REAL TIME APPLICATION NEDİR?

Real-time Application (Gerçek Zamanlı Uygulama), gerçek zamanlı işlemleri yürütmek için tasarlanmış bir yazılımdır. Gerçek zamanlı uygulamalar, belirli bir zamanda belirli bir işlemi tamamlamak zorundadır ve gecikmeye toleransı yoktur. Bu tür uygulamalar çeşitli endüstriyel ve emniyet kritik sistemlerde kullanılır, örneğin otomatik sürüş sistemleri, hava trafik kontrol sistemleri, kritik tıbbi ekipmanlar, enerji yönetimi sistemleri, kontrol sistemleri veya üretim tesisleri gibi. Gerçek zamanlı uygulamalar için RTOS (Real-time Operating System) veya RTA (Real-time Automation) kullanılır.

RTOS NEDİR?

Real-time operating system (RTOS) gerçek zamanlı işletim sistemi olarak tanımlanır. Bu tür işletim sistemleri, öncelikli görevleri gerçek zamanlı olarak yerine getirmek için tasarlanmıştır. RTOS, genellikle tahmini işlem süreleri olan görevleri yürütmek için kullanılır ve görevler arasındaki etkileşimi kontrol etmek için kullanılır. Örneğin, bir otomatik sürüş sistemi veya bir hava trafik kontrol sistemi gibi endüstriyel veya emniyet kritik uygulamalar için RTOS kullanılabilir.

RTOS HANGİ AMAÇLA TASARLANMIŞTIR?

RTOS, gerçek zamanlı sistemler için tasarlanmıştır. Gerçek zamanlı sistemler, belirli bir zamanda belirli bir işlemi tamamlamak zorundadır. Örneğin, bir otomatik sürüş sistemi, sürücünün fren pedalına basması anında fren yapması gerekir. Bu tür sistemler için gecikmeye toleransı yoktur.

RTOS, gerçek zamanlı işlemleri yürütmek için öncelikli görevleri desteklemek için tasarlanmıştır. Bu, gerçek zamanlı işlemlerin zamanında yerine getirilmesini sağlar. RTOS aynı zamanda görevler arasındaki etkileşimi kontrol etmek için kullanılır. Örneğin, bir RTOS, bir görevin diğer görevleri etkilememesi için gerekli olan güvenli bir yol sağlar.
RTOS ayrıca, sistem kaynaklarının etkili bir şekilde yönetilmesini sağlar. Örneğin, bir RTOS, bellek ve CPU kaynaklarını etkili bir şekilde yönetebilir ve görevler arasında paylaştırabilir.
Son olarak, RTOS, çoklu görevli işletim sistemi olarak tasarlanmıştır. Bu, birden fazla görevin aynı anda yürütülmesini veya aralıklı olarak yürütülmesini sağlar. Bu, sistemler için daha yüksek verimliliği ve daha yüksek performansı sağlar.

OPEN VE SAFE RTOS NEDİR?

Open RTOS, kaynak kodları açık olan gerçek zamanlı işletim sistemidir. Bu, kullanıcıların sistemi özelleştirmek, uyarlayarak veya eklemeler yaparak kendi ihtiyaçlarına göre kullanabilmelerine olanak tanır. Open RTOS, genellikle açık kaynak kodlu projelerde veya ticari amaçlar için kullanılır.

Safe RTOS, güvenli gerçek zamanlı işletim sistemi olarak tanımlanır. Bu tür işletim sistemleri, görevler arasındaki etkileşimi kontrol etmek için güvenli bir yol sağlar ve sistem kaynaklarının etkili bir şekilde yönetilmesini sağlar. Safe RTOS genellikle emniyet kritik sistemlerde kullanılır. Örneğin, hava trafik kontrol sistemleri, otomatik sürüş sistemleri veya tıbbi ekipmanlar gibi sistemlerde.

GPOS NEDİR?

General Purpose Operating System (GPOS) genel amaçlı işletim sistemi olarak tanımlanır. Bu tür işletim sistemleri, çeşitli uygulamalar için kullanılabilir ve çeşitli işlemleri yürütmek için tasarlanmıştır. GPOS genellikle masaüstü veya dizüstü bilgisayarlar, sunucular ve mobil cihazlar gibi cihazlar için kullanılır.

GPOS, çoklu görevli işletim sistemi olarak tasarlanmıştır. Bu, birden fazla görevin aynı anda yürütülmesini veya aralıklı olarak yürütülmesini sağlar. GPOS ayrıca, sistem kaynaklarının etkili bir şekilde yönetilmesini sağlar. Örneğin, GPOS, bellek ve CPU kaynaklarını etkili bir şekilde yönetebilir ve görevler arasında paylaştırabilir. GPOS genellikle gerçek zamanlı sistemlerde kullanılmaz.
Popüler GPOS örnekleri arasında Windows, MacOS, Linux, iOS, Android, gibi sistemler yer alır.

GPOS İLE RTOS ARASINDAKİ FARKLAR NELERDİR?

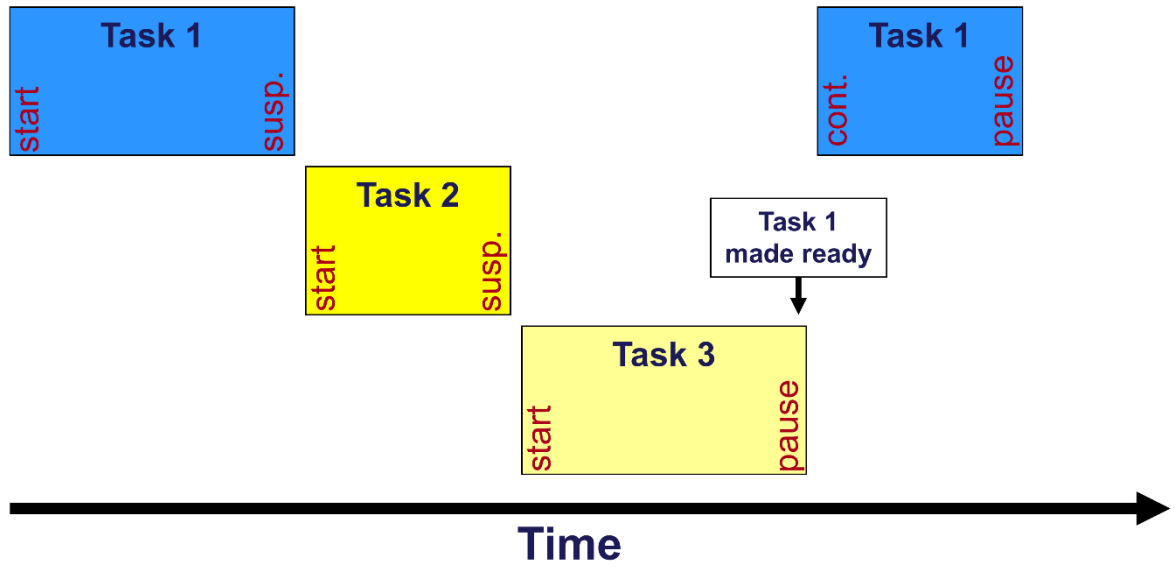
- Gecikme Toleransı: GPOS genellikle gecikmeye toleranslıdır, ancak RTOS gerçek zamanlı işlemler için gecikmeye toleransı yoktur.
- Öncelikli Görevler: GPOS genellikle öncelikli görevleri desteklememekte, ancak RTOS öncelikli görevleri desteklemek için tasarlanmıştır.
- Görevler Arası Etkileşim: GPOS genellikle görevler arası etkileşimi kontrol etmemekte, ancak RTOS görevler arası etkileşimi kontrol etmek için tasarlanmıştır.
- Sistem Kaynaklarının Yönetimi: GPOS ve RTOS sistem kaynaklarının etkili bir şekilde yönetilmesini sağlar, ancak RTOS' un gerçek zamanlı sistemler için optimize edilmiş olması nedeniyle kaynak yönetimi daha önemlidir.
- Kullanım Alanları: GPOS genellikle masaüstü veya dizüstü bilgisayarlar, sunucular ve mobil cihazlar gibi cihazlar için kullanılırken, RTOS genellikle endüstriyel ve emniyet kritik uygulamalar için kullanılır. Örneğin otomatik sürüş sistemleri, hava trafik kontrol sistemleri, kritik tıbbi ekipmanlar, enerji yönetimi sistemleri, kontrol sistemleri veya üretim tesisleri.

FreeRTOS NEDİR?

FreeRTOS, açık kaynak kodlu, gerçek zamanlı işletim sistemi (RTOS) olarak tanımlanır. FreeRTOS, küçük ve orta ölçekli mikroişlemcili sistemler için tasarlanmıştır ve ücretsiz olarak kullanılabilir. FreeRTOS, öncelikli görevleri destekler, görevler arasındaki etkileşimi kontrol eder ve sistem kaynaklarını etkili bir şekilde yönetir. FreeRTOS ayrıca, bellek yönetimi, veri yapıları ve diğer işlevleri içeren birçok kütüphane içerir. FreeRTOS, masaüstü veya dizüstü bilgisayarlar, sunucular ve mobil cihazlar gibi cihazlar için kullanılabilir.

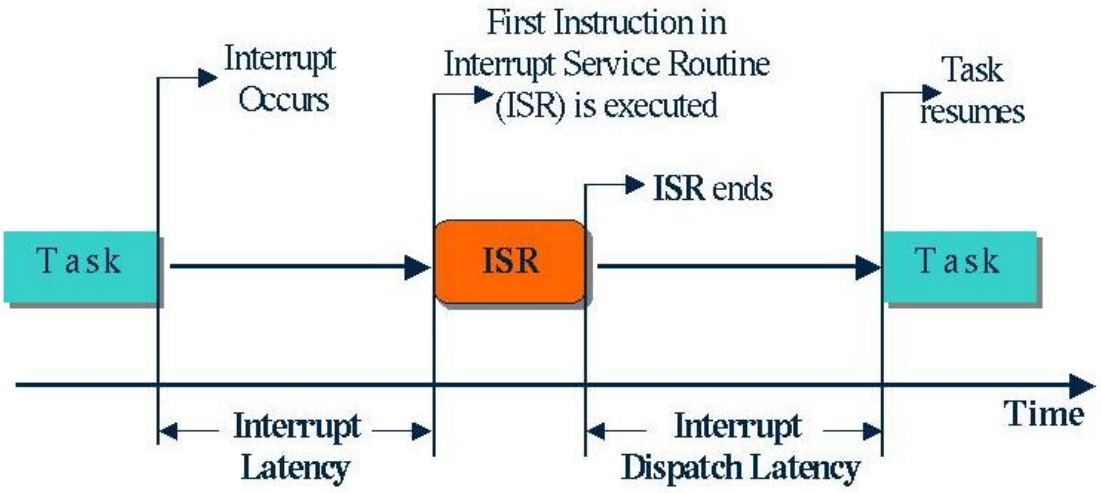
GÖREV ZAMANLAMA (TASK SCHEDULING) NEDİR?

Task scheduling, işletim sistemi tarafından yürütülen görevlerin zamanlamasını ve atandıkları kaynakları yönetme işlemidir. Görev planlama, görevlerin belirli bir sıraya göre çalışmasını sağlar ve görevler arasında kaynakları (özellikle CPU ve bellek) eşit bir şekilde paylaştırır. Görev planlama ayrıca, görevlerin öncelik sırasını belirlemek için kullanılır ve görevler arasındaki etkileşimi kontrol etmek için kullanılır. Görev planlama, gerçek zamanlı işletim sistemlerinde önemlidir ve gerçek zamanlı uygulamalar için gereklidir.



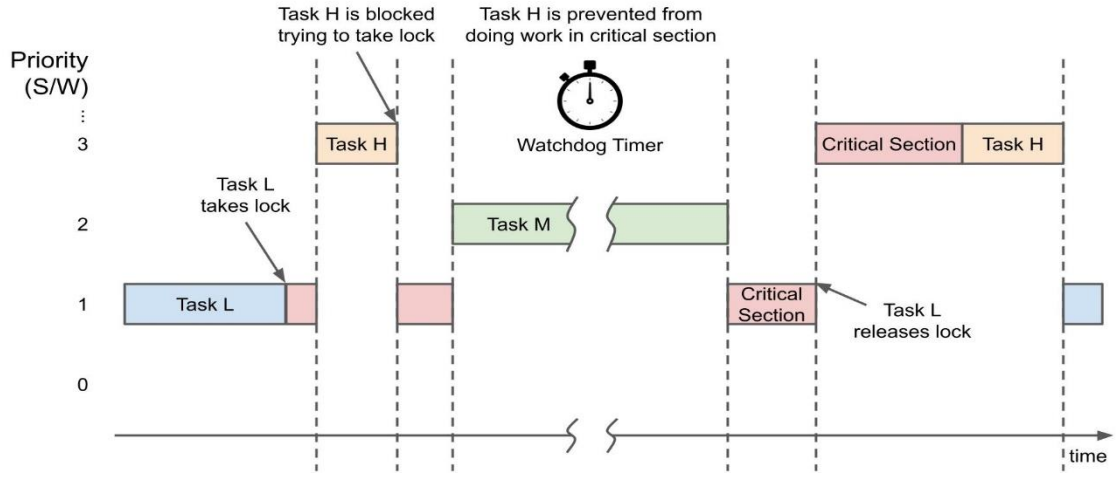
KESME GECİKMESİ (INTERRUPT LATENCY) NEDİR?

Interrupt Latency, gerçek zamanlı işletim sistemlerinde (RTOS) bir önemli kavramdır. Interrupt Latency, bir araya gelme (interrupt) sırasında işlemcinin araya gelme kaynağını işleme başlamasına kadar geçen zamandır. Bu zaman, araya gelme kaynağının işlemcinin öncelikli görevlerine karşı kesintiye uğramasına neden olabilir. Bu nedenle, RTOS'lar genellikle düşük araya gelme gecikmesi (low interrupt latency) sağlamak için tasarlanır. Düşük araya gelme gecikmesi, araya gelme kaynağının işlemcinin öncelikli görevlerine karşı kesintiye uğramasını azaltır ve gerçek zamanlı uygulamalar için gerekli olan gecikme toleransını sağlar.



PRIORITY INVERSION NEDİR?

Priority Inversion, gerçek zamanlı işletim sistemlerinde (RTOS) bir önemli kavramdır. Priority Inversion, öncelikli görevlerin düşük öncelikli görevler tarafından bloke edilmesi sonucu ortaya çıkar. Örneğin, öncelikli görev A, düşük öncelikli görev B tarafından kullanılan bir kaynağı (örneğin, bellek veya semafor) beklerken, düşük öncelikli görev C bu kaynağı alır ve görev B tarafından bloke edilir. Bu durumda, görev A bloke edilir ve öncelikli görevler arasındaki etkileşim kontrol edilmemiştir. Priority Inversion, gerçek zamanlı uygulamalar için gerekli olan gecikme toleransını azaltır ve öncelikli görevlerin performansını etkileyebilir.



CMSIS NEDİR?

Cortex Microcontroller Software Interface Standard (CMSIS), ARM tarafından geliştirilen bir ara yazılım standartıdır. Bu standart, Cortex-M serisi mikroişlemciler için yazılım geliştirmeyi kolaylaştırmak amacıyla tasarlanmıştır. CMSIS, mikroişlemciler arasında uyumlu bir şekilde yazılım geliştirmenizi sağlar. Bu sayede, aynı yazılım kodunu farklı Cortex-M mikroişlemcilerinde çalıştırabilirsiniz.

CMSIS, birçok özellik sunar. Örneğin,

- Cortex-M serisi mikroişlemciler için temel işlevleri sağlar,
- yazılım geliştirme ortamları için sürücüler ve araçlar sağlar,
- bellek yönetimi, sistem çağrıları ve diğer özellikleri sağlar.

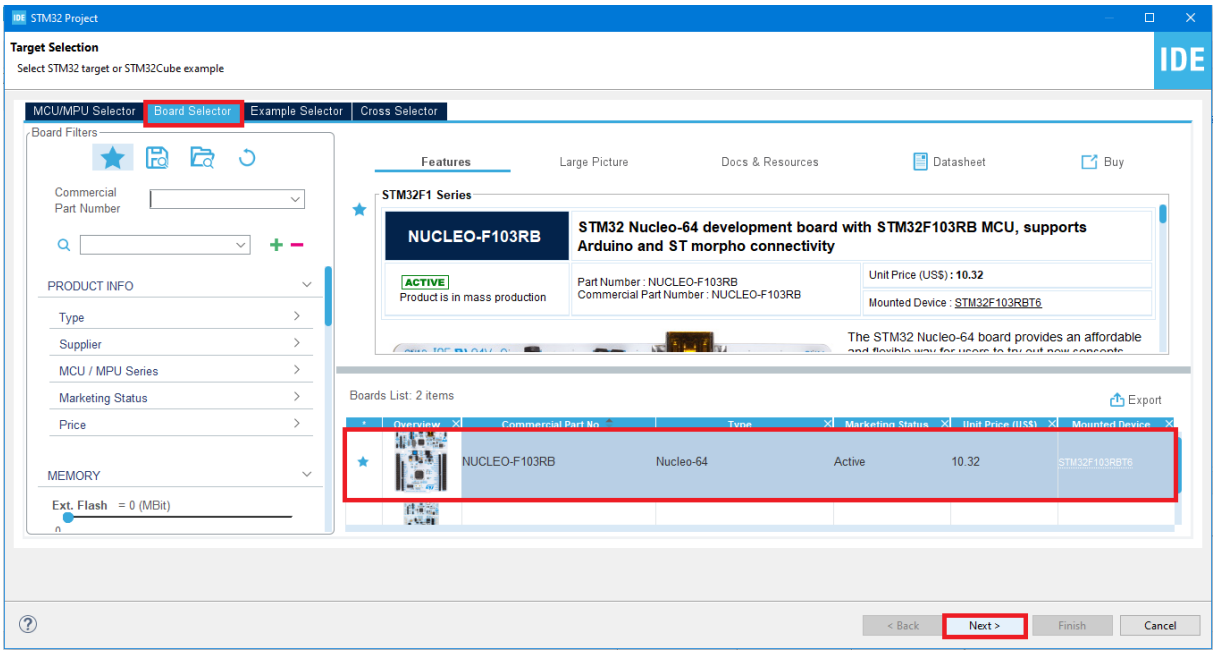
“CMSIS_V1” ve “CMSIS_V2” olmak üzere iki farklı sürümü vardır. CMSIS_V1, Cortex-M serisi mikroişlemciler için temel işlevleri sağlayan düşük seviyeli bir ara yazılımdır. CMSIS_V2 ise, CMSIS_V1'in üzerine yapılmış, daha yüksek seviyeli bir ara yazılımdır. CMSIS_V2, daha yüksek seviyeli özellikler ve daha geniş bir destek sunar. Örneğin, CMSIS_V2, Cortex-M serisi mikroişlemciler için sistem çağrıları (system calls) ve bölgesel bellek erişimi gibi özellikleri sunar.

CMSIS_V1: Eğer projeniz için düşük seviyede bir destek yeterli ise veya projeniz için özel bir özellik gerekli değilse, CMSIS_V1 kullanabilirsiniz. Bu sürüm, Cortex-M serisi mikroişlemciler için temel işlevleri sağlar ve daha az yer kaplar.

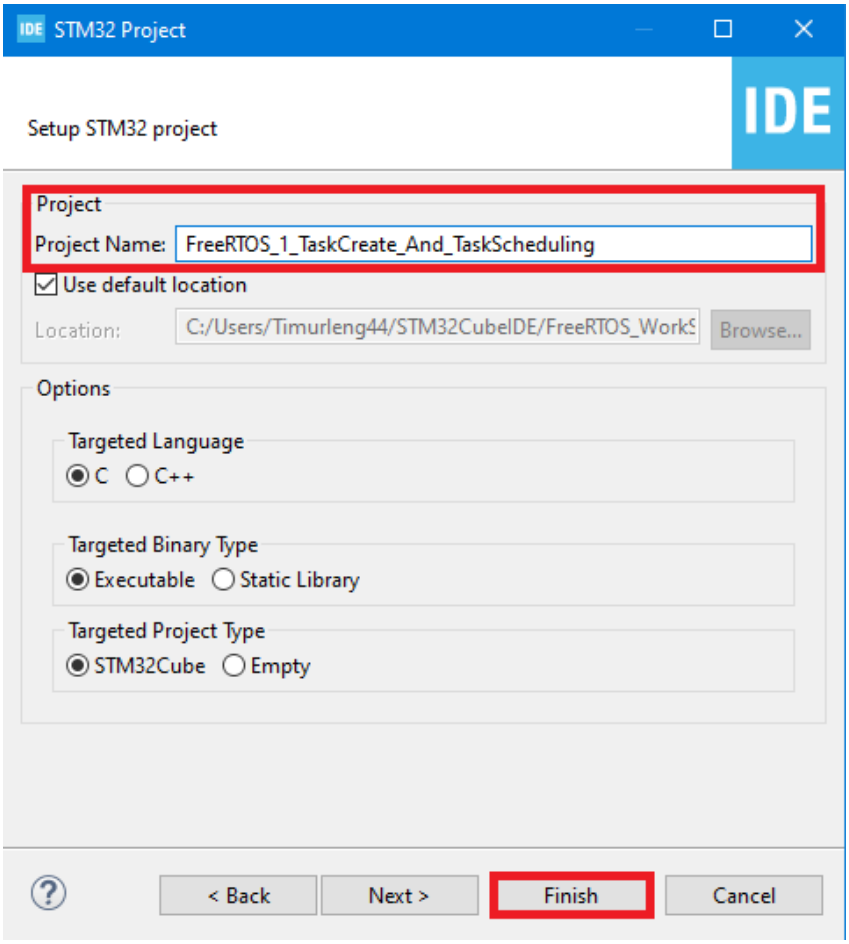
CMSIS_V2: Eğer projeniz için daha yüksek seviyede bir destek veya özel bir özellik gerekli ise, CMSIS_V2'yi kullanabilirsiniz.

FreeRTOS’ DA GÖREV OLUŞTURMA VE ZAMANLAMA PROJESİ

Öncelikle yeni bir proje oluşturuyoruz ve NUCLEO-F103RB kartımızı seçiyoruz ardından next butonuna basıyoruz.



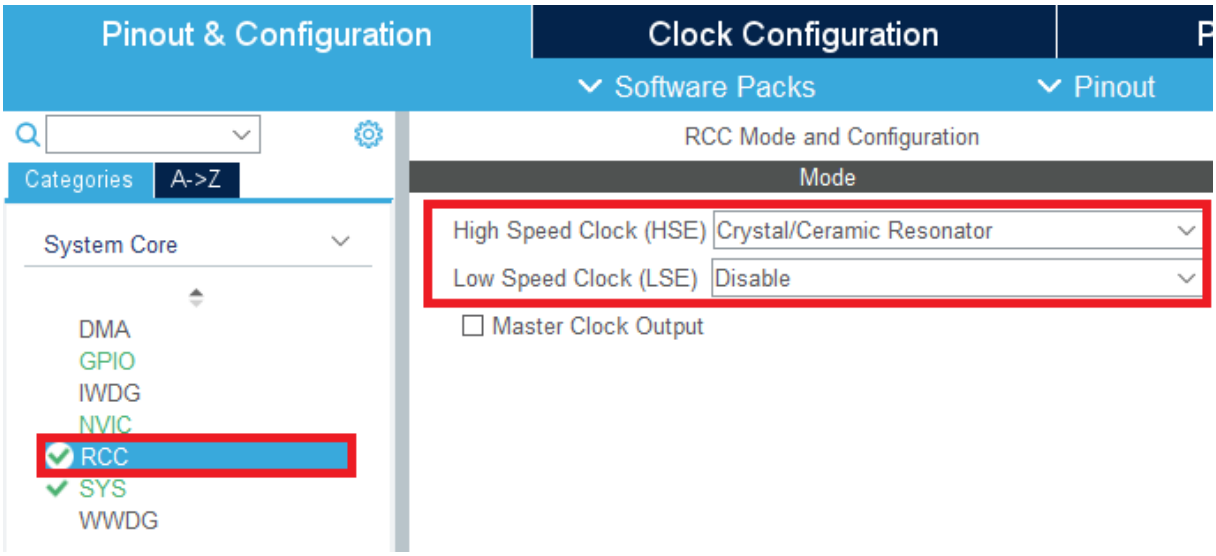
Bu adımda Projeyi isimlendirip Finish butonuna basıyoruz böylelikle projeyi oluşturmuş olduk.



Bu adımda STM32CubeMX üzerinden konfigrasyonları yapacağız

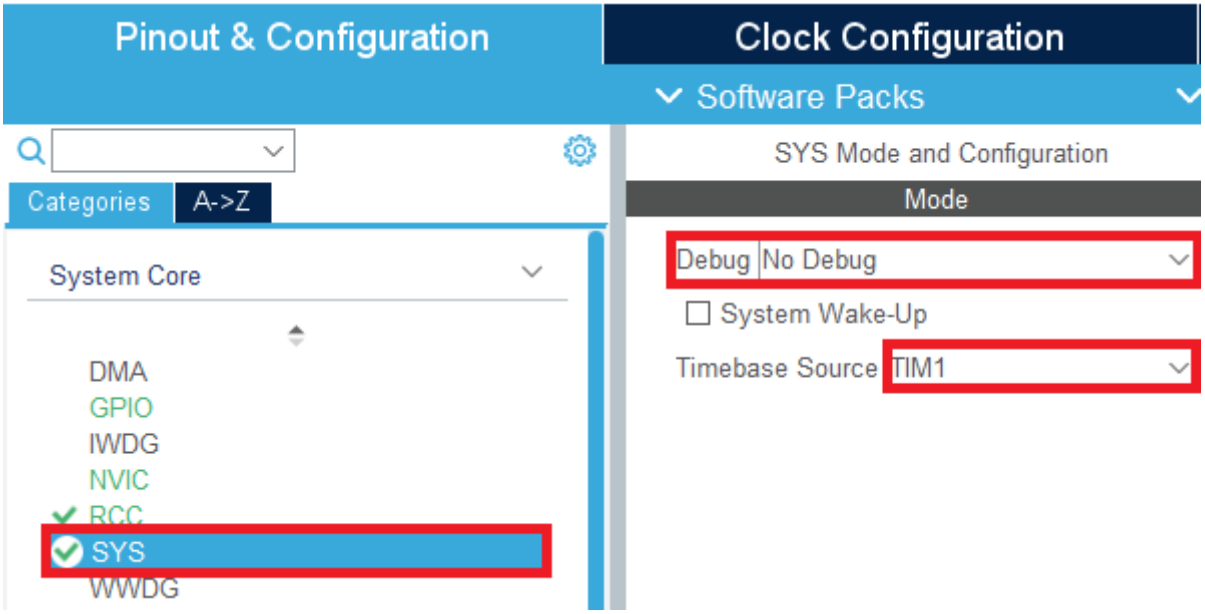
İlk aşama:

Pinout & Configration -> System Core -> RCC git ardından LSE kısmında DISABLE seçeneğini, HSE kısmında ise Crystal/Ceramic Resonator seçeneğini seç.



İkinci aşama:

Pinout & Configration -> System Core -> SYS git ardından DEBUG kısmında “NO DEBUG” seçeneğini, TimeBase Source kısmında ise “TIM1” seçeneğini seç.



NOT: TimeBase Source mikroişlemcinin zaman temelini sağlayan kaynak olarak kullanılan kaynakları tanımlar. Bu kaynaklar arasında, mikroişlemcinin içindeki kristal osilatör, çoklu veya teklu osilatör, internal RC osilatör gibi kaynaklar bulunabilir. Timebase source sayesinde, mikroişlemci üzerinde çalışan uygulama veya sistemin zaman ve tarih bilgilerine dayalı işlemleri gerçekleştirebilirsiniz.

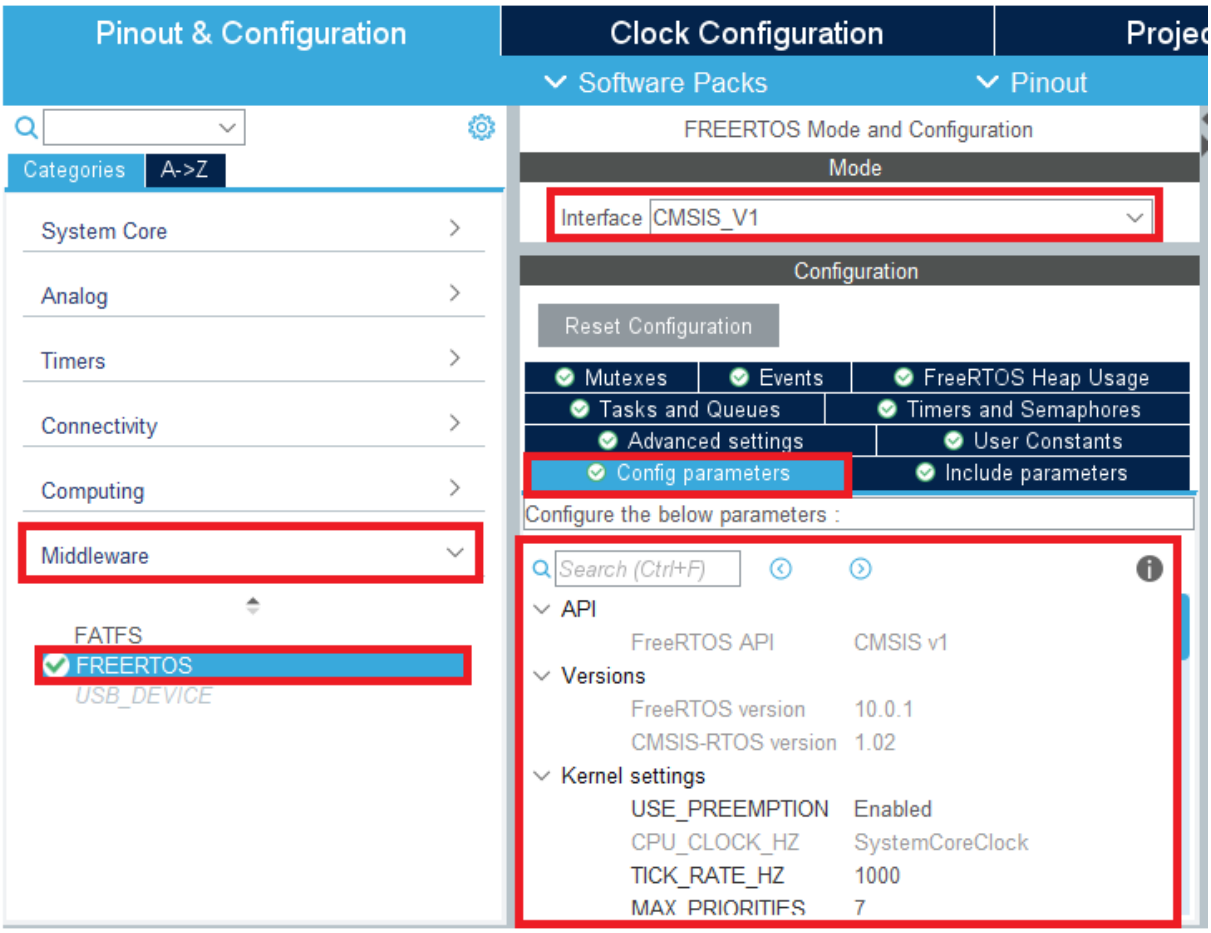
Örnek olarak aşağıdaki işlemler verilebilir:

- sistemin çalışma zamanını ölçmek.
- sistemin çalışma süresi arasındaki farkı hesaplamak.
- sistem üzerinde belirli bir zaman aralığı içinde işlem yapmak.
- sistemin belirli bir zaman aralığı içinde çalışmasını sağlamak.

Bu adımda FreeRTOS ayarlamalarını yapacağız.

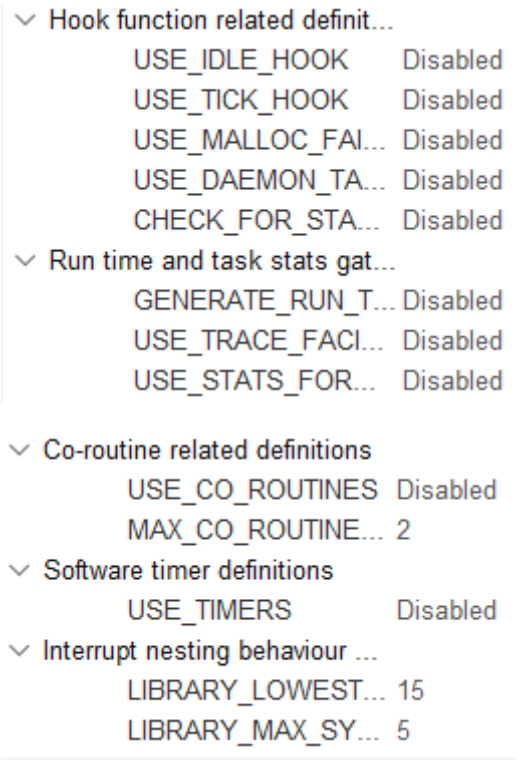
İlk aşama:

Pinout & Configuration -> Middleware -> FREERTOS kısmına gidiyor Interface kısmından CMSIS_V1’ i seçiyoruz ve Config parameters kısmından konfigürasyon parametrelerini aşağıda bulunan resimlerde bulunduğu gibi ayarlıyoruz.



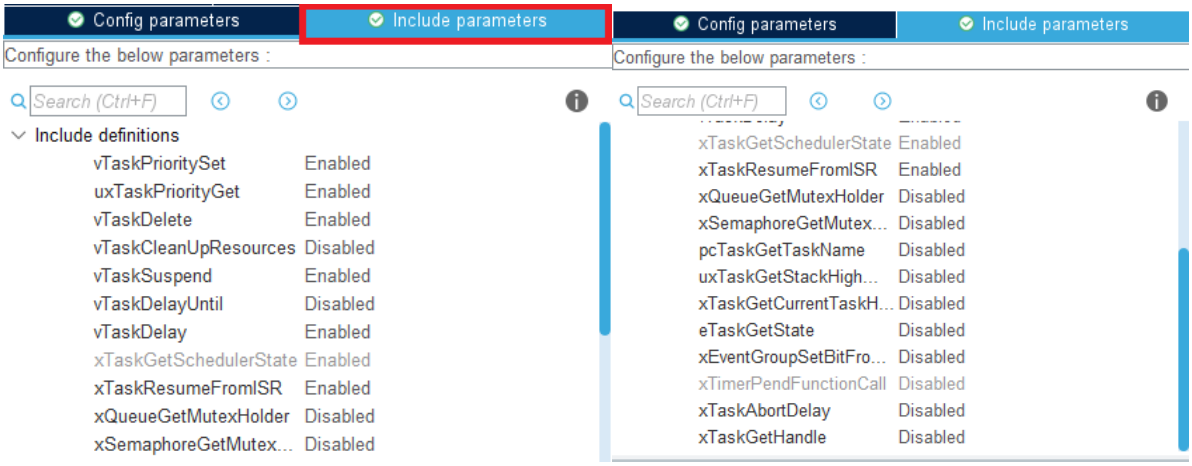
MAX_PRIORITIES	7
MINIMAL_STACK_S...	128 Words
MAX_TASK_NAME...	16
USE_16_BIT_TICKS	Disabled
IDLE_SHOULD_YIE...	Enabled
USE_MUTEXES	Enabled
USE_RECURSIVE_...	Disabled
USE_COUNTING_S...	Disabled
QUEUE_REGISTRY...	8
USE_APPLICATION...	Disabled
ENABLE_BACKWA...	Enabled
USE_PORT_OPTIMI...	Enabled

USE_PORT_OPTIMI...	Enabled
USE_TICKLESS_IDLE	Disabled
USE_TASK_NOTIFI...	Enabled
RECORD_STACK_...	Disabled
Memory management setti...	
Memory Allocation	Dynamic / Static
TOTAL_HEAP_SIZE	3072 Bytes
Memory Manageme...	heap_4
Hook function related definit...	
USE_IDLE_HOOK	Disabled
USE_TICK_HOOK	Disabled
USE_MALLOC_FAI...	Disabled



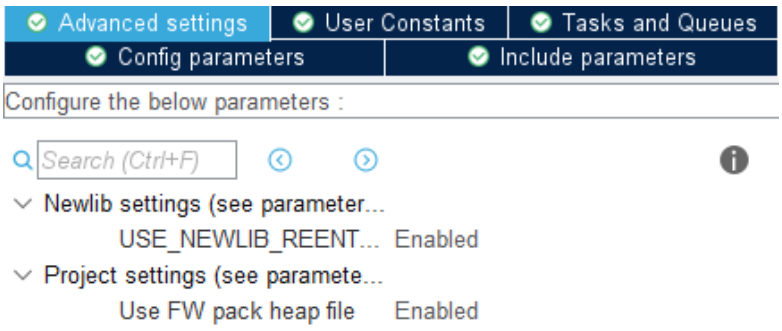
İkinci aşama:

FREERTOS kısmında Include parameters kısmında bulunan parametreleri düzenliyoruz.



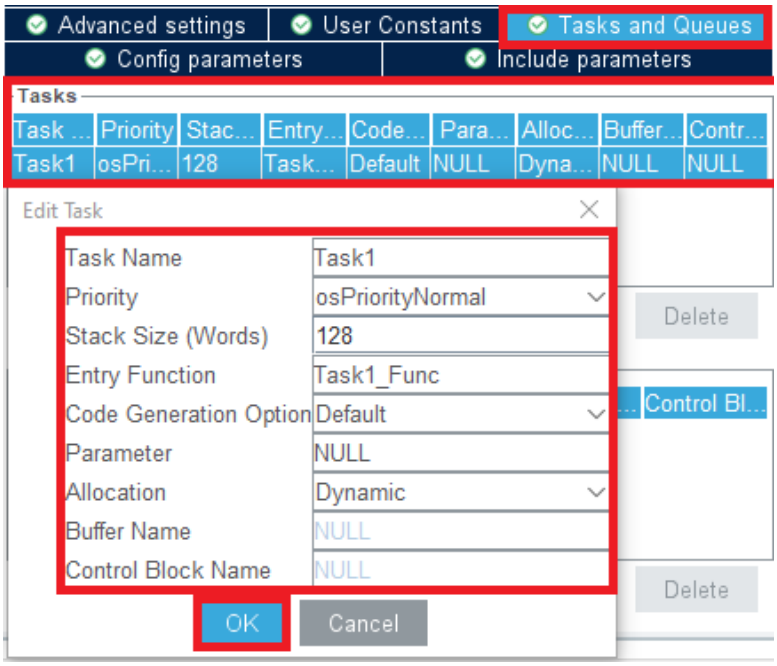
Üçüncü aşama:

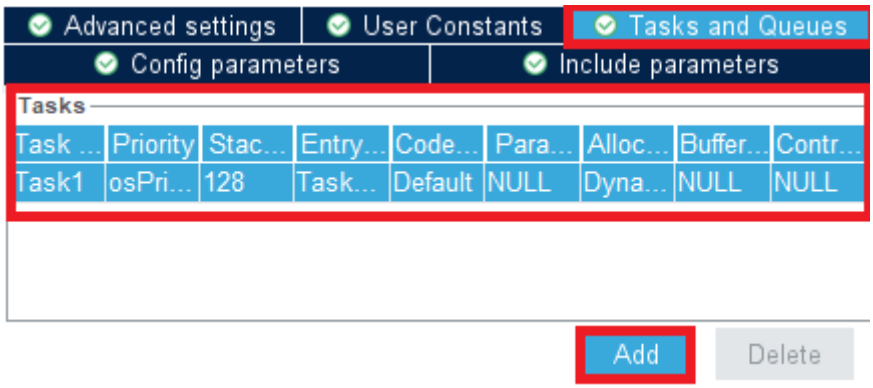
FREERTOS kısmında Advanced parameters kısmında bulunan parametreleri düzenliyoruz.

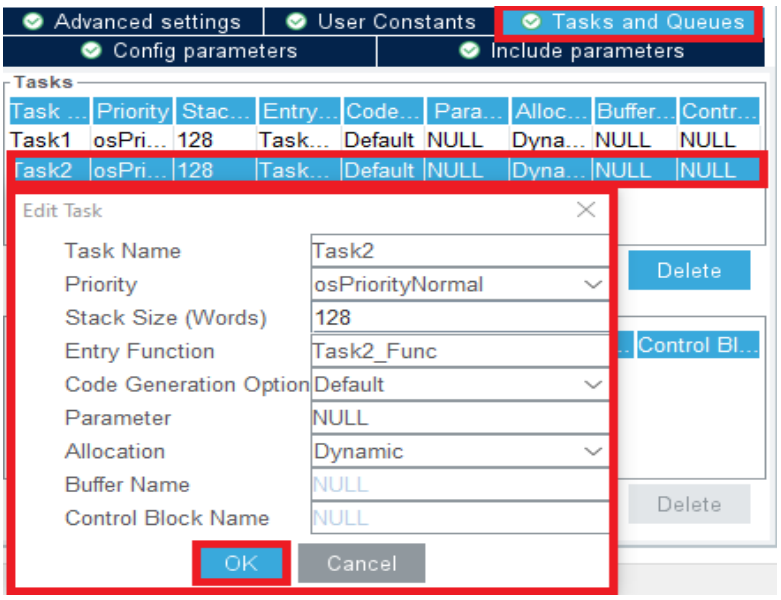


Dördüncü aşama:

FREERTOS kısmında **“Task and Queues”** kısmında default task’ ı düzenleyip ardından yeni bir task daha oluşturuyoruz.

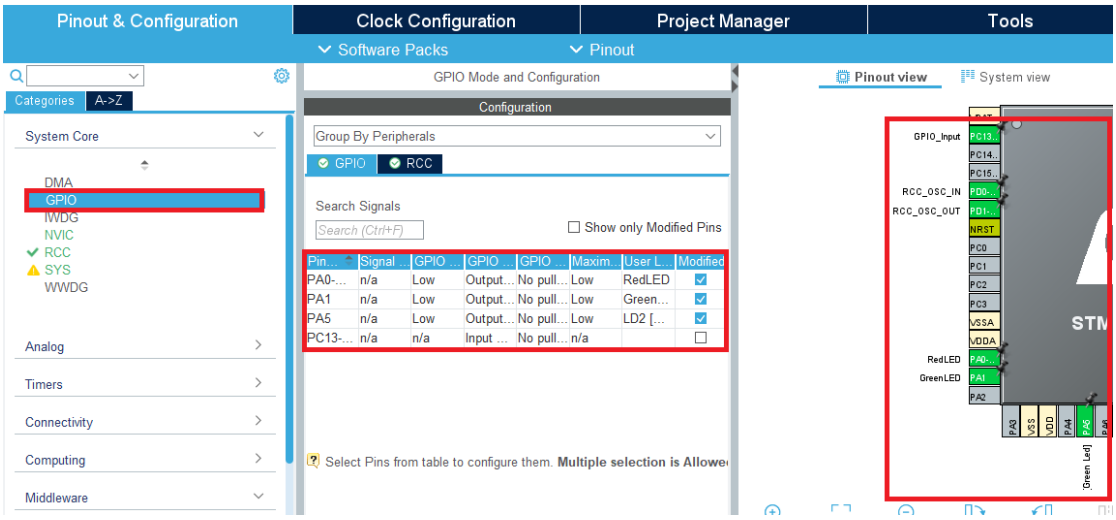




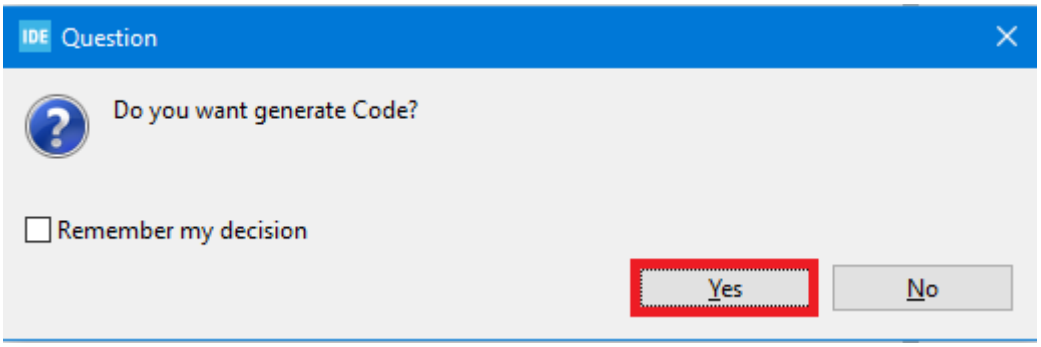


Beşinci aşama:

PA0, PA1 ve PA5’ i OUTPUT, PA3,’ INPUT olarak ayarlıyoruz.



Son olarak “CTRL + S” basıyoruz ve çıkan kutucukta YES butonuna basıyoruz.



Bu adımda yazılım aşamalarını gerçekleştireceğiz

İlk aşama:

osThreadId Task3Handle; değişkenimizi oluşturduk ardından void * Task3_Func(); fonksiyonumuzun prototipini oluşturduk.

```
/* Private variables ---------------------------------------------------------*/
osThreadId Task1Handle;// STM32CubeIDEMX otomatik oluşturuyor
osThreadId Task2Handle;// STM32CubeIDEMX otomatik oluşturuyor
/* USER CODE BEGIN PV */
osThreadId Task3Handle;// Kendimiz oluşturduk
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
void Task1_Func(void const * argument);// STM32CubeIDEMX otomatik oluşturuyor
void Task2_Func(void const * argument);// STM32CubeIDEMX otomatik oluşturuyor

/* USER CODE BEGIN PFP */
void *Task3_Func(void const * argument);// Kendimiz oluşturduk
/* USER CODE END PFP */
```

İkinci aşama:

Task1_Func(); , Task2_Func(); , Task3_Func(); fonksiyonlarımızın içini dolduruyoruz.

```
/* USER CODE END Header_Task2_Func */
void Task2_Func(void const * argument)
{
  /* USER CODE BEGIN Task2_Func */
  /* Infinite loop */
  for(;;)
  {
      HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_1);
      osDelay(500);
  }
  /* USER CODE END Task2_Func */
}
```

```
/* USER CODE END Header_Task1_Func */
void Task1_Func(void const * argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
      HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_0);
      osDelay(500);
  }
  /* USER CODE END 5 */
}
```

```
/* USER CODE BEGIN 4 */
void *Task3_Func(void const * argument)
{
      for(;;)
      {
         if(!HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13))
         {
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
         }
         else
         {
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
         }
      }
}
/* USER CODE END 4 */
```

Üçüncü aşama:

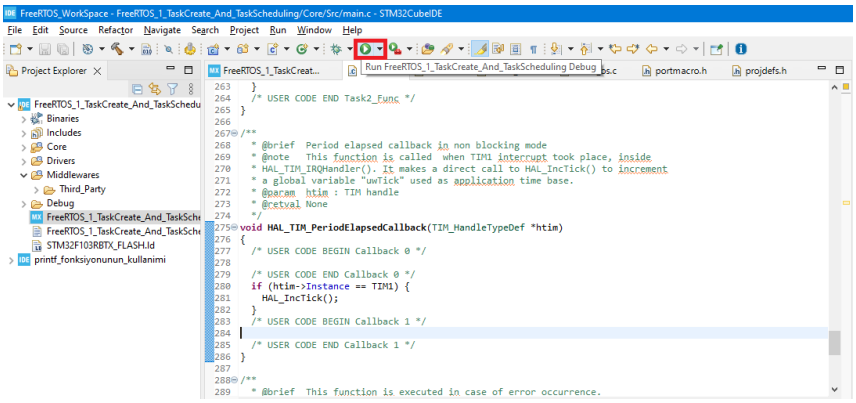
xTaskCreate(); fonksiyonu ile 3. Görevimizi oluşturuyoruz geri kalan görevler STM32CubeIDEMX tarafından otomatik olarak oluşturulduğu için onlar üzerinde işlem yapmamıza gerek yok ayrıca osKernelStart(); fonksiyonu yerine vTaskStartScheduler(); fonksiyonunu kullanabiliriz. Bu fonksiyon ile görev zamanlaması başlatılmış oluyor.

Şu anda tüm görevlerin önceliklerini eşit olarak ayarladık hepsine eşit zamanlar ayrılarak görevler arasında değişim gerçekleştirilecek.

```
/* Create the thread(s) */
/* definition and creation of Task1 */
osThreadDef(Task1, Task1_Func, osPriorityNormal, 0, 128);// STM32CubeIDEMX otomatik oluşturuyor
Task1Handle = osThreadCreate(osThread(Task1), NULL);// STM32CubeIDEMX otomatik oluşturuyor

/* definition and creation of Task2 */
osThreadDef(Task2, Task2_Func, osPriorityNormal, 0, 128);// STM32CubeIDEMX otomatik oluşturuyor
Task2Handle = osThreadCreate(osThread(Task2), NULL);// STM32CubeIDEMX otomatik oluşturuyor

xTaskCreate( Task3_Func, "Task3", configMINIMAL_STACK_SIZE, NULL, 0, &Task3Handle);
/* USER CODE BEGIN RTOS_THREADS */
/* add threads, ... */
/* USER CODE END RTOS_THREADS */

/* Start scheduler */
osKernelStart();//Bu fonksiyon yerine vTaskStartScheduler(); kullanabiliriz ikiside aynı isi yapıyor

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

Dördüncü aşama:

Kodumuzu mikrodenetleyicimize yüklüyoruz.



Yürütülen kodların sonuçlarını github’ da bulunan videodan izleyebilirsiniz.