Real Time Operating Systems (RTOS)

Spring 2022

Scheduling : Ex #01 – #04

Week #3 Lesson Plan (*January 1st – February 3rd*)

Monday – Class

- Scheduling

Wednesday – Class

- Overview of Ex-Set #2 - Scheduling Ex #1 – Ex #4

Thursday / Friday – Lab

- Ex-Set #2 - Scheduling Ex #1 – Ex #4

Recorded

On Line

Next Week….

- Monday: Reading
  - Book #1 : Chapter #3 (3.1 – 3.7)
- Tuesday: Lab “Report”
  - Ex-Set #2 Report : myCourses Quiz – Due 2/7 @ 11:59 PM
- Wednesday: Reading
  - Book #2 : Chapter #3 (3.1 – 3.8)

Lab Overview…

- **Exercise #01 – Create and Run a Single Task That Executes Repeatedly**
- **Exercise #02 – Implement a Single Periodic Task**
- **Exercise #03 – Create and Run Multiple Independent Periodic Tasks**
- **Exercise #04 – Evaluate Priority Preemptive Scheduling Policies**

- NOTE : This presentation is only an overview of each of the exercises and is intended to clarify any questions that might arise. Please complete the exercises by following the detailed instructions in the textbook.
- Please pay particular attention to the ***Exercise Review*** section at the end of each section. These reviews are excellent at summarizing the exercise's learning objectives.

In this exercise you will learn how to develop one of the simplest, essential functions of a multitasking system, the execution of a single, continuously executing, task.

EXERCISE #01 – CREATE AND RUN A SINGLE TASK THAT EXECUTES REPEATEDLY

Exercise #01 – Create and Run a Single Task That Executes Repeatedly

Initial Setup for ALL RTOS Exercises…

- Middleware → FREERTOS
  - **CMSSIS_V1**
- System Core → SYS
  - Timebase Source **TIM1**
- Connectivity → USB_OTG_FS
  - **Disable**
- Middleware → USB_HOST
  - **Disable**

Exercise #01 – Create and Run a Single Task That Executes Repeatedly

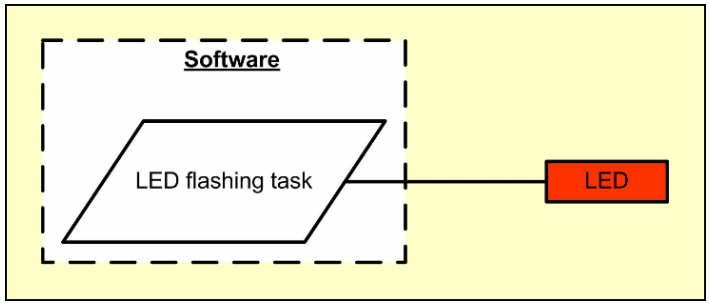
```
/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
I2C_HandleTypeDef hi2c1;

I2S_HandleTypeDef hi2s2;
I2S_HandleTypeDef hi2s3;

SPI_HandleTypeDef hspi1;

osThreadId defaultTaskHandle;
/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);
static void MX_I2S2_Init(void);
static void MX_I2S3_Init(void);
static void MX_SPI1_Init(void);
void StartDefaultTask(void const * argument);

/* USER CODE BEGIN PFP */
```

```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */
```

```
  /* USER CODE BEGIN RTOS_TIMERS */
  /* start timers, add new ones, ... */
  /* USER CODE END RTOS_TIMERS */

  /* USER CODE BEGIN RTOS_QUEUES */
  /* add queues, ... */
  /* USER CODE END RTOS_QUEUES */

  /* Create the thread(s) */
  /* definition and creation of defaultTask */
  osThreadDef(defaultTask, StartDefaultTask, osPriorityNormal, 0, 128);
  defaultTaskHandle = osThreadCreate(osThread(defaultTask), NULL);

  /* USER CODE BEGIN RTOS_THREADS */
  /* add threads, ... */
  /* USER CODE END RTOS_THREADS */

  /* Start scheduler */
  osKernelStart();

  /* We should never get here as control is now taken by the scheduler */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

```
/* USER CODE END Header_StartDefaultTask */
void StartDefaultTask(void const * argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {

  }
  /* USER CODE END 5 */
}
```

Exercise #01 – Create and Run a Single Task That Executes Repeatedly



LED Flashing Task

Loop Forever:
*Turn the **RED** LED on*
2000 mSec Delay
*Turn the **RED** LED off*
500 mSec Delay
End loop.

a) Software Delays

```
/****** Part A ******/
// Turn RED LED On
HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_SET);
// Delay ~2000 mSec
for (int i=0; i < 2000000; i++);
// Turn RED LED Off
HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
// Delay ~500 mSec
for (int i=0; i < 500000; i++);
/****** End A *******/
```

b) osDelay()

Note : 1000 ≈ 1 mSec

c) vTaskDelay()

d) Software Delays with vTaskDelay()

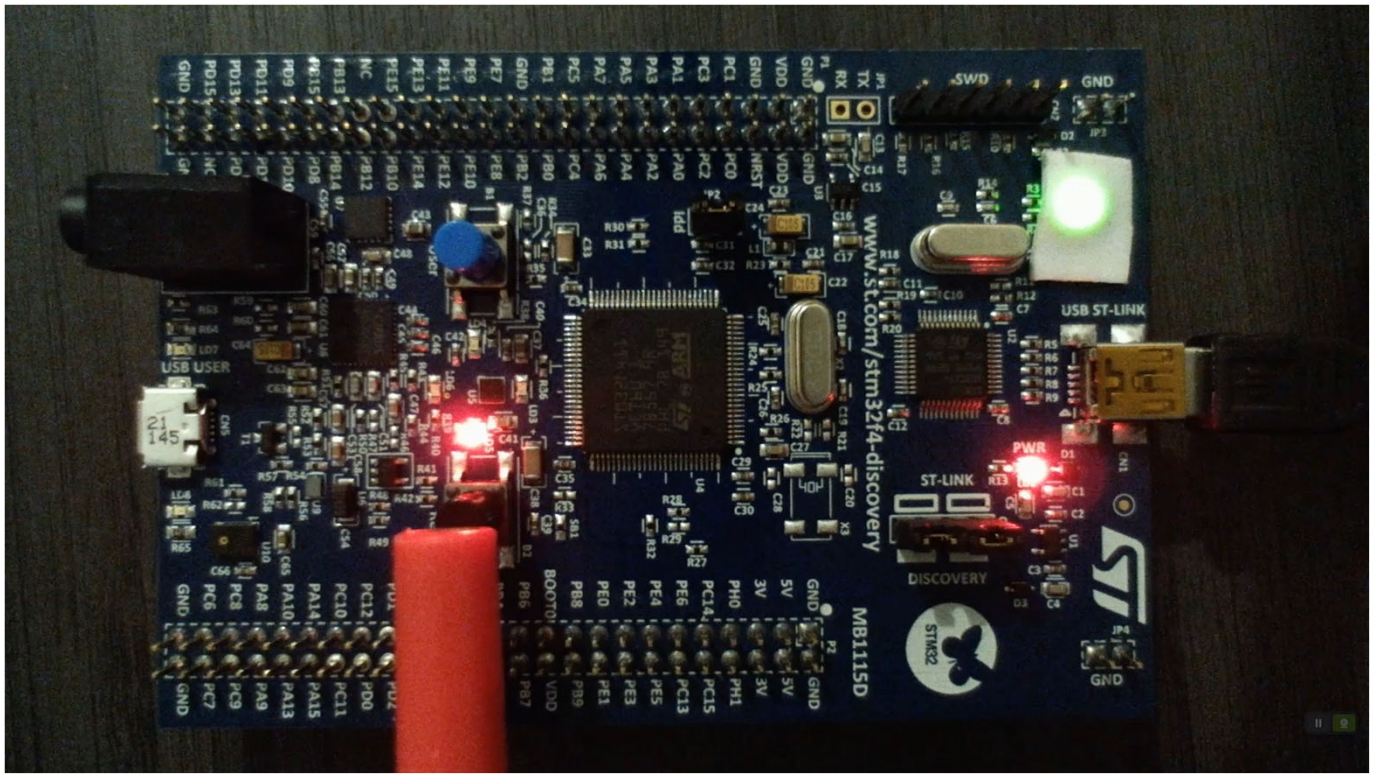
e) Turn ALL LED's on, then Turn ALL LED's off

Exercise #01 – Create and Run a Single Task That Executes Repeatedly

- Software Delays:
  - Timing is not exact.
  - Only counts when task is running. In a multi-task setting, when the task is suspended… the count is suspended.
  - Effectively, software delays are USELESS!
- osDelay() / vTaskDelay():
  - Effectively identical. osDealy() is not RTOS specific were as vTaskDelay() is specific to freeRTOS
  - These delay functions set a fixed delay, which began at the moment they are called.
  - vTaskDelay() causes a task to block for the specified number of ticks from the time vTaskDelay() is called.
  - It is therefore difficult to use vTaskDelay() by itself to generate a fixed execution frequency as the time between a task unblocking following a call to vTaskDelay() and that task next calling vTaskDelay() may not be fixed.
  - The task may take a different path through the code between calls, or may get interrupted or preempted a different number of times each time it executes.
- There's a better way…. Exercise #02

Exercise #01 – Create and Run a Single Task That Executes Repeatedly

Results…



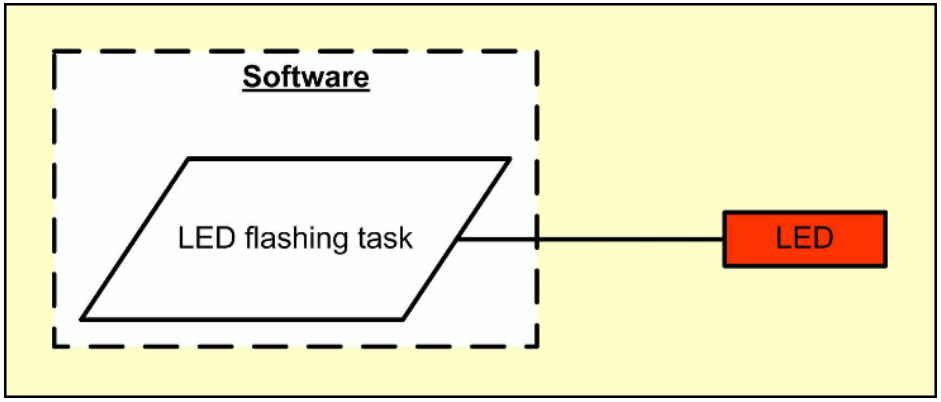
Fundamental purpose of the exercise: to create and run a single periodic task having an accurate periodic time.

EXERCISE #02 – IMPLEMENT A SINGLE PERIODIC TASK

Exercise #02 – Implement a Single Periodic Task

- vTaskDelayUntil() can be used by periodic tasks to ensure a constant execution frequency.
- vTaskDelayUntil() specifies the absolute (exact) time at which it wishes to unblock.
- Whereas vTaskDelay() specifies a wake time relative to the time at which the function is called,
- osDelayUntil() & vTaskDelayUntil() are effectively identical.
  - osDelayUntil() is not RTOS specific
  - vTaskDelayUntil() is specific to freeRTOS

Exercise #02 – Implement a Single Periodic Task



LED Flashing Task

Loop Forever:
*Turn the **RED** LED on*
1000 mSec Software Delay
2000 mSec Delay-Until
*Turn the **RED** LED off*
2000 mSec Delay-Until
End loop.

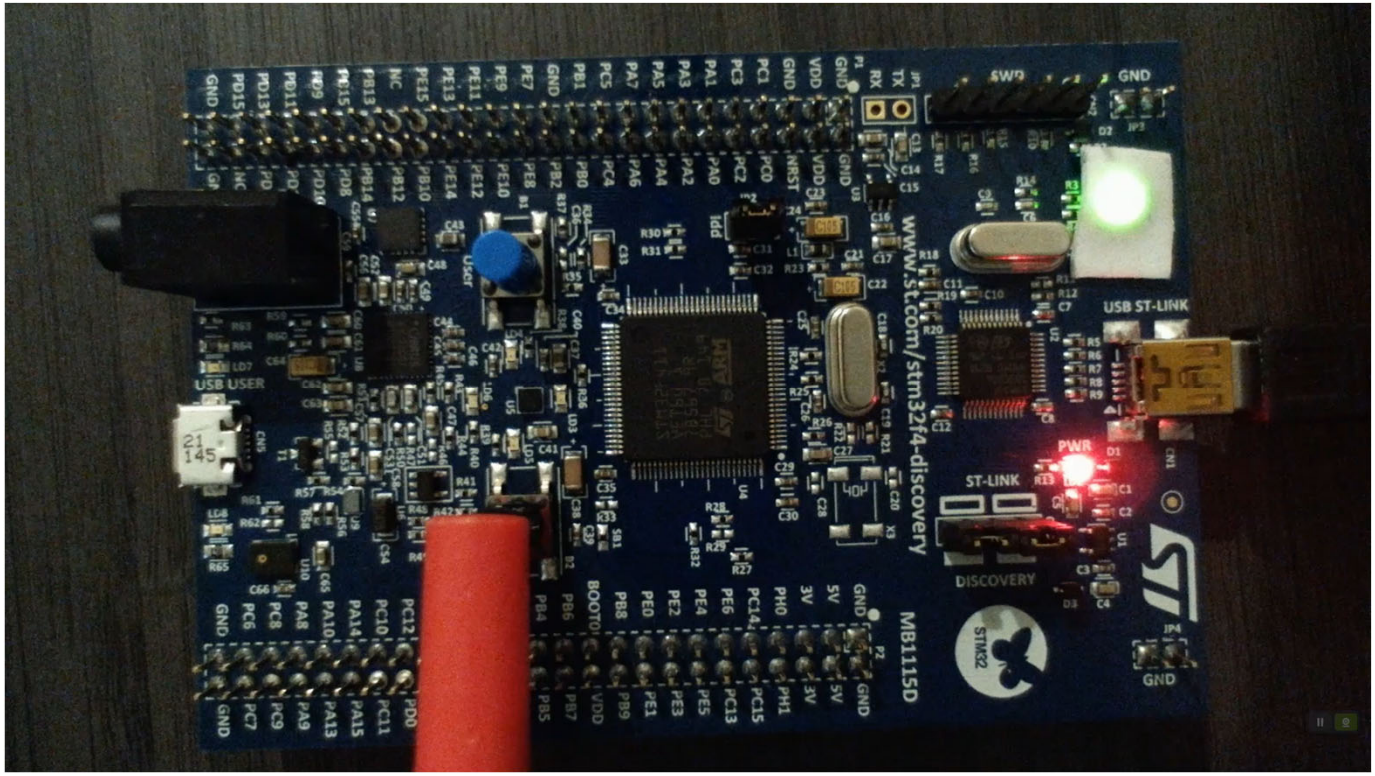
a) 1000 mSec Delays (the books says 2000)

```
void StartDefaultTask(void const * argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  TickType_t TaskTimeStamp;
  TickType_t DelayTimeMsec = 2000;
  TaskTimeStamp = xTaskGetTickCount();
  /* Infinite loop */
  for(;;)
   {
      // Turn RED LED On
      HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_SET);

      // Delay ~1000 mSec
      for (int i=0; i < 1000000; i++);

      // Delay 2000 mSec
      osDelayUntil(&TaskTimeStamp, DelayTimeMsec);

      // Turn RED LED Off
      HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);

      // Delay 2000 mSec
      osDelayUntil(&TaskTimeStamp, DelayTimeMsec);
   }
  /* USER CODE END 5 */
}
```

b) 500 mSec Delay

Exercise #02 – Implement a Single Periodic Task

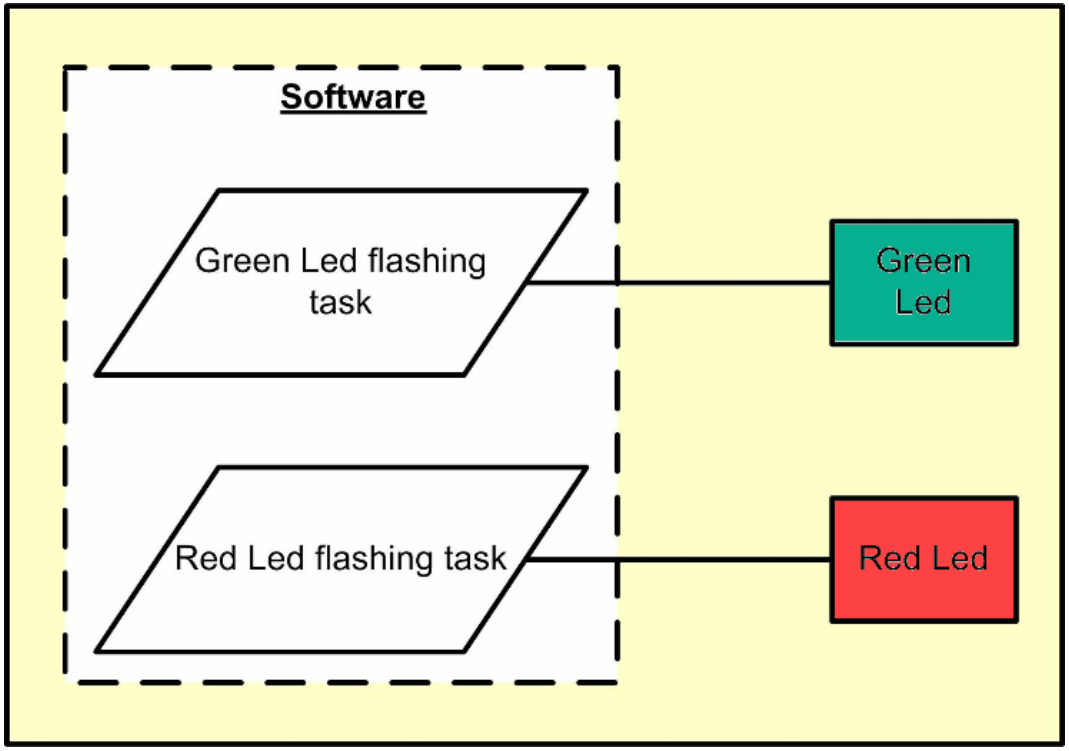
Results…



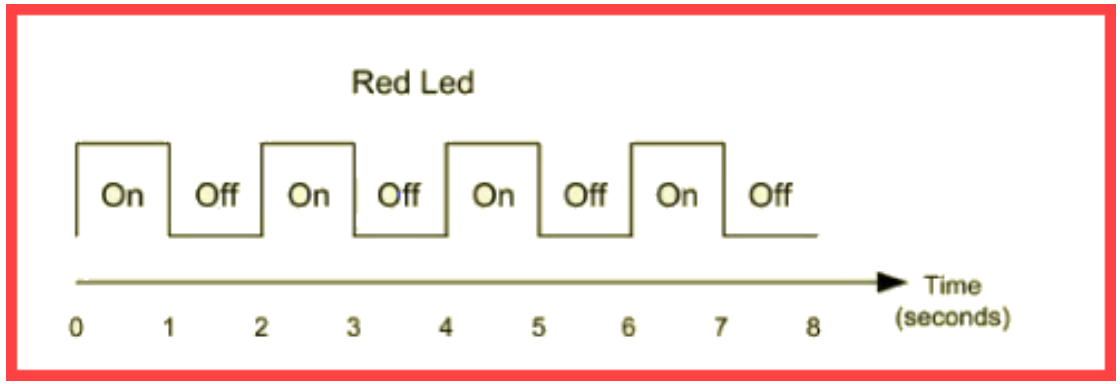
Fundamental purpose of the exercise is to create and run multiple independent periodic tasks.

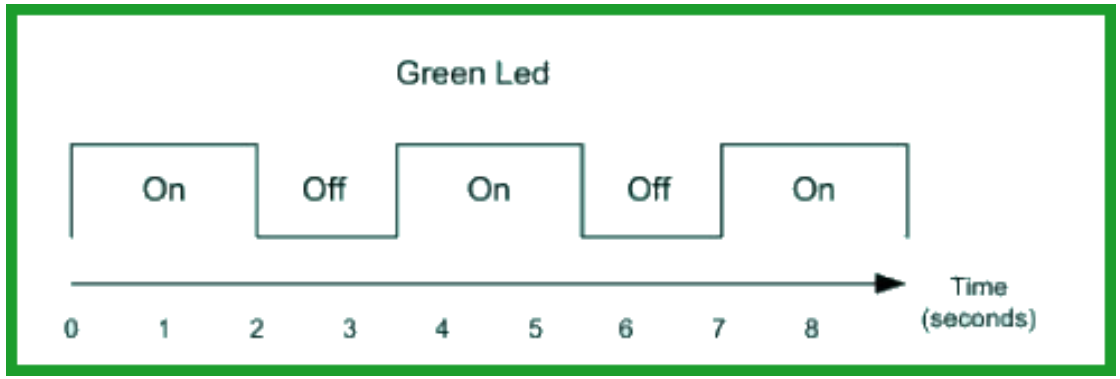
EXERCISE #03 – CREATE AND RUN MULTIPLE INDEPENDENT PERIODIC TASKS

Exercise #03 – Create and Run Multiple Independent Periodic Tasks



Exercise #03 – Create and Run Multiple Independent Periodic Tasks





Exercise #03 – Create and Run Multiple Independent Periodic Tasks

```
34
35 /* Private define ------------------------
36 /* USER CODE BEGIN PD */
37 #define GREEN_LED   GPIO_PIN_12
38 #define ORANGE_LED  GPIO_PIN_13
39 #define RED_LED     GPIO_PIN_14
40 #define BLUE_LED    GPIO_PIN_15
41 /* USER CODE END PD */
42
```

```
68
69 void Start_RED_LED(void const * argument);
70 void Start_GREEN_LED(void const * argument);
71
```

```
134
135   /* definition and creation of RED_LED */
136   osThreadDef(RED_LED, Start_RED_LED, osPriorityNormal, 0, 128);
137   RED_LEDHandle = osThreadCreate(osThread(RED_LED), NULL);
138
139   /* definition and creation of GREEN_LED */
140   osThreadDef(GREEN_LED, Start_GREEN_LED, osPriorityNormal, 0, 128);
141   GREEN_LEDHandle = osThreadCreate(osThread(GREEN_LED), NULL);
142
```

Exercise #03 – Create and Run Multiple Independent Periodic Tasks

```
455
456 /* USER CODE END Header_Start_RED_LED */
457 void Start_RED_LED(void const * argument)
458 {
459   /* USER CODE BEGIN StartFlashRed */
460
461   /* Infinite loop */
462   for(;;)
463   {
464
465   }
466   /* USER CODE END StartFlashRed */
467 }
468
```

```
479
480 /* USER CODE END Header_Start_GREEN_LED */
481 void Start_GREEN_LED(void const * argument)
482 {
483   /* USER CODE BEGIN Start_GREEN_LED */
484
485     // Toggle RED LED
486     HAL_GPIO_TogglePin(GPIOD, RED_LED);
487
488
489   }
490   /* USER CODE END Start_GREEN_LED */
491 }
492
```

RED LED Task

Initialize osDelayUntil() Function
Loop Forever:
- *Toggle **RED** LED*
- *1 Sec Delay with osDelayUntil()*
- *Toggle **RED** LED*
- *1 Sec Delay with osDelayUntil()*

End loop.

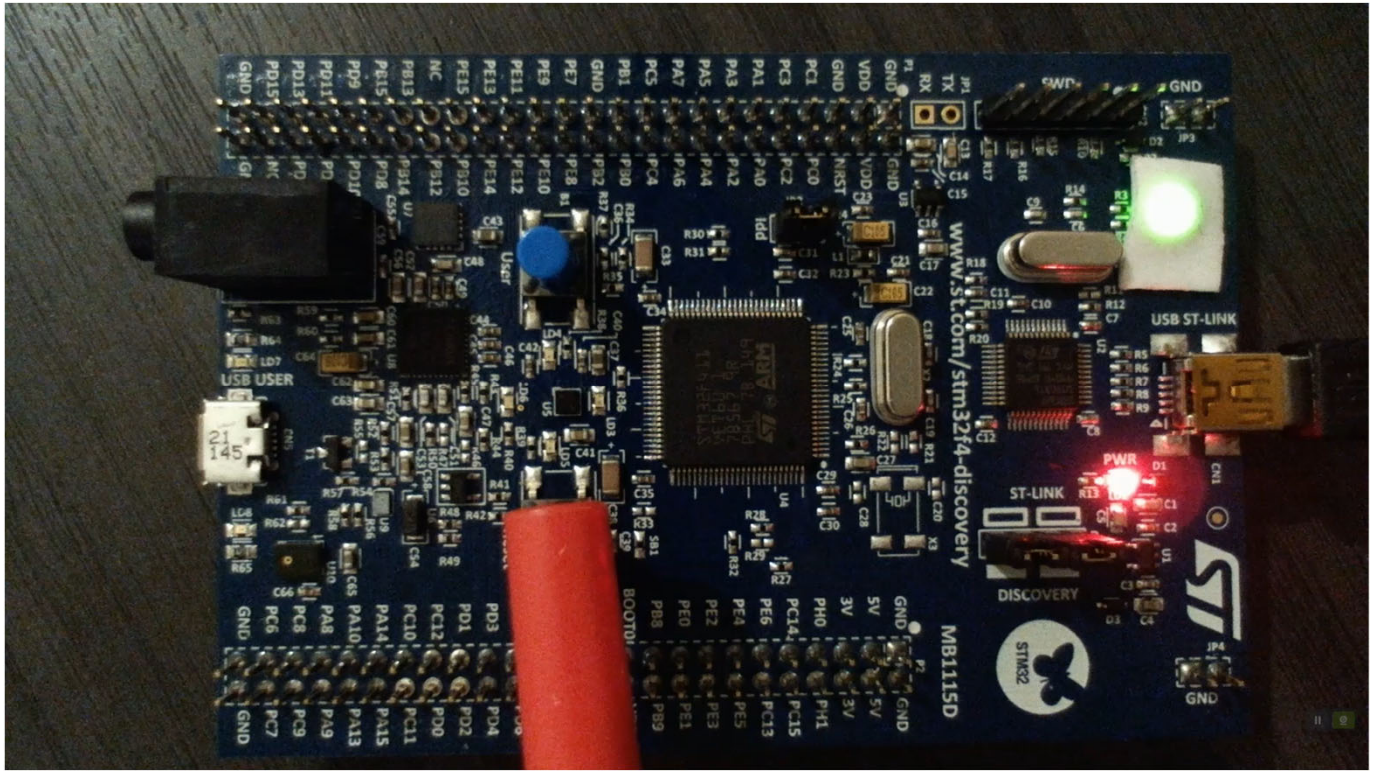
GREEN LED Task

Initialize osDelayUntil() Function
Loop Forever:
- *Toggle **GREEN** LED*
- *2 Sec Delay with osDelayUntil()*
- *Toggle **GREEN** LED*
- *1.5 Sec Delay with osDelayUntil()*

End loop.

Exercise #03 – Create and Run Multiple Independent Periodic Tasks

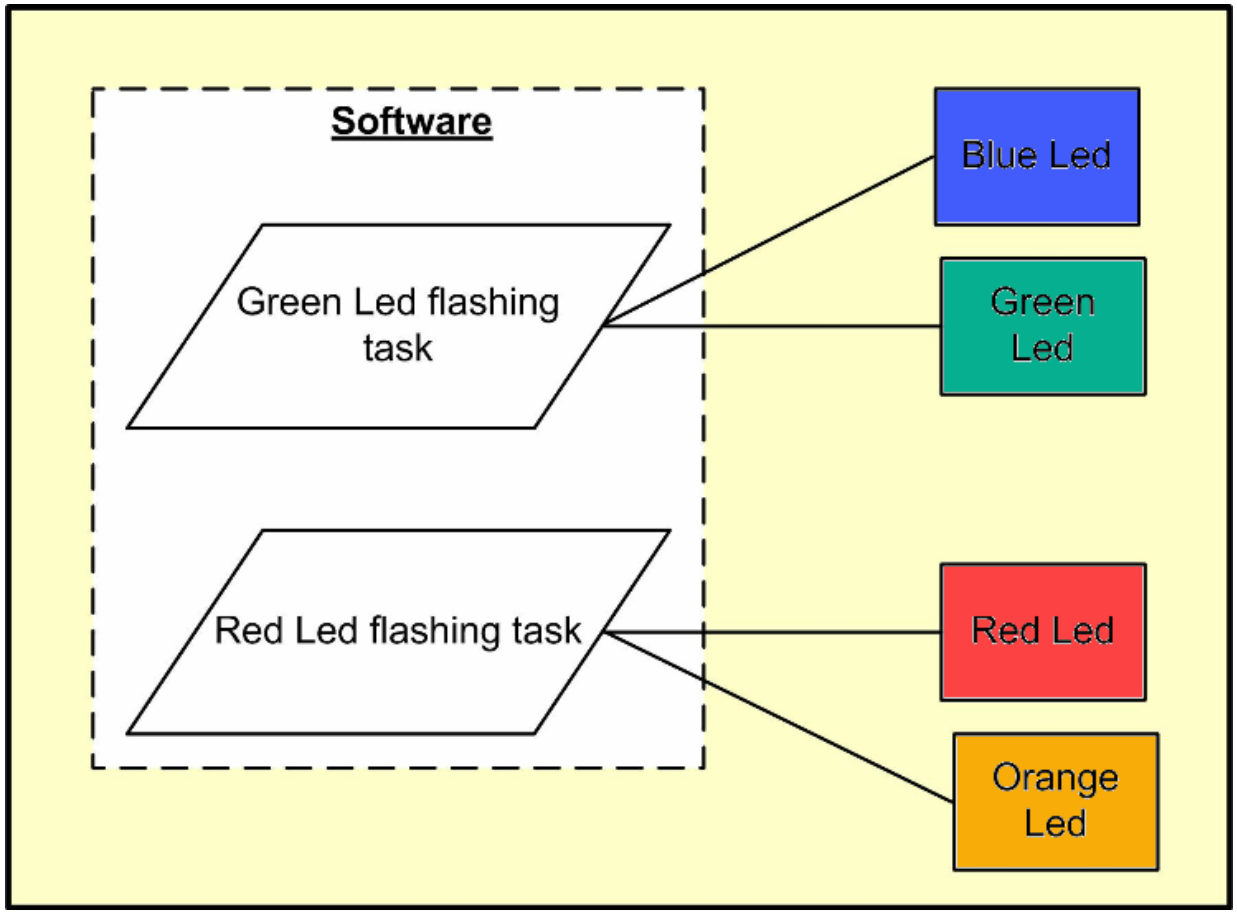
Results…



Fundamental purpose of the exercise is to gain a good understanding of task behavior where a priority preemptive scheduling policy is used.

EXERCISE #04 – EVALUATE PRIORITY PREEMPTIVE SCHEDULING POLICIES

Exercise #04 – Evaluate Priority Preemptive Scheduling Policies



Exercise #04 – Evaluate Priority Preemptive Scheduling Policies

BLUE / GREEN LED Task

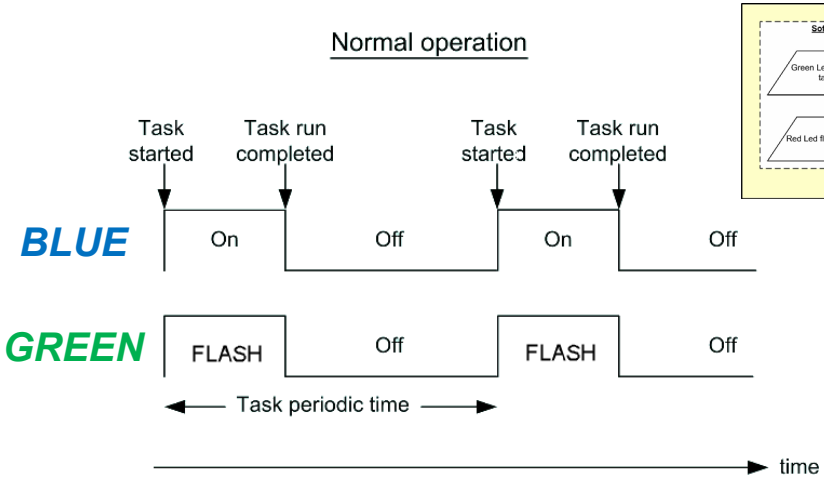
Loop Forever:
- *Turn BLUE LED On*
- *Toggle GREEN LED for 4.0 Seconds @ ~ 20 Hz*
- *Turn GREEN LED Off*
- *Turn BLUE LED Off*
- *Suspend for 6.0 sec Delay with osDelay()*

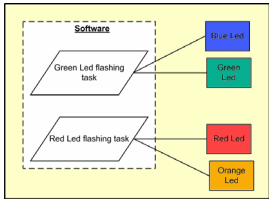
End loop.

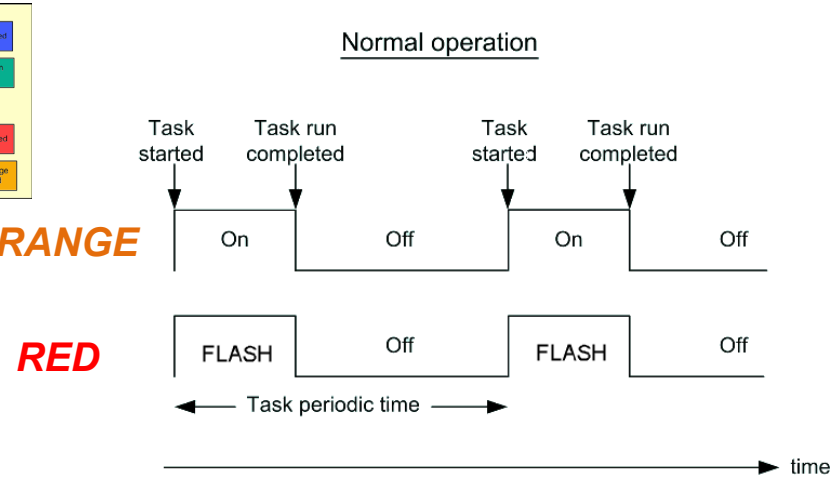
ORANGE / RED LED Task

Loop Forever:
- *Turn ORANGE LED On*
- *Toggle RED LED for 0.5 Seconds @ ~ 20 Hz*
- *Turn RED LED Off*
- *Turn ORANGE LED Off*
- *Suspend for 1.5 sec Delay with osDelay()*

End loop.







Exercise #04 – Evaluate Priority Preemptive Scheduling Policies

Sample Code for BLUE / GREEN Task

```
482
483 /* USER CODE END Header_Start_Blue_Green */
484 void Start_Blue_Green(void const * argument)
485 {
486   /* USER CODE BEGIN Start_Blue_Green */
487   /* Infinite loop */
488   for(;;)
489   {
490       HAL_GPIO_WritePin(GPIOD, BLUE_LED, GPIO_PIN_SET);
491
492       for (int i=0; i<=160; i++)    // 4.0 Sec -> i = 4.0 / 25 msec = 160
493       {
494           HAL_GPIO_TogglePin(GPIOD, GREEN_LED);
495           osDelay(25);              // F=20 HZ -> T= 50 mSec -> T/2 = 25 mSec
496       }
497
498       HAL_GPIO_WritePin(GPIOD, BLUE_LED, GPIO_PIN_RESET);
499       HAL_GPIO_WritePin(GPIOD, GREEN_LED, GPIO_PIN_RESET);
500       osDelay(6000);
501   }
502   /* USER CODE END Start_Blue_Green */
503 }
504
```

Exercise #04 – Evaluate Priority Preemptive Scheduling Policies

1) Test **BLUE** / **GREEN** Task Alone – Normal Task Priority
2) Test **ORANGE** / **RED** Task Alone – Normal Task Priority
3) Test **BLUE** / **GREEN** Task & **ORANGE** / **RED** Task Together, Both with Normal Task Priority
4) Test **BLUE** / **GREEN** Task & **ORANGE** / **RED** Task Together
   - **BLUE** / **GREEN** Task with Normal Task Priority
   - **ORANGE** / **RED** Task with Above-Normal Task Priority
5) Test **BLUE** / **GREEN** Task & **ORANGE** / **RED** Task Together
   - **BLUE** / **GREEN** Task with Above-Normal Task Priority
   - **ORANGE** / **RED** Task with Normal Task Priority

Exercise #04 – Evaluate Priority Preemptive Scheduling Policies

```
135
136   /* definition and creation of Orange_Red */
137   osThreadDef(Orange_Red, Start_Orange_Red, osPriorityAboveNormal, 0, 128);
138 //  osThreadDef(Orange_Red, Start_Orange_Red, osPriorityNormal, 0, 128);
139   Orange_RedHandle = osThreadCreate(osThread(Orange_Red), NULL);
140
141   /* definition and creation of Blue_Green */
142 //  osThreadDef(Blue_Green, Start_Blue_Green, osPriorityAboveNormal, 0, 128);
143   osThreadDef(Blue_Green, Start_Blue_Green, osPriorityNormal, 0, 128);
144   Blue_GreenHandle = osThreadCreate(osThread(Blue_Green), NULL);
145
```