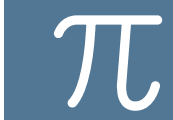
Introduction to
Real-Time Operating Systems

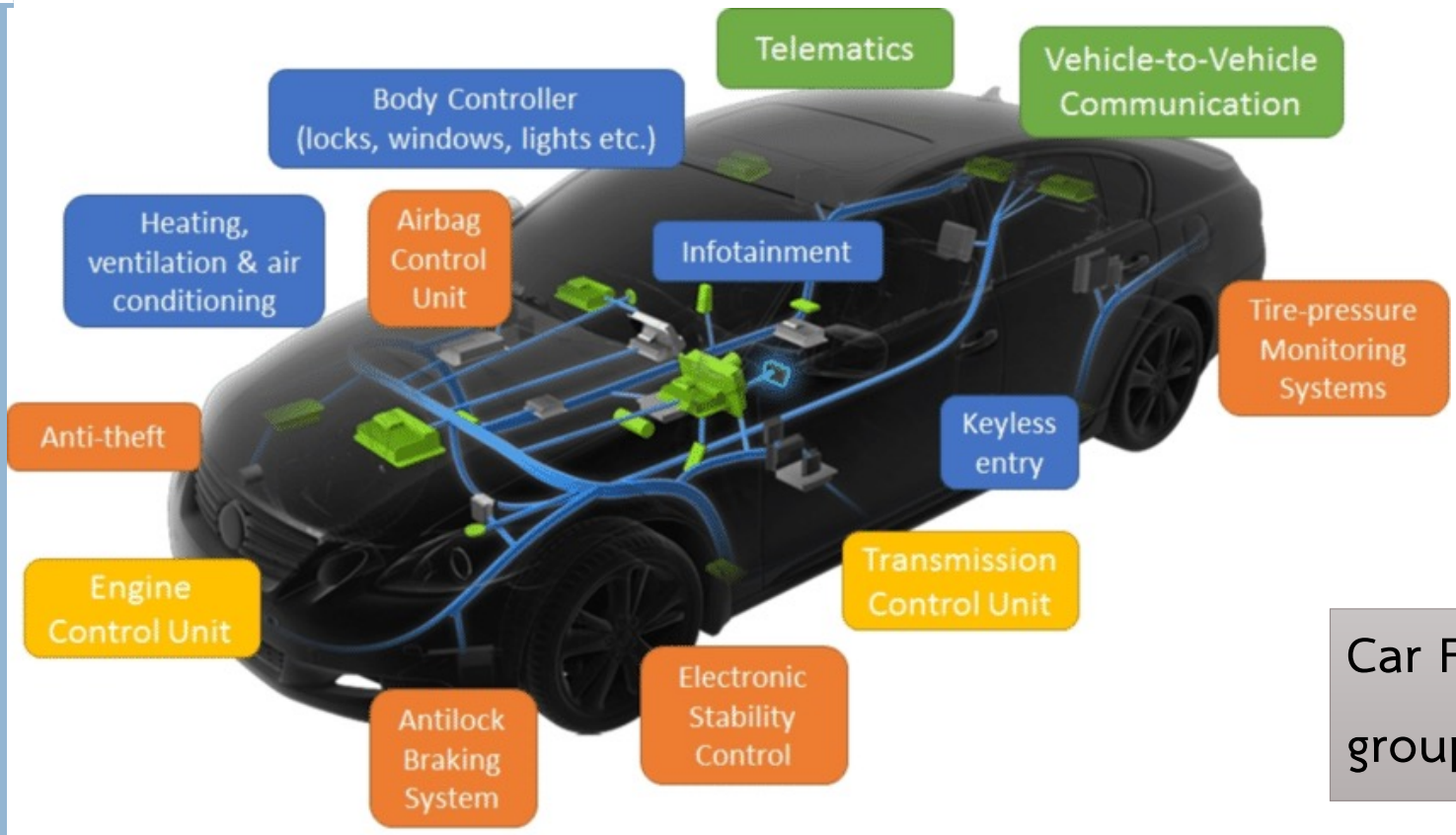


Agenda

1. Embedded System
2. mbed
3. Digital I/O
4. Serial / UART
5. Real Time Operating System
6. mbedOS
7. Producer Consumer Problem

Real Time Embedded System

Real Time Embedded System



Car Functions grouped by ECU

https://www.researchgate.net/publication/320198036

What is an RTOS

An RTOS is an operating system specialized for real time operations. In order to be classifiable as an RTOS an operating system must:

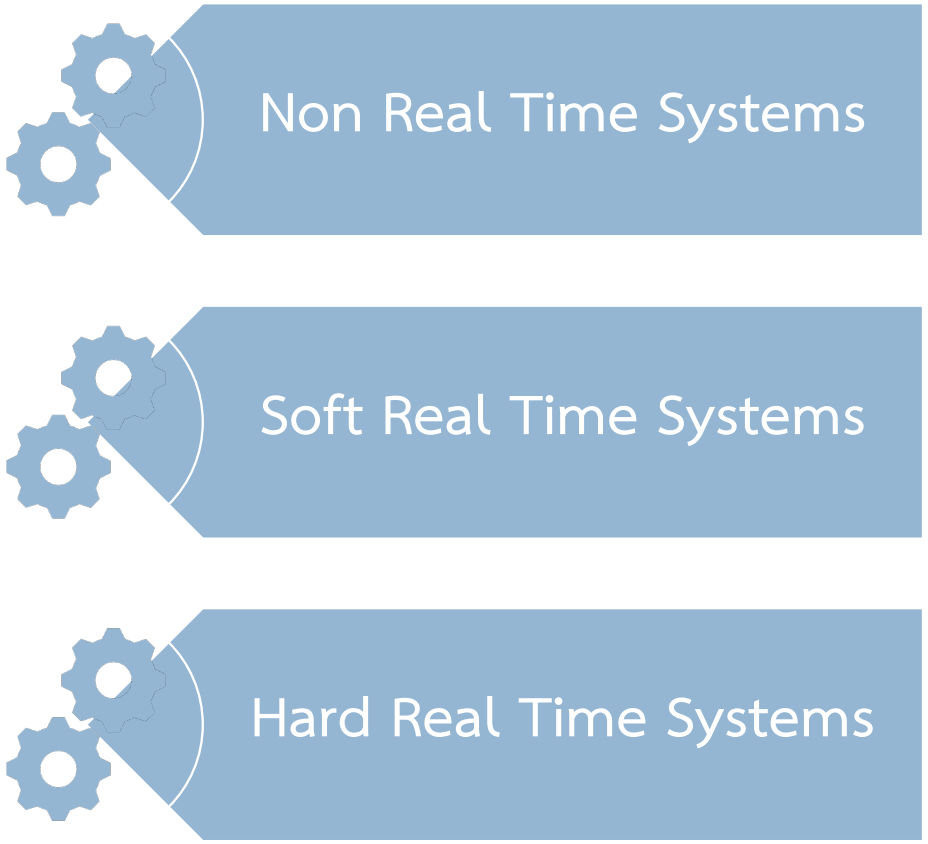
Have response time **predictability**.

Be **deterministic**.

Other qualities like speed, features set, small size etc, while important, are not what really characterize an RTOS.

http://www.chibios.org/dokuwiki/doku.php?id=chibios:articles:rtos_concepts

Systems Classification



Non Real Time Systems

- A non real time system is a system where there are **no deadlines involved**. Non-RT systems could be described as follow:

- *"A non real time system is a system where the programmed reaction to an event will certainly **happen sometime in the future**".*

Soft Real Time Systems

- A Soft real time system is a system where not meeting a deadline **can have undesirable but not catastrophic effects**, a performance degradation for example. SRTs could be described as follow:

- *"A soft real time system is a system where the programmed reaction to an event is **almost always completed within a known finite time**".*

Hard Real Time Systems

- An Hard Real Time (HRT) system is a system where not meeting a deadline can **have catastrophic effects**. HRT systems require a much more strict definition and could be described as follow:

- *"An hard real time system is a system where the programmed reaction to an event **must be guaranteed** to be completed within a known finite time".*

Examples

Soft Real Time Systems

- DVD players
- Portable music players
- Virtual reality
- RTS games
- Car navigation

Hard Real Time Systems

- Car ECUs
- Missile control
- Nuclear reactor
- Motor control

RTOS Functions

- Scheduling, States and Priorities
- Interrupts handling

Good RTOS Must be

- Response Time
  - Interrupt latency
  - Threads fly-back time
  - Context switch time
- Jitter
- Size
- Reliability
- Synchronization Primitives

RTOS

https://en.wikipedia.org/wiki/Comparison_of_real-time_operating_systems

Contiki-6LOWPAN

Texas Instruments

µC/OS™
RTOS and Stacks

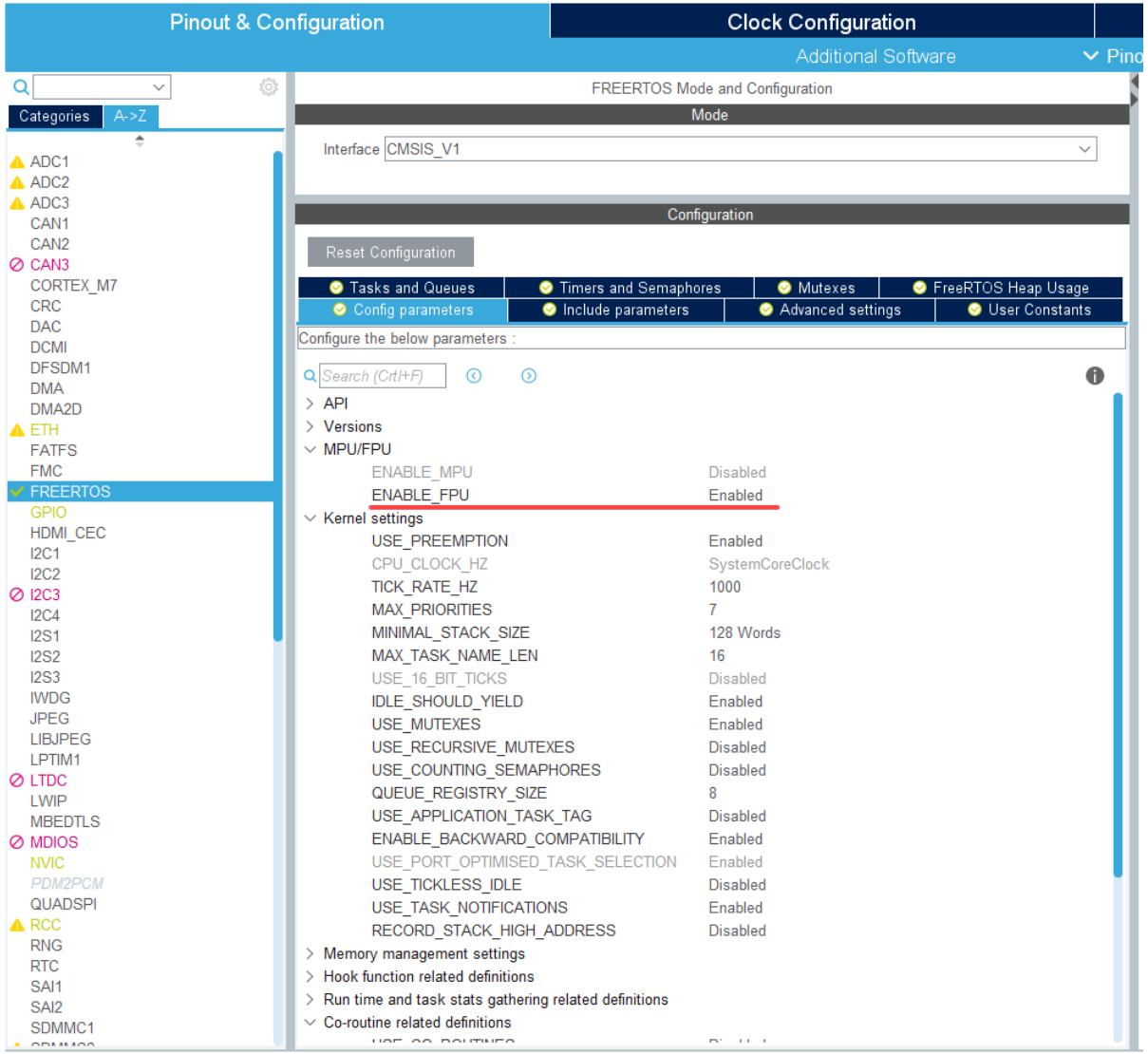
VXWORKS

rtos.com

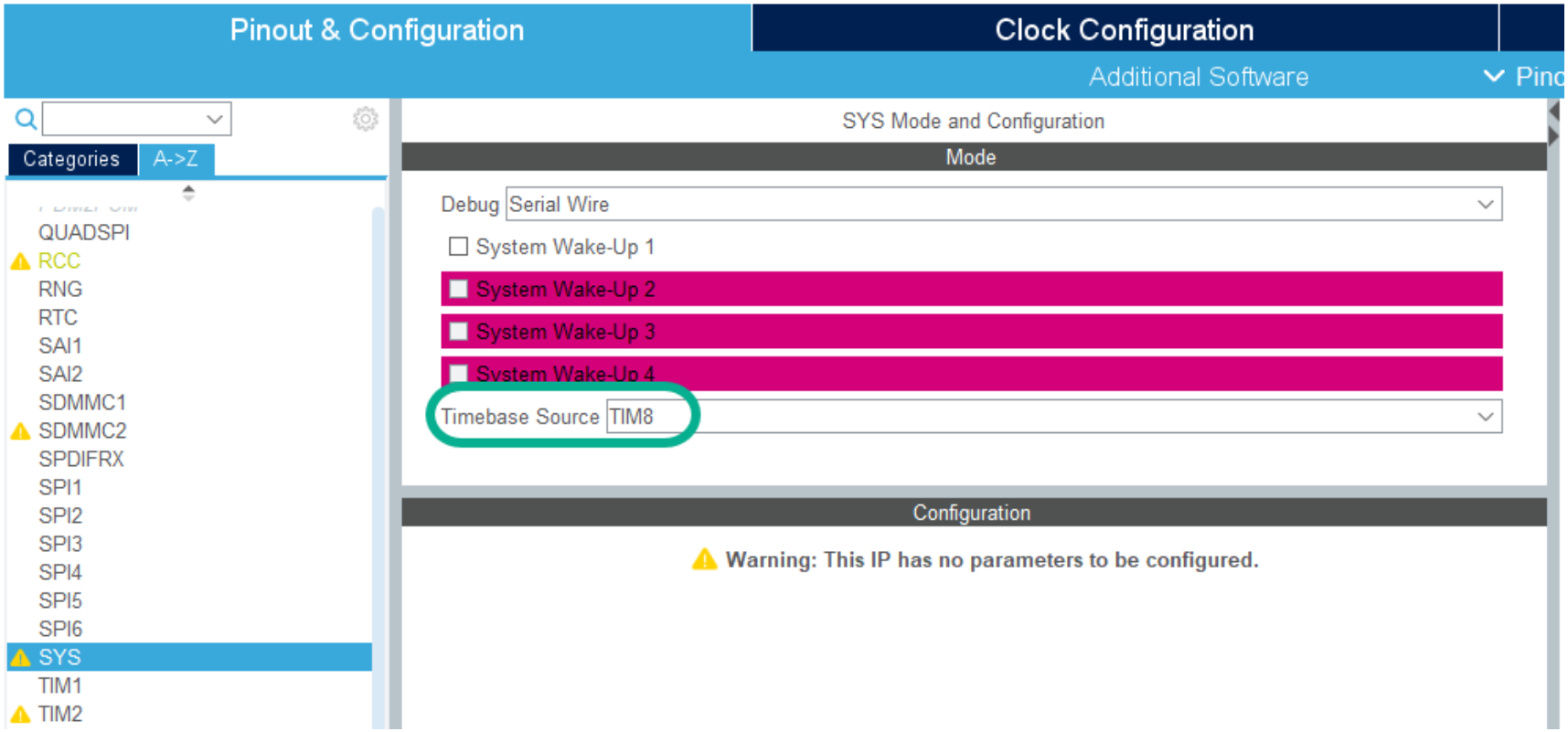
Workshop 1.1

Create LED Blinking Task

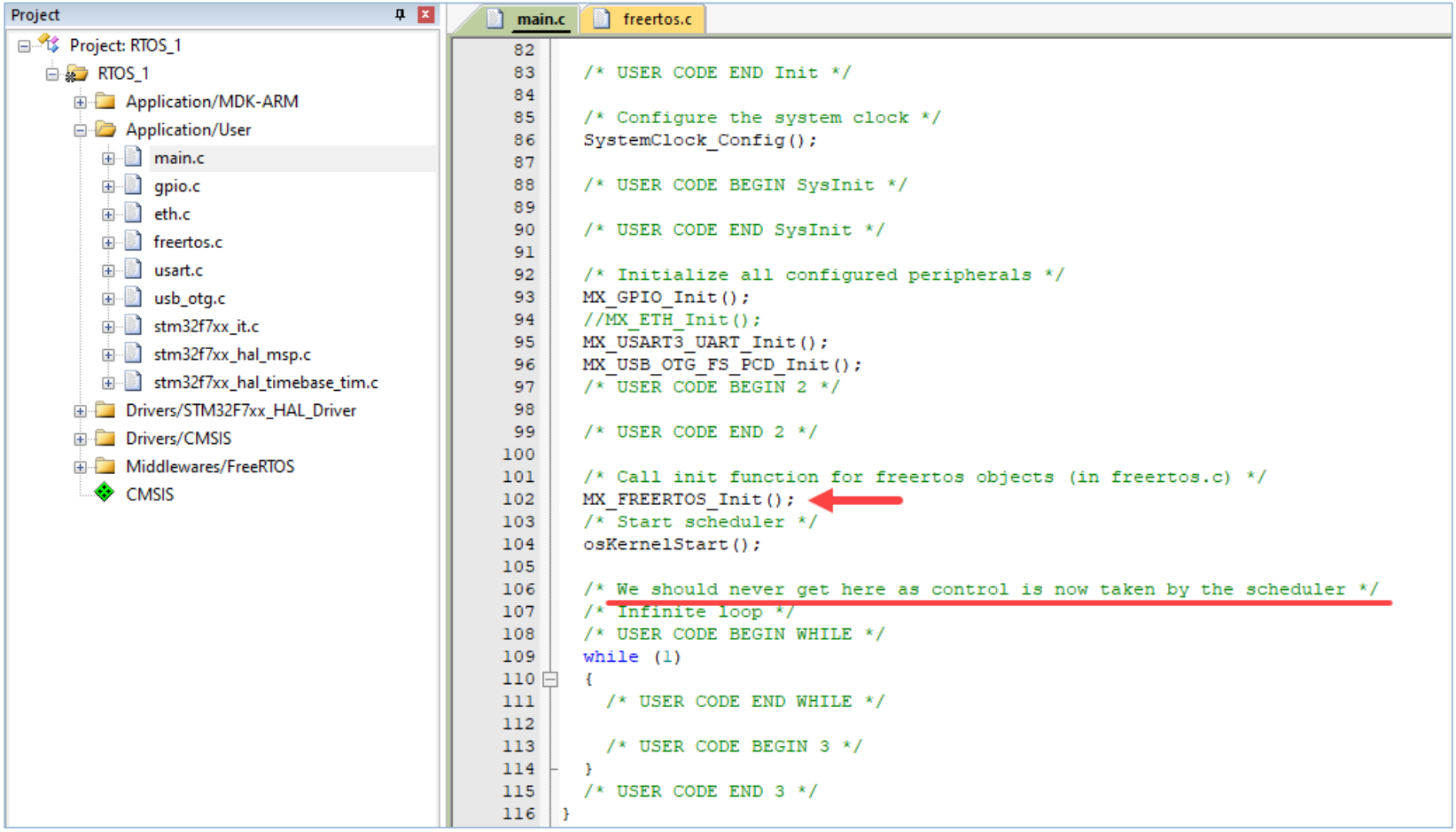
Enable FreeRTOS in STM32CubeMX



Change Timer for RTOS



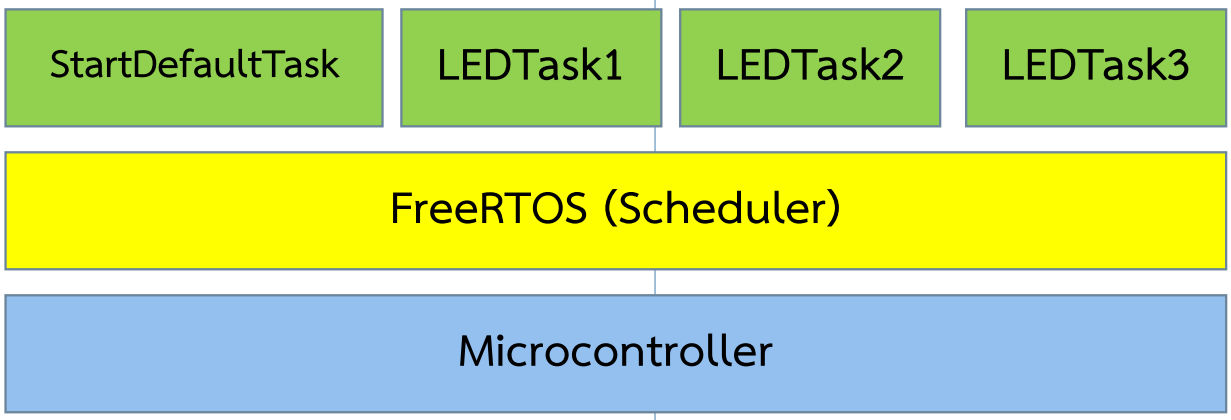
Nothing in main.c



main.c freertos.c

```
80  /* USER CODE END GET_IDLE_TASK_MEMORY */
81
82  /**
83    * @brief  FreeRTOS initialization
84    * @param  None
85    * @retval None
86    */
87  void MX_FREERTOS_Init(void) {
88    /* USER CODE BEGIN Init */
89
90    /* USER CODE END Init */
91
92    /* USER CODE BEGIN RTOS_MUTEX */
93    /* add mutexes, ... */
94    /* USER CODE END RTOS_MUTEX */
95
96    /* USER CODE BEGIN RTOS_SEMAPHORES */
97    /* add semaphores, ... */
98    /* USER CODE END RTOS_SEMAPHORES */
99
100   /* USER CODE BEGIN RTOS_TIMERS */
101   /* start timers, add new ones, ... */
102   /* USER CODE END RTOS_TIMERS */
103
104   /* USER CODE BEGIN RTOS_QUEUES */
105   /* add queues, ... */
106   /* USER CODE END RTOS_QUEUES */
107
108   /* Create the thread(s) */
109   /* definition and creation of defaultTask */
110   osThreadDef(defaultTask, StartDefaultTask, osPriorityNormal, 0, 128);
111   defaultTaskHandle = osThreadCreate(osThread(defaultTask), NULL);
112
113   /* USER CODE BEGIN RTOS_THREADS */
114   /* add threads, ... */
115   xTaskCreate(LEDTask1, "Blink LD1", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY, &xTask1);
116   xTaskCreate(LEDTask2, "Blink LD2", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY, &xTask2);
117   xTaskCreate(LEDTask3, "Blink LD3", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY, &xTask3);
118
119   /* USER CODE END RTOS_THREADS */
120
121 }
```

Init FreeRTOS



FreeRTOS API

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

https://www.freertos.org/a00106.html

Table 1. FreeRTOS™ API

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

xTaskCreate()

```
BaseType_t xTaskCreate(     TaskFunction_t pvTaskCode,
                            const char * const pcName,
                            configSTACK_DEPTH_TYPE usStackDepth,
                            void *pvParameters,
                            UBaseType_t uxPriority,
                            TaskHandle_t *pxCreatedTask
                       );
```

https://www.freertos.org/a00125.html

LED Tasks

main.c freertos.c

```
37  /* Private define ------------------------------------------
38  /* USER CODE BEGIN PD */
39
40  /* USER CODE END PD */
41
42  /* Private macro -------------------------------------------
43  /* USER CODE BEGIN PM */
44
45  /* USER CODE END PM */
46
47  /* Private variables ---------------------------------------
48  /* USER CODE BEGIN Variables */
49  TaskHandle_t xTask1, xTask2, xTask3, xTask4, xTask5;
50
51  /* USER CODE END Variables */
52  osThreadId defaultTaskHandle;
53
54  /* Private function prototypes -----------------------------
55  /* USER CODE BEGIN FunctionPrototypes */
56  void LEDTask1(void *);
57  void LEDTask2(void *);
58  void LEDTask3(void *);
59
60  /* USER CODE END FunctionPrototypes */
```

```
/* USER CODE BEGIN RTOS_THREADS */
/* add threads, ... */
xTaskCreate(LEDTask1, "Blink LD1", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY, &xTask1);
xTaskCreate(LEDTask2, "Blink LD2", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY, &xTask2);
xTaskCreate(LEDTask3, "Blink LD3", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY, &xTask3);

/* USER CODE END RTOS_THREADS */
```

LED Tasks

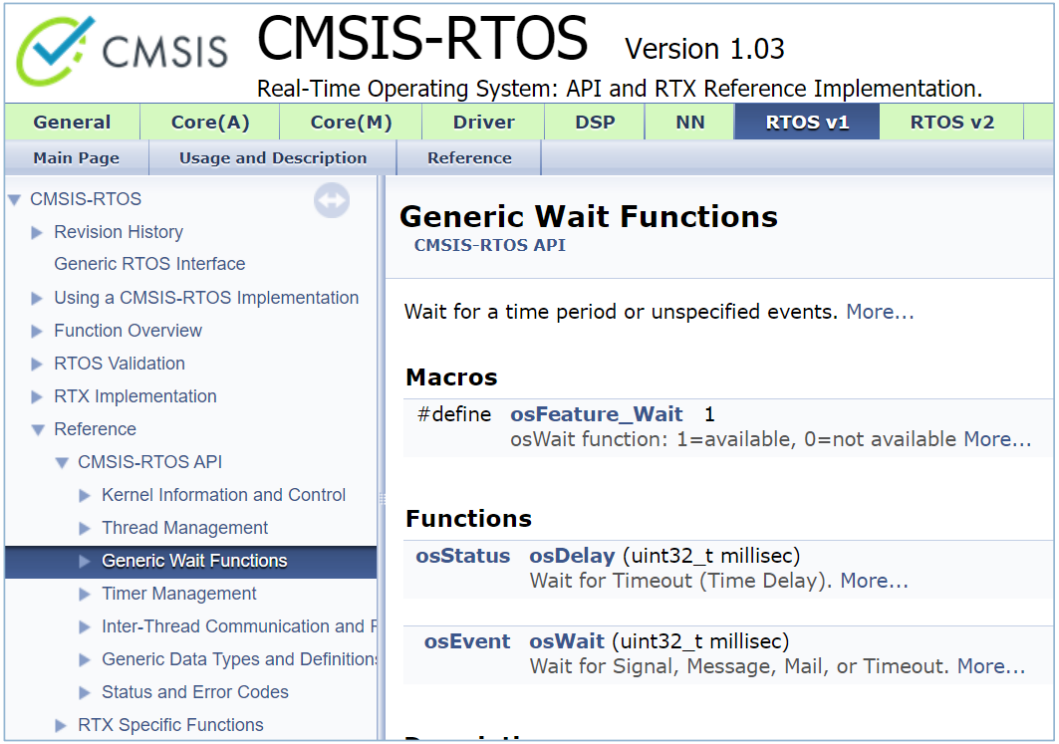
main.c freertos.c

```
139 }
140
141 /* Private application code ------------------------
142 /* USER CODE BEGIN Application */
143 void LEDTask1(void * pvParameters)
144 {
145   for(;;){
146     HAL_GPIO_TogglePin (LD1_GPIO_Port, LD1_Pin);
147     vTaskDelay(200);
148     //osDelay(200);
149   }
150 }
151
152 void LEDTask2(void * pvParameters)
153 {
154   for(;;){
155     HAL_GPIO_TogglePin (LD2_GPIO_Port, LD2_Pin);
156     vTaskDelay(500);
157     //osDelay(500);
158   }
159 }
160
161 void LEDTask3(void * pvParameters)
162 {
163   for(;;){
164     HAL_GPIO_TogglePin (LD3_GPIO_Port, LD3_Pin);
165     vTaskDelay(800);
166     //osDelay(800);
167   }
168 }
169
170
171 /* USER CODE END Application */
172
```

vTaskDelay / osDelay

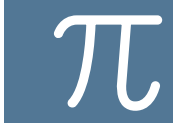
```
void vTaskDelay( const TickType_t xTicksToDelay );
```

https://www.freertos.org/a00127.html



Workshop 1.2

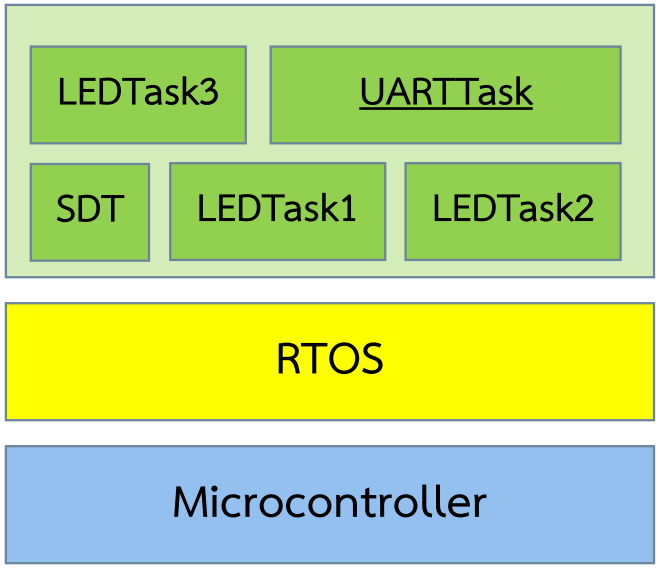
Add A Serial Task



Workshop 1.2 Code

```
/* USER CODE BEGIN RTOS_THREADS */
/* add threads, ... */
xTaskCreate(LEDTask1, "Blink LD1", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY, &xTask1);
xTaskCreate(LEDTask2, "Blink LD2", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY, &xTask2);
xTaskCreate(LEDTask3, "Blink LD3", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY, &xTask3);

HAL_UART_Transmit(&huart3, (uint8_t *) "\n\r\n\r", 4, 100);
xTaskCreate(UARTTask, "UART Task 1", configMINIMAL_STACK_SIZE, (void *) one, tskIDLE_PRIORITY, &xTask4);

/* USER CODE END RTOS_THREADS */
```

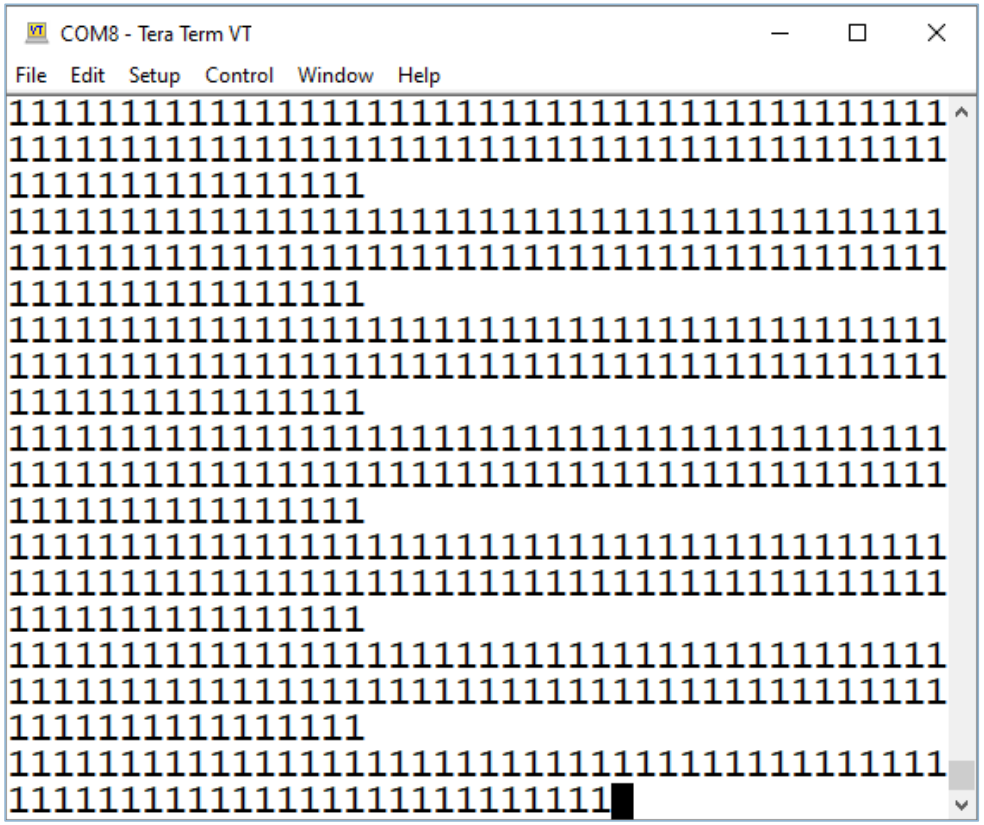
```
177  void UARTTask(void * pvParameters)
178  {
179    char * num = pvParameters;
180
181    for(;;){
182
183      for (uint8_t i=0; i<100; i++)
184        {
185          HAL_UART_Transmit(&huart3, (uint8_t *) num, 1, 50);
186          vTaskDelay(50);
187        }
188
189        HAL_UART_Transmit(&huart3, (uint8_t *) "\n\r", 2, 50);
190        vTaskDelay(500);
191    }
192  }
193
```

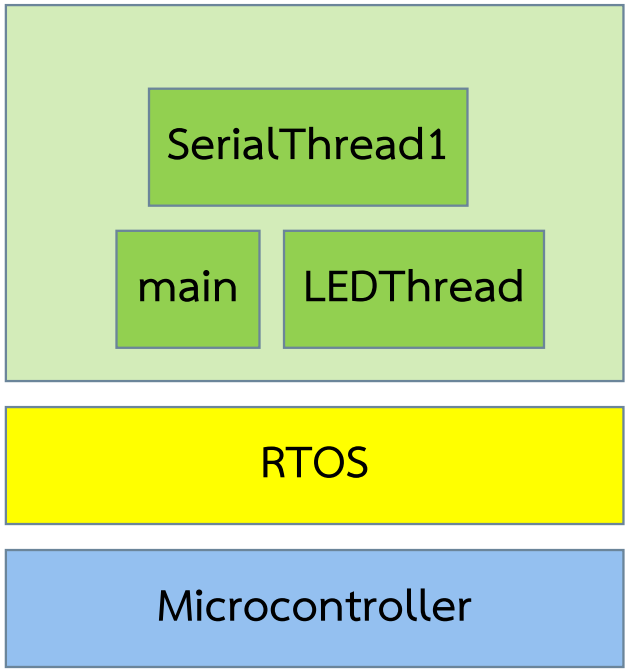


Variables & Function Prototypes

```
47  /* Private variables ---------------------------------
48  /* USER CODE BEGIN Variables */
49  TaskHandle_t xTask1, xTask2, xTask3, xTask4, xTask5;
50
51  char * one = "1";
52  char * two = "2";
53
54  extern UART_HandleTypeDef huart3;
55
56  /* USER CODE END Variables */
57  osThreadId defaultTaskHandle;
58
59  /* Private function prototypes -----------------------
60  /* USER CODE BEGIN FunctionPrototypes */
61  void LEDTask1(void *);
62  void LEDTask2(void *);
63  void LEDTask3(void *);
64  void UARTTask(void *);
65  /* USER CODE END FunctionPrototypes */
66
```

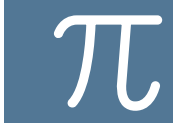
Workshop 1.2 Result





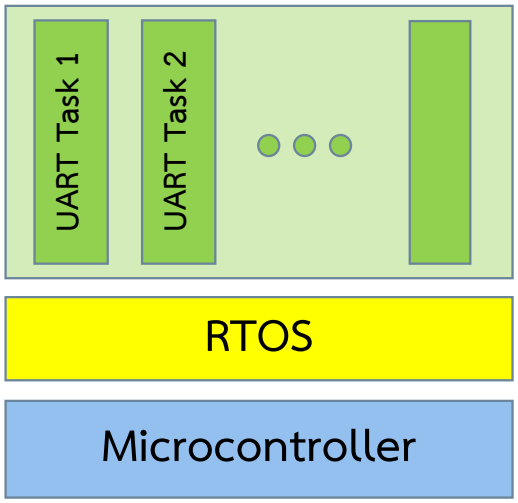
Workshop 1.3

Add Another Serial Task



Workshop 1.3 Code - STM32

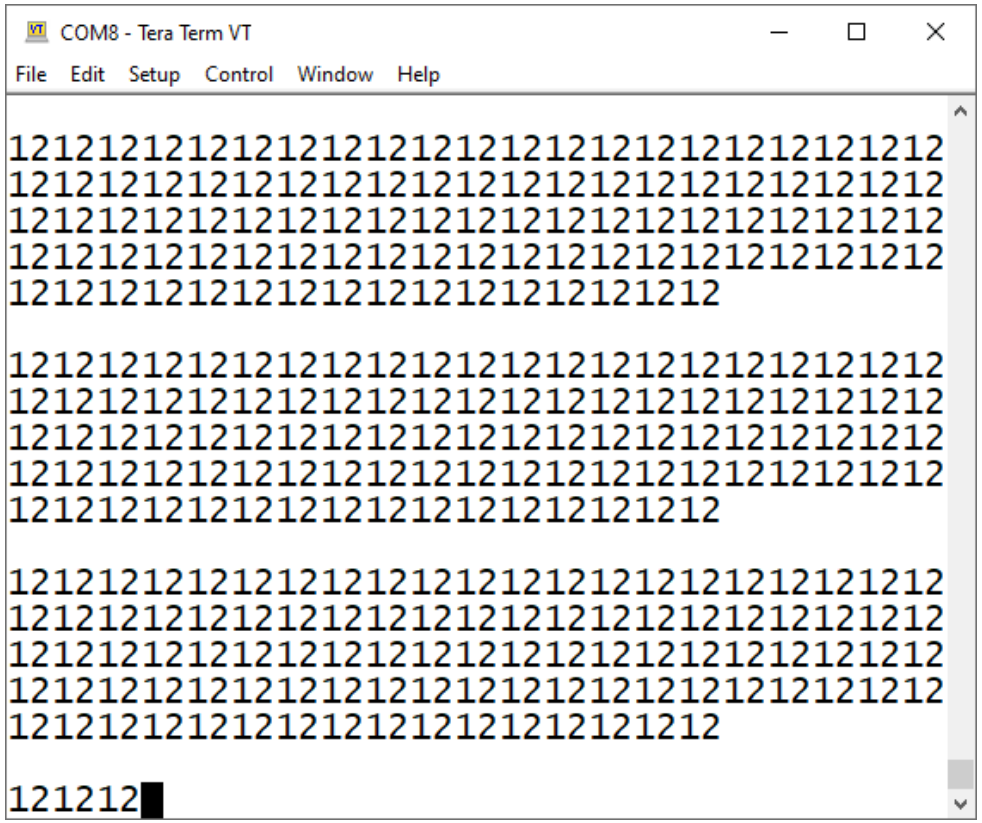
```
/* USER CODE BEGIN RTOS_THREADS */
/* add threads, ... */
xTaskCreate(LEDTask1, "Blink LD1", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY, &xTask1);
xTaskCreate(LEDTask2, "Blink LD2", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY, &xTask2);
xTaskCreate(LEDTask3, "Blink LD3", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY, &xTask3);

HAL_UART_Transmit(&huart3, (uint8_t *) "\n\r\n\r", 4, 100);
xTaskCreate(UARTTask, "UART Task 1", configMINIMAL_STACK_SIZE, (void *) one, tskIDLE_PRIORITY, &xTask4);
xTaskCreate(UARTTask, "UART Task 2", configMINIMAL_STACK_SIZE, (void *) two, tskIDLE_PRIORITY, &xTask5);

/* USER CODE END RTOS_THREADS */
```




```
179  void UARTTask(void * pvParameters)
180  {
181    char * num = pvParameters;
182
183    for(;;){
184
185      for (uint8_t i=0; i<100; i++)
186        {
187          HAL_UART_Transmit(&huart3, (uint8_t *) num, 1, 50);
188          vTaskDelay(50);
189        }
190
191        HAL_UART_Transmit(&huart3, (uint8_t *) "\n\r", 2, 50);
192        vTaskDelay(500);
193    }
194  }
195
```

Critical Region

Workshop 1.3 Result



Race Condition

Mutual Exclusion : Mutex

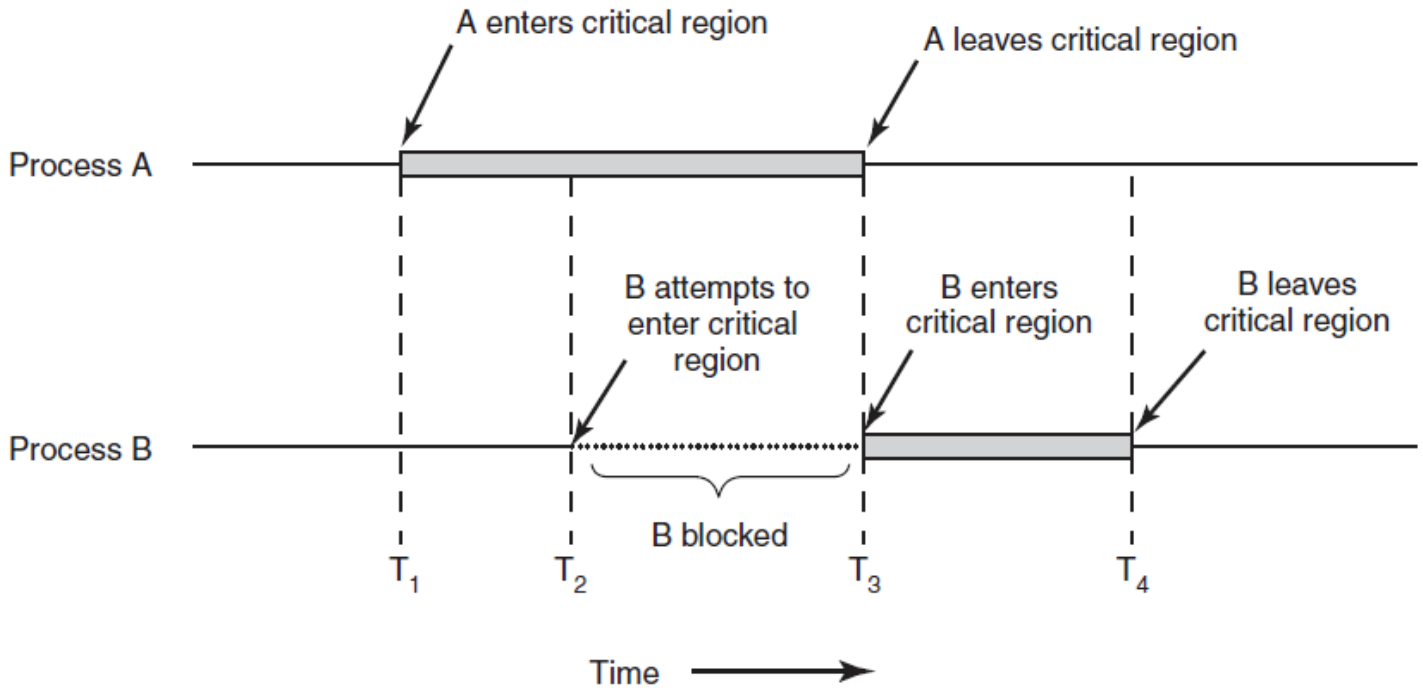


Figure 2-22. Mutual exclusion using critical regions.

Workshop 1.4

Add Mutex

Workshop 1.4 Code – STM32

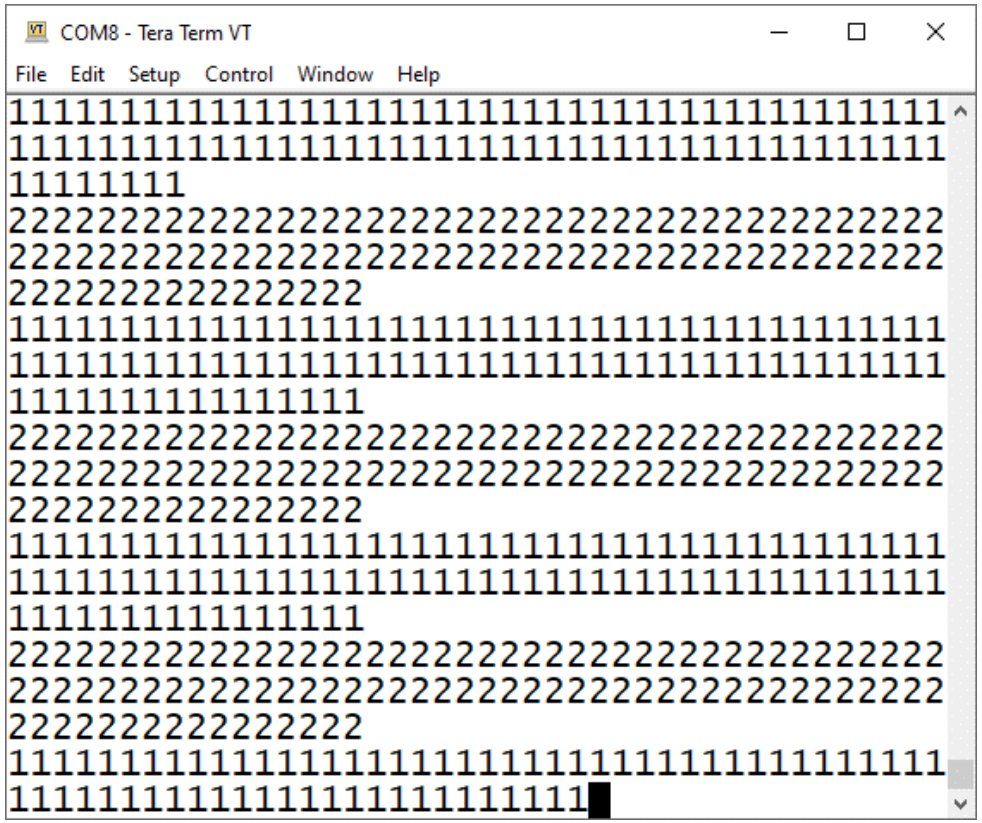
```
47  /* Private variables ---------------------------------------
48  /* USER CODE BEGIN Variables */
49  TaskHandle_t xTask1, xTask2, xTask3, xTask4, xTask5;
50  char * one = "1";
51  char * two = "2";
52
53  SemaphoreHandle_t xSemaphore1 = NULL;
54
55  extern UART_HandleTypeDef huart3;
56  /* USER CODE END Variables */
```

```
118   /* USER CODE BEGIN RTOS_THREADS */
119   /* add threads, ... */
120   xTaskCreate(LEDTask1, "Blink LD1", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY, &xTask1);
121   xTaskCreate(LEDTask2, "Blink LD2", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY, &xTask2);
122   xTaskCreate(LEDTask3, "Blink LD3", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY, &xTask3);
123
124   HAL_UART_Transmit(&huart3, (uint8_t *) "\n\r\n\r", 4, 100);
125   xSemaphore1 = xSemaphoreCreateMutex();
126   xTaskCreate(UARTTask, "UART Task 1", configMINIMAL_STACK_SIZE, (void *) one, tskIDLE_PRIORITY, &xTask4);
127   xTaskCreate(UARTTask, "UART Task 2", configMINIMAL_STACK_SIZE, (void *) two, tskIDLE_PRIORITY, &xTask5);
128   /* USER CODE END RTOS_THREADS */
129
```

Workshop 1.4 Code – STM32

```
179  void UARTTask(void * pvParameters)
180  {
181    char * num = pvParameters;
182
183    for(;;){
184
185      if ( xSemaphore1 != NULL )
186      {
187        if( xSemaphoreTake( xSemaphore1, ( TickType_t ) 1000 ) == pdTRUE )
188        {
189            for (uint8_t i=0; i<100; i++)
190            {
191              HAL_UART_Transmit(&huart3, (uint8_t *) num, 1, 50);
192              vTaskDelay(50);
193            }
194            HAL_UART_Transmit(&huart3, (uint8_t *) "\n\r", 2, 50);
195            xSemaphoreGive(xSemaphore1);
196            taskYIELD();
197        }
198      }
199
200    }
201  }
202
```

Workshop 1.4 Result



The Producer-Consumer Problem

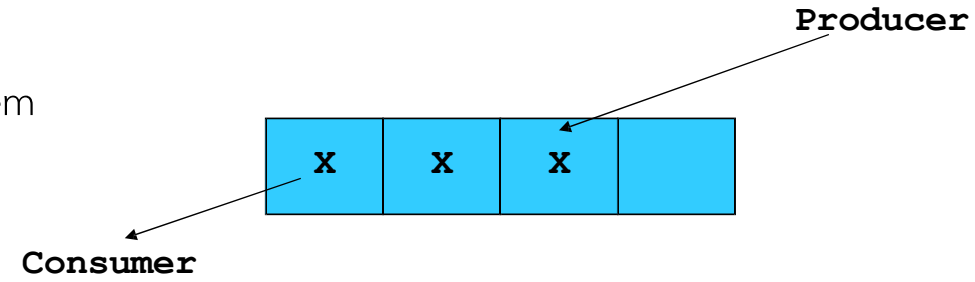
A producer

- produces data items and stores items in a buffer
- sleep when the buffer is full and awake a consumer

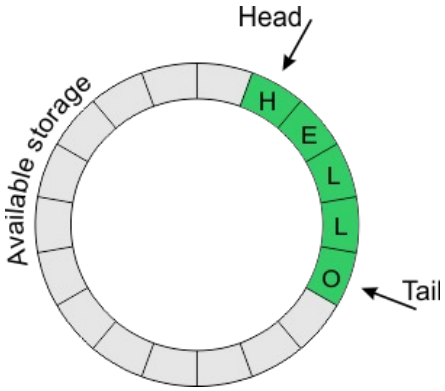
A consumer

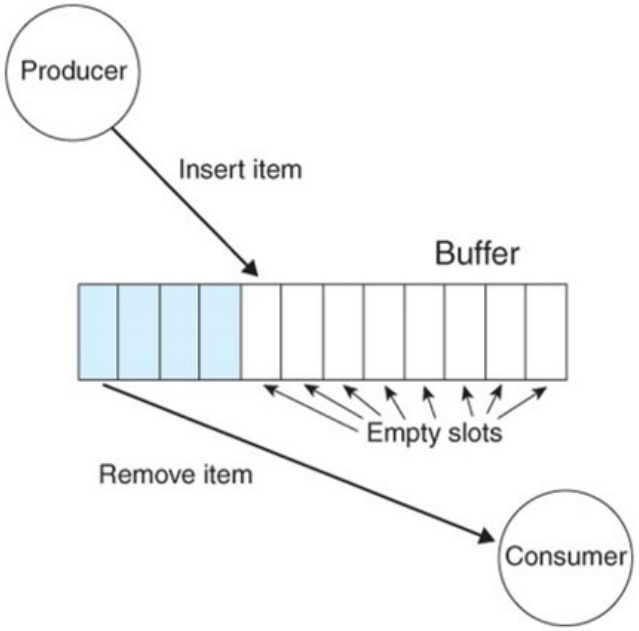
- takes items out of the buffer and consumes items
- sleep when the buffer is empty and awake a producer

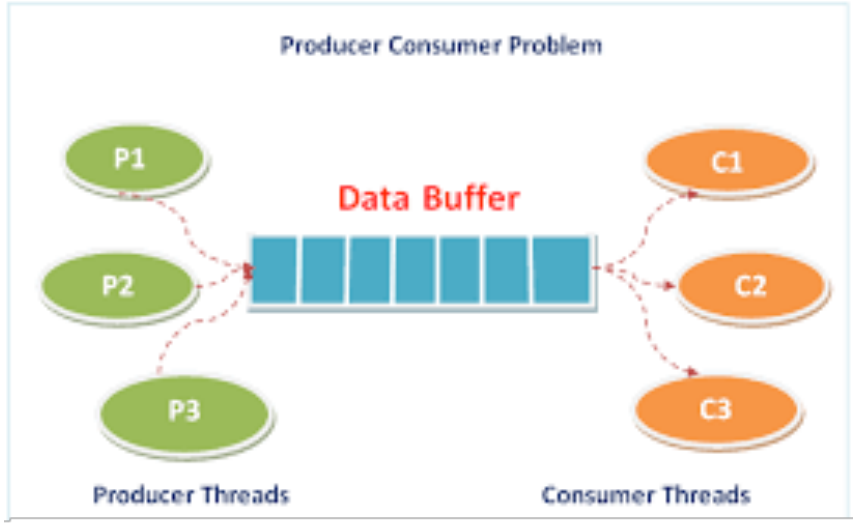
Also called the **bounded buffer** problem



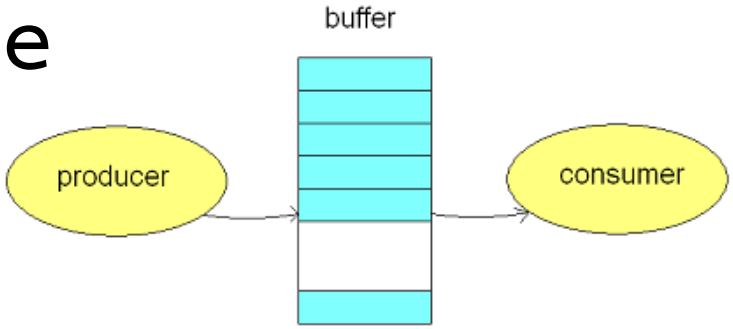
PCP : Single & Multiple Tasks







PCP Solution : Pseudocode

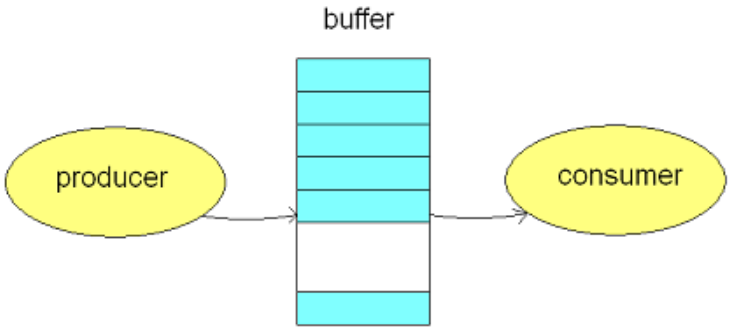



```
int count = 0;
#define N 4 /* buf size */
prod() {
   while(TRUE) {
        item = produce()
        if (count == N)
               sleep();
        insert_item();
        count++;
        if (count == 1)
               wakeup(con);
   }
}
```

Critical Region

```
con() {
   while(TRUE) {
        if (count == 0)
               sleep();
        remove_item();
        count--;
        if (count == N-1)
               wakeup(prod);
   }
}
```

PCP Solution : Mutex




```
int count = 0;
#define N 4 /* buf size */
prod() {
   while(TRUE) {
        item = produce()
        if (count == N)
               sleep();
        acquire_lock()
        insert_item();
        count++;
        release_lock()
        if (count == 1)
               wakeup(con);
   }
}
```

```
con() {
   while(TRUE) {
        if (count == 0)
               sleep();
        acquire_lock()
        remove_item();
        count--;
        release_lock();
        if (count == N-1)
               wakeup(prod);
   }
}
```

Problematic execution sequence

```
prod() {
    while(TRUE) {
        item = produce()
        if (count == N)
            sleep();
        acquire_lock()
        insert_item();
        count++;
        release_lock()
        if (count == 1)
            wakeup(con);
```

```
con() {
    while(TRUE) {
        if (count == 0)
            sleep();
        acquire_lock()
        remove_item();
        count--;
        release_lock();
        if (count == N-1)
            wakeup(prod);
    }
}
```

wakeup without a matching sleep is lost

Unmatching sleep/wakeup signals

Semaphore

- Dijkstra (1965) introduced two primitives that are more powerful than simple sleep and wakeup alone.
  - P(): *proberen,* from Dutch *to test.*
  - V(): *verhogen,* from Dutch *to increment.*
  - Also called *wait* & *signal, down* & *up.*

Semaphore Operations

wait(*S*):

```
S.count--;
if (S.count < 0) {
        add this process to S.L;
        sleep;
}
```

signal(*S*):

```
S.count++;
if (S.count <= 0) {
        remove a process P from S.L;
        wakeup(P);
}
```

PCP : Semaphore

```
#define N = 4

semaphore mutex = 1;

/* count empty slots */
semaphore empty = N;

/* count full slots */
semaphore full = 0;
```

```
prod() {
   while(TRUE) {
        item = produce()
        wait(empty);
        wait(mutex)
        insert_item();
        signal(mutex);
        signal(full);
   }
}
```

```
con() {
   while(TRUE) {
        wait(full);
        wait(mutex);
        remove_item();
        signal(mutex);
        signal(empty);
   }
}
```

FreeRTOS - Semaphore

Semaphores

[API]

TIP: 'Task Notifications' can provide a light weight alternative to semaphores in many situations

Modules

- xSemaphoreCreateBinary
- xSemaphoreCreateBinaryStatic
- vSemaphoreCreateBinary [use xSemaphoreCreateBinary() for new designs]
- xSemaphoreCreateCounting
- xSemaphoreCreateCountingStatic
- xSemaphoreCreateMutex
- xSemaphoreCreateMutexStatic
- xSemaphoreCreateRecursiveMutex
- xSemaphoreCreateRecursiveMutexStatic
- vSemaphoreDelete
- xSemaphoreGetMutexHolder
- xSemaphoreTake
- xSemaphoreTakeFromISR
- xSemaphoreTakeRecursive
- xSemaphoreGive
- xSemaphoreGiveRecursive
- xSemaphoreGiveFromISR
- uxSemaphoreGetCount

FreeRTOS - Queue

Queue Management

[API]

Modules

- xQueueCreate
- xQueueCreateStatic
- vQueueDelete
- xQueueSend
- xQueueSendFromISR
- xQueueSendToBack
- xQueueSendToBackFromISR
- xQueueSendToFront
- xQueueSendToFrontFromISR
- xQueueReceive
- xQueueReceiveFromISR
- uxQueueMessagesWaiting
- uxQueueMessagesWaitingFromISR
- uxQueueSpacesAvailable
- xQueueReset
- xQueuePeek
- xQueuePeekFromISR
- vQueueAddToRegistry
- pcQueueGetName
- vQueueUnregisterQueue
- xQueueIsQueueEmptyFromISR
- xQueueIsQueueFullFromISR
- xQueueOverwrite
- xQueueOverwriteFromISR