Nathan Bunnell
Embedded Real-Time Operating Systems
ECE-40290
Student ID: U08895857

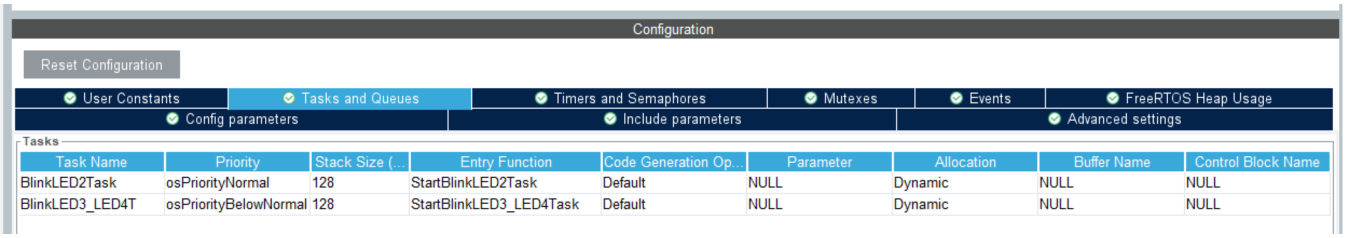
Date: 10/25/2021

Assignment 4: Timers

The following will document completion of the fourth assignment for ECE-40290, with the stated goals of:

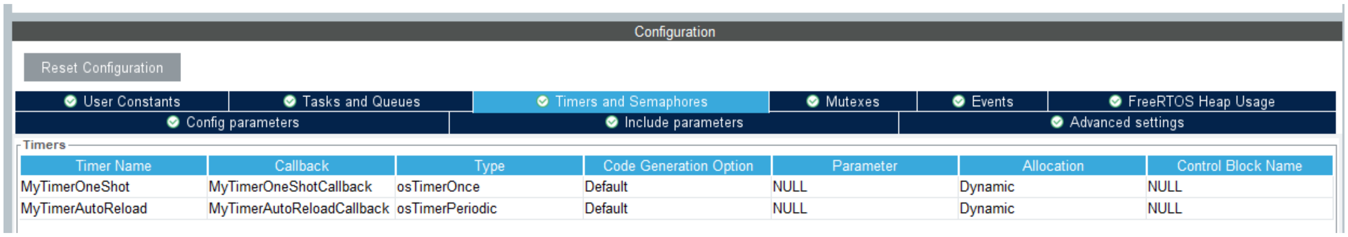
- **Create "Task 1" that blinks the LED2 at a rate of 1 second**
- **Create "Task 2" that blinks the "Wifi/BLE" LED at a rate of 2 seconds**
- **Create "Timer 1" that is a One-Shot timer function named prvMyTimerOneShot(). The timer should fire 15 seconds after startup and display a message "prvMyTimerOneShot" on the console.**
- **Create "Timer 2" that is an auto-reload timer function named prvMyTimerAutoReload(). The timer should fire every 5 seconds and display a count and a message on the console, for example, "prvMyTimerAutoReload: 1".**

As before, generate a default FreeRTOS-based project file and open the configuration. We'll define tasks and timers as seen below:



Tasks

Task Name	Priority	Stack Size (...	Entry Function	Code Generation Op...	Parameter	Allocation	Buffer Name	Control Block Name
BlinkLED2Task	osPriorityNormal	128	StartBlinkLED2Task	Default	NULL	Dynamic	NULL	NULL
BlinkLED3_LED4T	osPriorityBelowNormal	128	StartBlinkLED3_LED4Task	Default	NULL	Dynamic	NULL	NULL



Timers

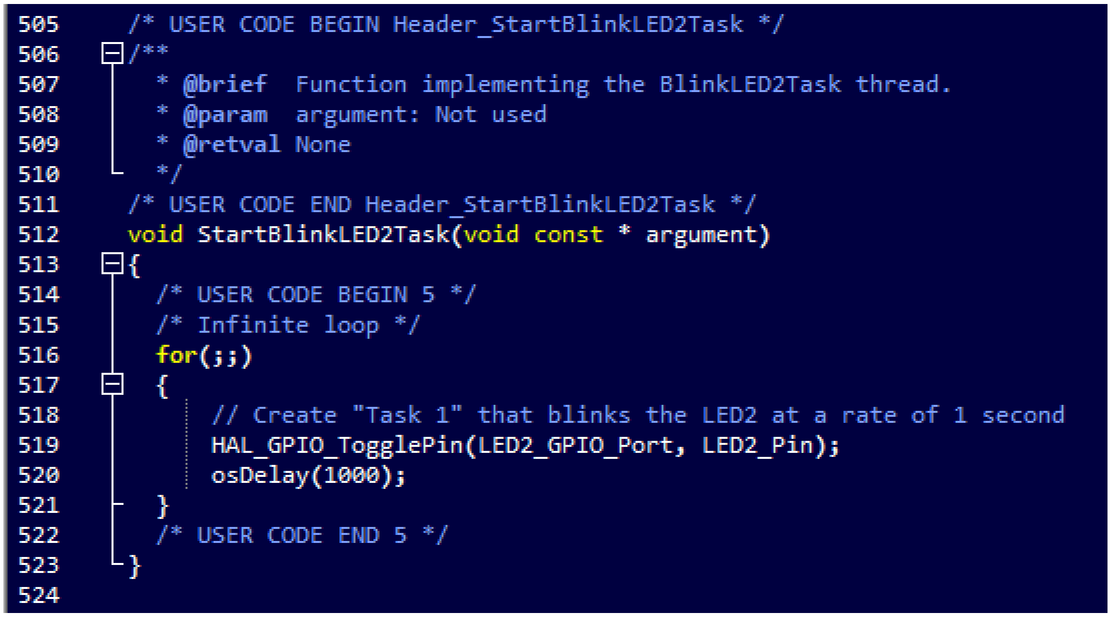
Timer Name	Callback	Type	Code Generation Option	Parameter	Allocation	Control Block Name
MyTimerOneShot	MyTimerOneShotCallback	osTimerOnce	Default	NULL	Dynamic	NULL
MyTimerAutoReload	MyTimerAutoReloadCallback	osTimerPeriodic	Default	NULL	Dynamic	NULL

Nathan Bunnell
Embedded Real-Time Operating Systems
ECE-40290
Student ID: U08895857

Date: 10/25/2021

1. Create "Task 1" that blinks the LED2 at a rate of 1 second

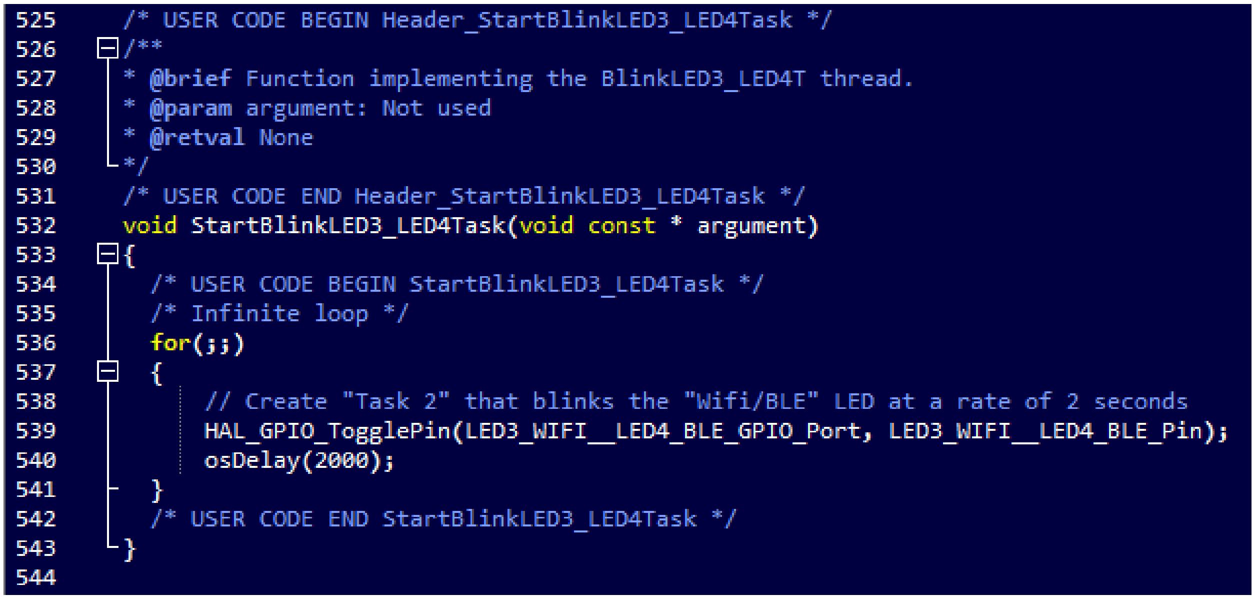
As in the Week 3 assignment, task code is fairly straightforward to implement in each case, *StartBlinkLED2Task()* seen here:



```
/* USER CODE BEGIN Header_StartBlinkLED2Task */
/**
  * @brief  Function implementing the BlinkLED2Task thread.
  * @param  argument: Not used
  * @retval None
  */
/* USER CODE END Header_StartBlinkLED2Task */
void StartBlinkLED2Task(void const * argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
      // Create "Task 1" that blinks the LED2 at a rate of 1 second
      HAL_GPIO_TogglePin(LED2_GPIO_Port, LED2_Pin);
      osDelay(1000);
  }
  /* USER CODE END 5 */
}
```

2. Create "Task 2" that blinks the "Wifi/BLE" LED at a rate of 2 seconds

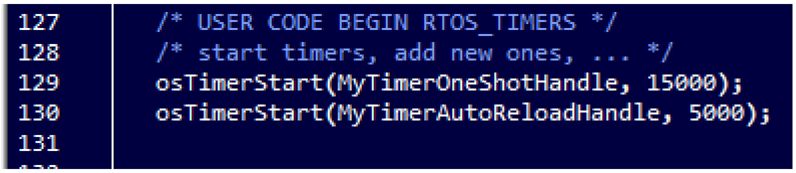
Again for *StartBlinkLED3_LED4Task()*:



```
/* USER CODE BEGIN Header_StartBlinkLED3_LED4Task */
/**
* @brief Function implementing the BlinkLED3_LED4T thread.
* @param argument: Not used
* @retval None
*/
/* USER CODE END Header_StartBlinkLED3_LED4Task */
void StartBlinkLED3_LED4Task(void const * argument)
{
  /* USER CODE BEGIN StartBlinkLED3_LED4Task */
  /* Infinite loop */
  for(;;)
  {
      // Create "Task 2" that blinks the "Wifi/BLE" LED at a rate of 2 seconds
      HAL_GPIO_TogglePin(LED3_WIFI__LED4_BLE_GPIO_Port, LED3_WIFI__LED4_BLE_Pin);
      osDelay(2000);
  }
  /* USER CODE END StartBlinkLED3_LED4Task */
}
```

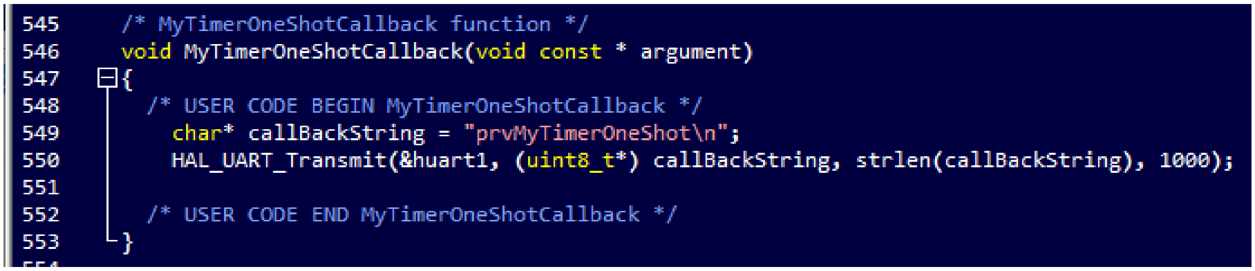
3. Create "Timer 1" that is a One-Shot timer function named prvMyTimerOneShot()

Moving to the timer function calls, we will need to make calls to osTimerStart() for each one, with the appropriate period value as a parameter:



```
  /* USER CODE BEGIN RTOS_TIMERS */
  /* start timers, add new ones, ... */
  osTimerStart(MyTimerOneShotHandle, 15000);
  osTimerStart(MyTimerAutoReloadHandle, 5000);
```

From there, the callback function for each is defined in a similar manner to the task definitions, see *MyTimerOneShotCallback()* here:

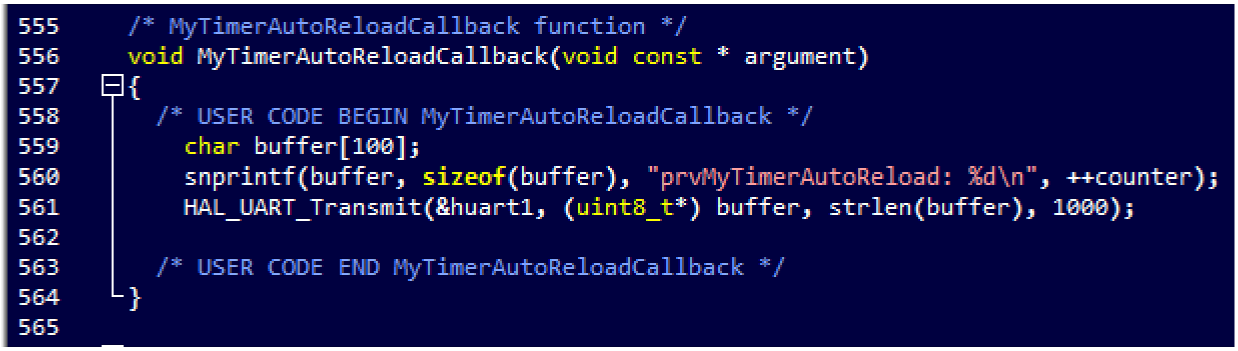


```
/* MyTimerOneShotCallback function */
void MyTimerOneShotCallback(void const * argument)
{
  /* USER CODE BEGIN MyTimerOneShotCallback */
    char* callBackString = "prvMyTimerOneShot\n";
    HAL_UART_Transmit(&huart1, (uint8_t*) callBackString, strlen(callBackString), 1000);

  /* USER CODE END MyTimerOneShotCallback */
}
```

4. Create "Timer 2" that is an auto-reload timer function named prvMyTimerAutoReload()

MyTimerAutoReloadCallback() is defined similarly:



```
/* MyTimerAutoReloadCallback function */
void MyTimerAutoReloadCallback(void const * argument)
{
  /* USER CODE BEGIN MyTimerAutoReloadCallback */
    char buffer[100];
    snprintf(buffer, sizeof(buffer), "prvMyTimerAutoReload: %d\n", ++counter);
    HAL_UART_Transmit(&huart1, (uint8_t*) buffer, strlen(buffer), 1000);

  /* USER CODE END MyTimerAutoReloadCallback */
}
```

Nathan Bunnell
Embedded Real-Time Operating Systems
ECE-40290
Student ID: U08895857

Date: 10/25/2021

Closing Thoughts

Again, the material presented made for an easy to implement assignment using these building blocks. As the course progresses, I'm reminded of my intro to ladder logic courses in college, where these simple ideas are eventually tied in together to build something like a traffic light simulator (or a coffee machine?) by the end of the term. Additionally, these sections on timers and tasks have led to more than one "AHA"-moments at work as I suddenly see similarities between FreeRTOS and whatever proprietary OS that Allen-Bradly controllers use for time and task scheduling.