
- Memory Pools
- Mail Queues

These objects allow you to manage task synchronization, communication, and timing within the RTOS.

3. **FreeRTOS vs.CMSIS-RTOS**
   FreeRTOS is an open-source and widely popular real-time operating system known for its rich and detailed API, which offers high configurability and direct integration with hardware. It supports a broad range of hardware vendors, making it highly portable within vendor ecosystems. On the other hand, CMSIS-RTOS is an API standard and interface specification developed by ARM that provides a simplified and standardized abstraction layer over various RTOS kernels, including FreeRTOS. While CMSIS-RTOS offers easier middleware integration and uniformity across different platforms, its flexibility is limited by the underlying RTOS it wraps. FreeRTOS benefits from a large and active community, whereas CMSIS-RTOS enjoys strong support from ARM and multiple vendors.

Observations: (students are expected to copy the screen shot of the obtained result)



Mention and Describe the FreeRTOS APIs related to Semaphore

1. xSemaphoreCreateBinary()
   Creates a binary semaphore with two states: available or not. Used for signaling between tasks or between an interrupt and a task.

2. xSemaphoreCreateCounting(UBaseType_t uxMaxCount, UBaseType_t uxInitialCount)
   Creates a counting semaphore that counts from 0 to uxMaxCount, useful for managing multiple instances of a resource.

3. xSemaphoreTake(SemaphoreHandle_t xSemaphore, TickType_t xTicksToWait)
   Attempts to acquire the semaphore, blocking for xTicksToWait ticks if unavailable. Returns pdTRUE if successful.

4. xSemaphoreGive(SemaphoreHandle_t xSemaphore)
   Releases the semaphore and unblocks any waiting tasks.

5. *xSemaphoreGiveFromISR(SemaphoreHandle_t xSemaphore, BaseType_t pxHigherPriorityTaskWoken)
   Safely releases the semaphore from an interrupt service routine, optionally triggering a context switch.

6. *xSemaphoreTakeFromISR(SemaphoreHandle_t xSemaphore, BaseType_t pxHigherPriorityTaskWoken)
   Attempts to acquire the semaphore from an ISR, safe for interrupt context.

Conclusion:

FreeRTOS semaphore APIs enable efficient synchronization and communication between tasks and interrupts. They provide flexible mechanisms for resource management and safe operation in both task and interrupt contexts.

Signature of faculty in-charge with date