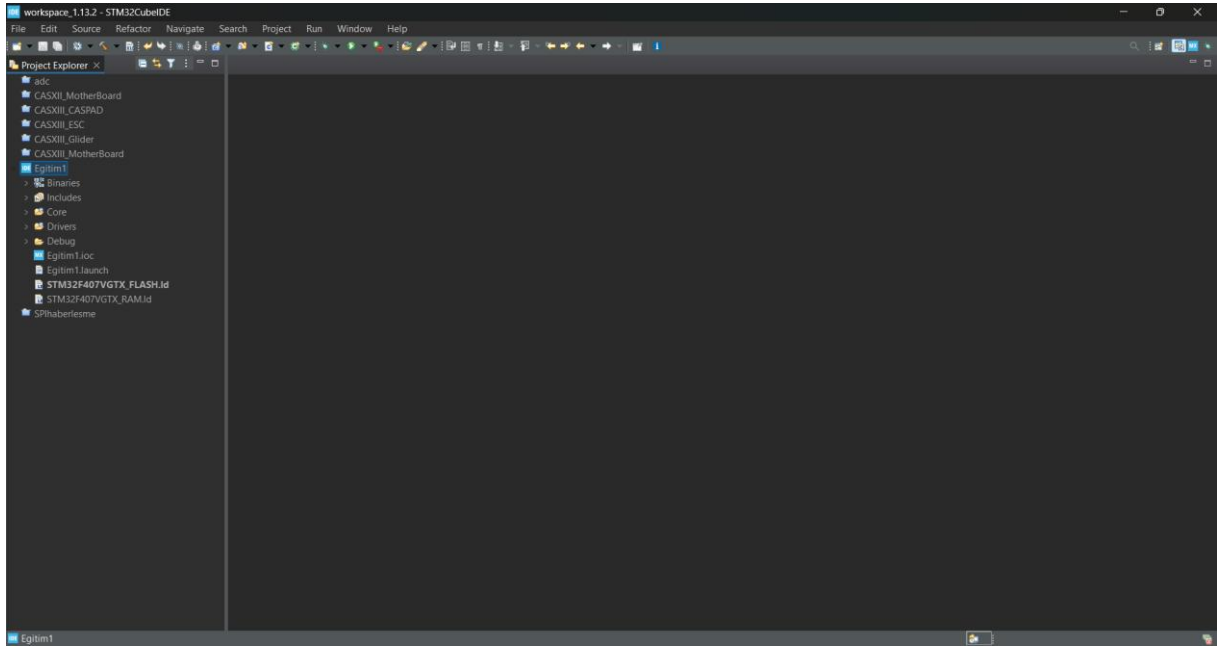


CAS ÇK GÖMÜLÜ YAZILIM 1. EĞİTİM ÇIKTISI

Kullandığımız STM işlemcisi ve kartı: STM32F407VGT6 işlemcisi ve STM32F407G-DISC1 geliştirme kartı.

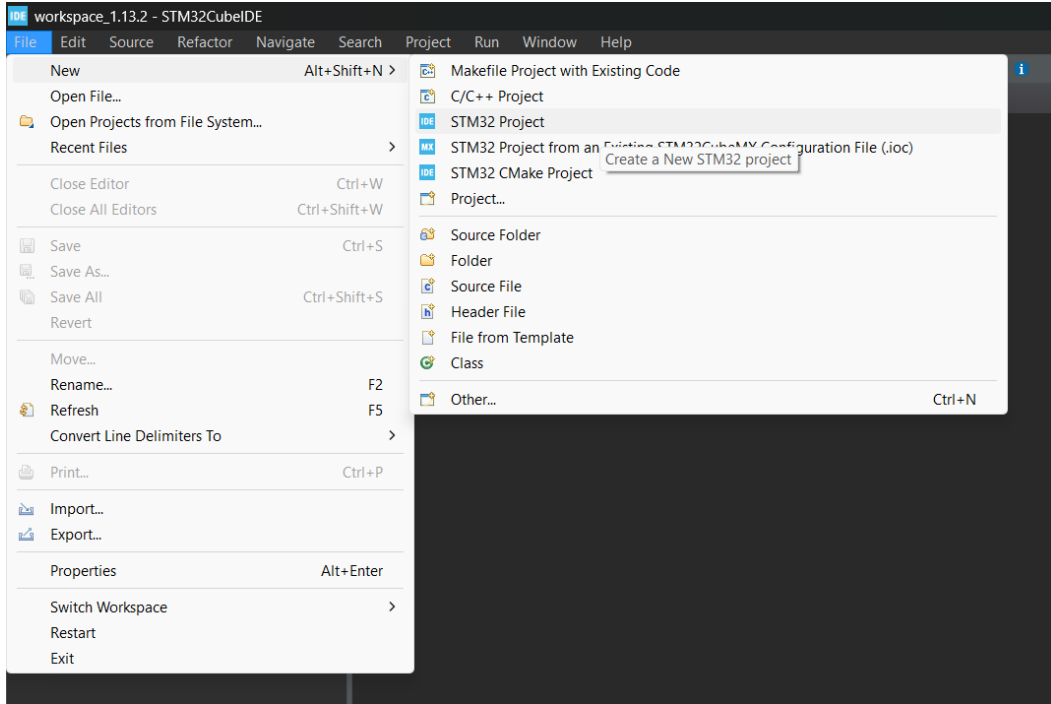


STM32 CubeIDE Arayüz:

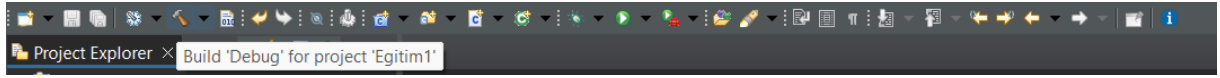


Sol tarafta Project Explorer kısmında workspace' imiz üzerindeki projeler gösterilir.

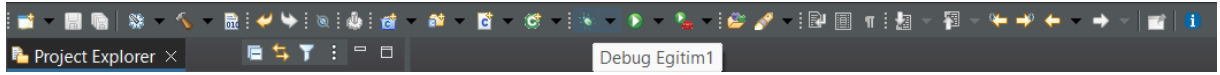
Yeni Proje oluřturma:



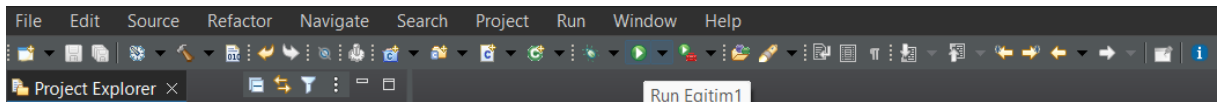
Projeyi Build etme:



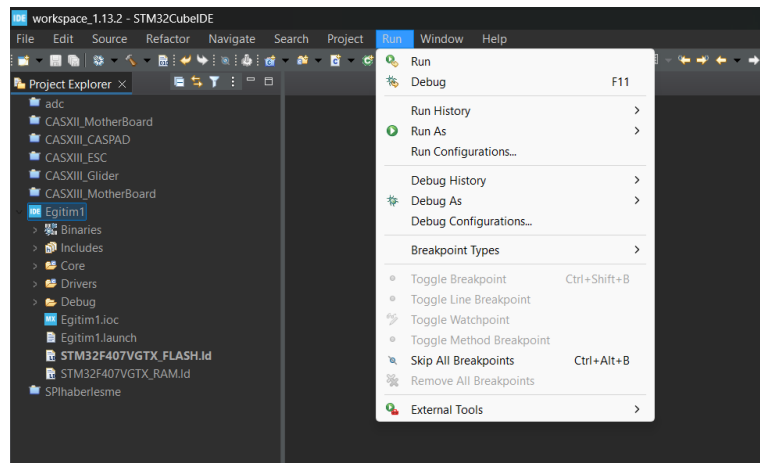
Debug modunda alıřtırma:



Run tuřu:

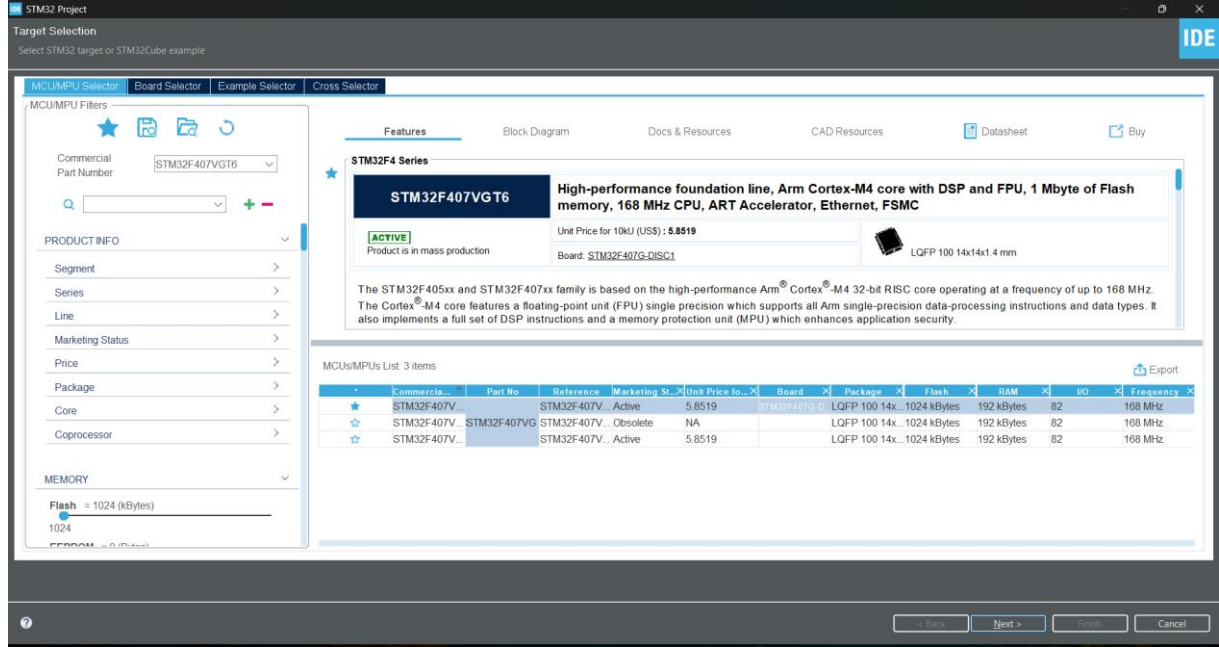


Bu iřlemleri bu pencereden de yapabiliriz:



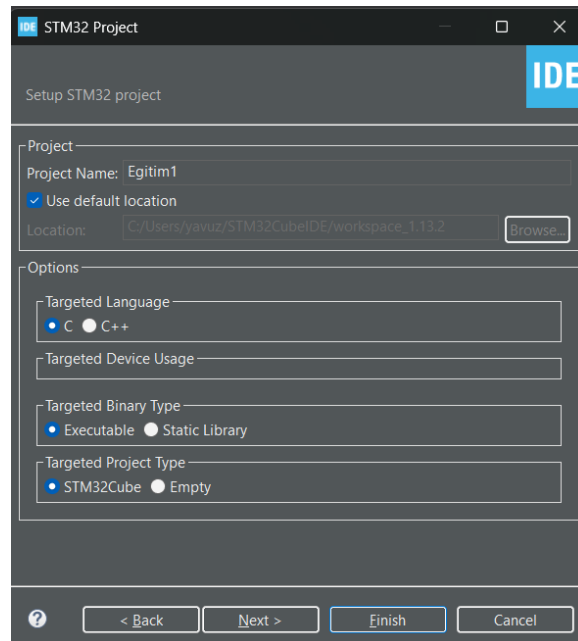
Proje oluřtururken yapılan ayarlar, ioc ekranı ve main.c dosyası:

Bu ekranda iřlemciyi grseldeki gibi seiyoruz ve bir dahaki projelerde kolaylık olması aısından yıldızlıyoruz.

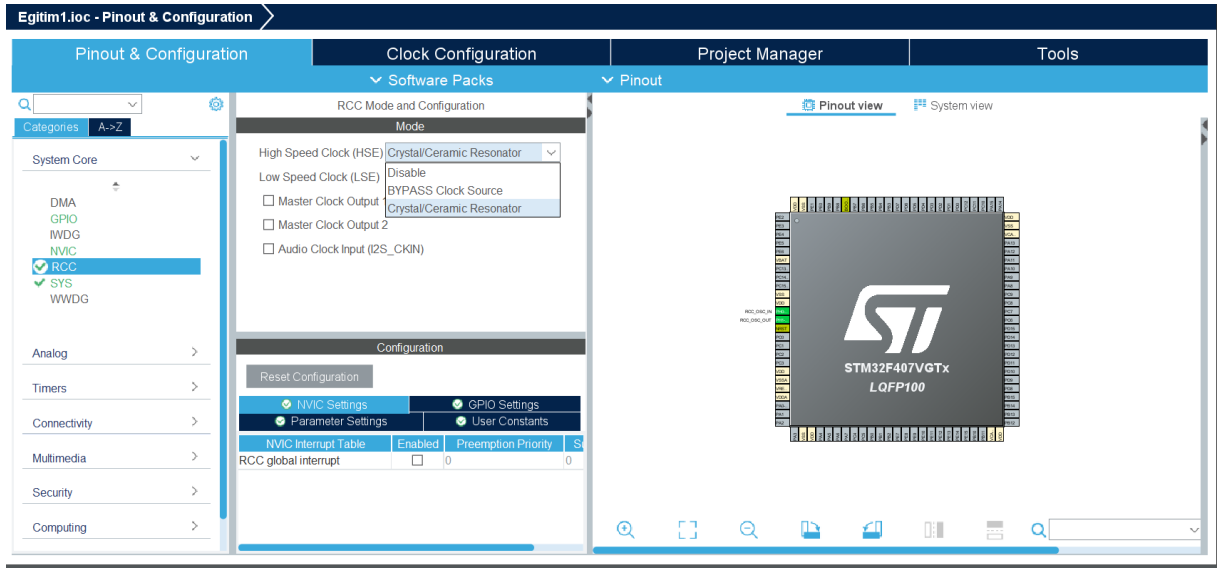


Dilersek board selector kısmından kartımızı seebiliriz. Kartı setiėimiz takdirde kart zerindeki yapılandırılmıř tm ayarlar kurulu olarak ioc ekranımıza gelir. Lakin biz projelerimizde genellikle sadece iřlemciyi seeceėiz ve sadece ihtiyacımız olan yapılandırmaları aktif hale getireceėiz.

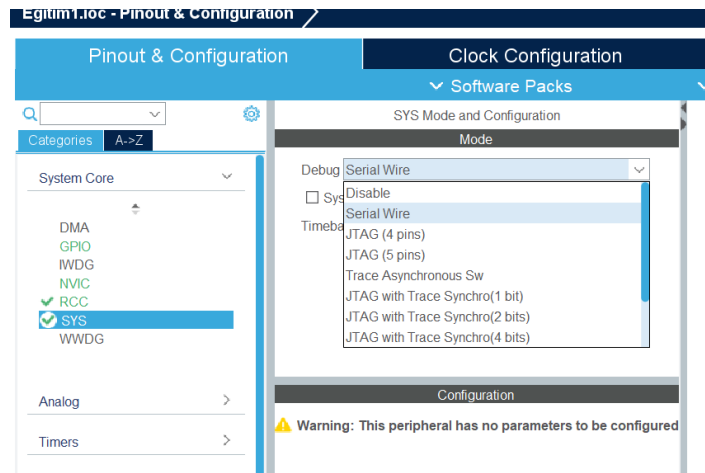
Next tuřuna bastıktan sonra Projemize isim verip finish tuřuna basıyoruz.



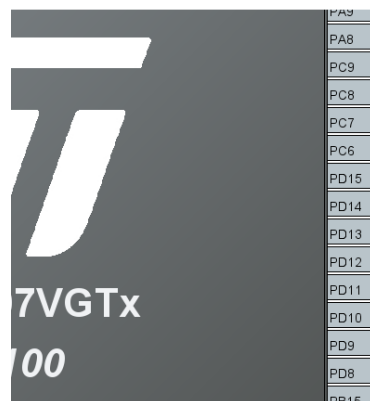
loc ekranımızda system core >> RCC>> High Speed Clock >> Crystal/Ceramic Resenator'ü seçiyoruz.



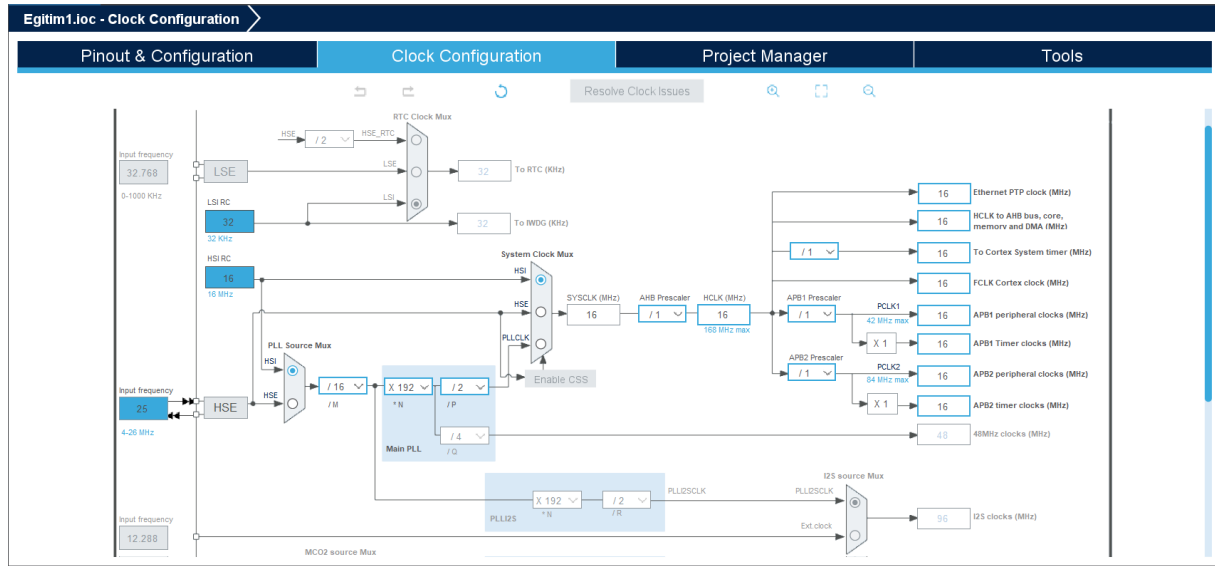
SYS>>Debug>>Serial Wire seçiyoruz.



Kart üzerindeki LD3(kırmızı), LD4(mavi), LD5(turuncu), LD6(yeşil) ledleri sırasıyla PD13, PD15, PD14, PD12 pinlerine bağlıdır.



Bu ekran clock ayarlarının yapıldığı yer. Bizi şimdilik ilgilendiren kısım en sondaki APB'lere giden sayılar. Resimde görüldüğü gibi clock, APB prescalarları ile 16MHz hızında çalışıyor.

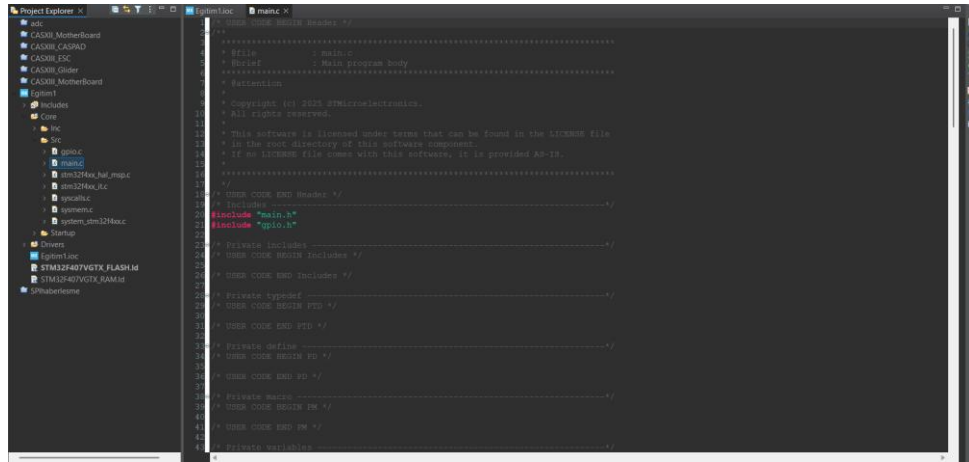


Bu kısımda Generate files kısmındaki ilk maddeyi aktif ediyoruz. Bu proje dosyalarımızın .h ve .c olarak ayrılmasını sağlıyor. Bu sayede dosyalar daha düzenli ve daha az karmaşık duruyor.

Temel olarak projemizin yapılandırması bu şekildeydi.

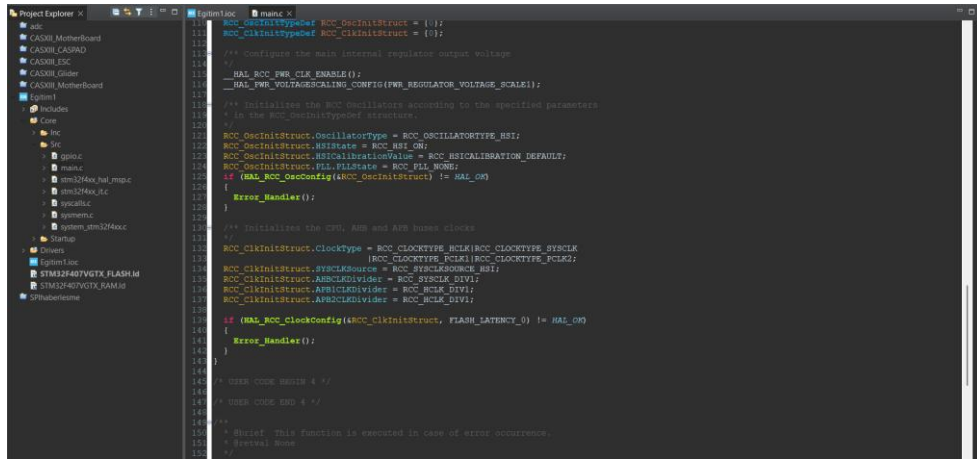
IOC ekranımızı kaydedip projemizin içindeki Core>>Src>>main.c dosyasını açıyoruz. Kodlarımızı bu dosyanın içine yazacağız. Stm otomatik olarak belirli yapılandırmaları dosyalarımıza ekliyor ve bizim kod yazabileceğimiz aralıkları oluşturuyor.

Genellikle yazacaklarımızı açıklama satırı ile belirtilmiş USER CODE aralıklarına yazacağız.



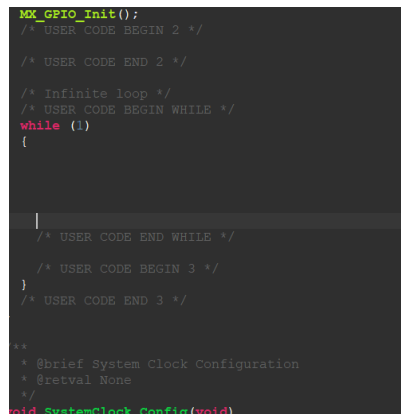
```
1  USER CODE BEGIN Header */
2
3  #ifndef MAIN_H
4  #define MAIN_H
5
6  #include "stm32f4xx_hal.h"
7
8  /* Copyright (c) 2023 STM32Microelectronics.
9  * All rights reserved.
10
11  * This software is licensed under terms that can be found in the LICENSE file
12  * in the root directory of this software component.
13  * If no LICENSE file comes with this software, it is provided AS-IS.
14
15  */
16
17  USER CODE END Header */
18
19  #include "main.h"
20  #include "gpio.h"
21
22  /* Private includes */
23  USER CODE BEGIN Includes */
24
25  USER CODE END Includes */
26
27  /* Private typedef */
28  USER CODE BEGIN PTD */
29
30  USER CODE END PTD */
31
32  /* Private define */
33  USER CODE BEGIN PD */
34
35  USER CODE END PD */
36
37  /* Private macro */
38  USER CODE BEGIN PM */
39
40  USER CODE END PM */
41
42  /* Private variables */
43
44  USER CODE BEGIN Private variables */
45
46  USER CODE END Private variables */
```

Otomatik oluşturulan satırlara örnek:



```
111  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
112  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
113
114  /** Configure the main internal regulator output voltage
115  */
116  __HAL_RCC_PWR_CLK_ENABLE();
117  __HAL_RCC_PWR_VOLTAGE_SCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
118
119  /** Initializes the RCC oscillators according to the specified parameters
120  * in the RCC_OscInitTypeDef structure.
121  */
122  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
123  RCC_OscInitStruct.HSIStrike = RCC_HSI_ON;
124  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
125  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
126  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
127  {
128      Error_Handler();
129  }
130
131  /** Initializes the CPU, APB and AHB buses clocks
132  */
133  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
134  |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
135  RCC_ClkInitStruct.PCLKSource = RCC_SYSCLKSOURCE_HSI;
136  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
137  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
138  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
139
140  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
141  {
142      Error_Handler();
143  }
144
145  USER CODE BEGIN 4 */
146
147  USER CODE END 4 */
148
149  /**
150  * Brief description of this function is provided in case of error occurrence.
151  * Details: This function is executed in case of error occurrence.
152  */
153  void Error_Handler(void)
```

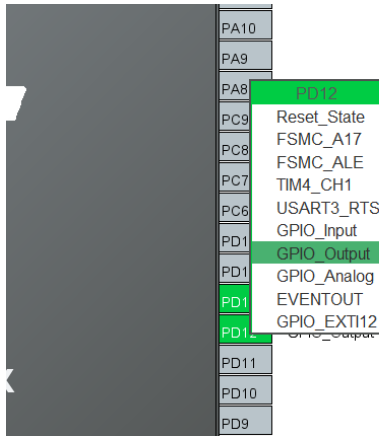
Genellikle kodumuzu döngü içinde çalıştırmak için int main(void) içindeki while aralığına yazacağız.



```
1  MX_GPIO_Init();
2  /* USER CODE BEGIN 2 */
3
4  /* USER CODE END 2 */
5
6  /* Infinite loop */
7  /* USER CODE BEGIN WHILE */
8  while (1)
9  {
10
11      /* USER CODE END WHILE */
12
13      /* USER CODE BEGIN 3 */
14  }
15  /* USER CODE END 3 */
16
17  /**
18  * @brief System Clock Configuration
19  * @retval None
20  */
21  void SystemClock_Config(void)
```

GPIO ile Led Yakma:

ioc ekranında ledlere bağlı olan pinle GPIO_Output olarak ayarlamamız gerekiyor.



Şimdilik PD12 ye bağlı Yeşil ledi (LD6) Output yapalım.

HAL Kütüphanesindeki bazı temel fonksiyonlar:

```
HAL_GPIO_WritePin(GPIOx, GPIO_PIN_y, GPIO_PIN_SET); // HIGH
HAL_GPIO_WritePin(GPIOx, GPIO_PIN_y, GPIO_PIN_RESET); // LOW
```

```
HAL_GPIO_TogglePin(GPIOx, GPIO_PIN_y);
```

```
// PD14 LED'ini yanıyorsa söndür, söndükse yak
HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_14);
```

Eğer pini input olarak okuyacaksak,

```
HAL_GPIO_ReadPin(GPIOx, GPIO_PIN_y);
```

```
if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0) == GPIO_PIN_SET) {
    // Butona basılmış
}
```

```
HAL_Delay(1000); // 1000 ms = 1 saniye bekle
```

ioc'yi kaydedip main.c ye geliyoruz ve while kısmına kodumuzu yazıyoruz.

Yeşil ledi yarım saniyede bir yakıp söndüren kod.

```
/* USER CODE END 2 */

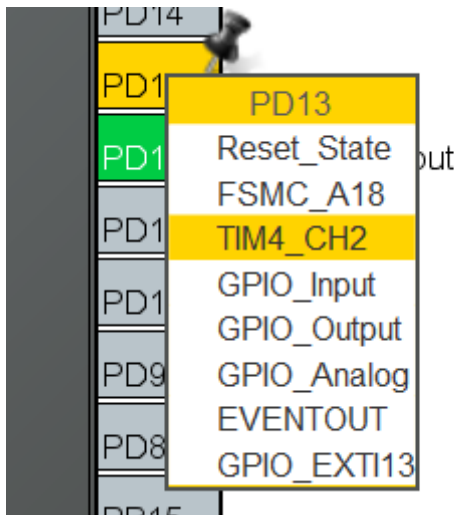
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_SET);
    HAL_Delay(500);
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
    HAL_Delay(500);

    /* USER CODE END WHILE */

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

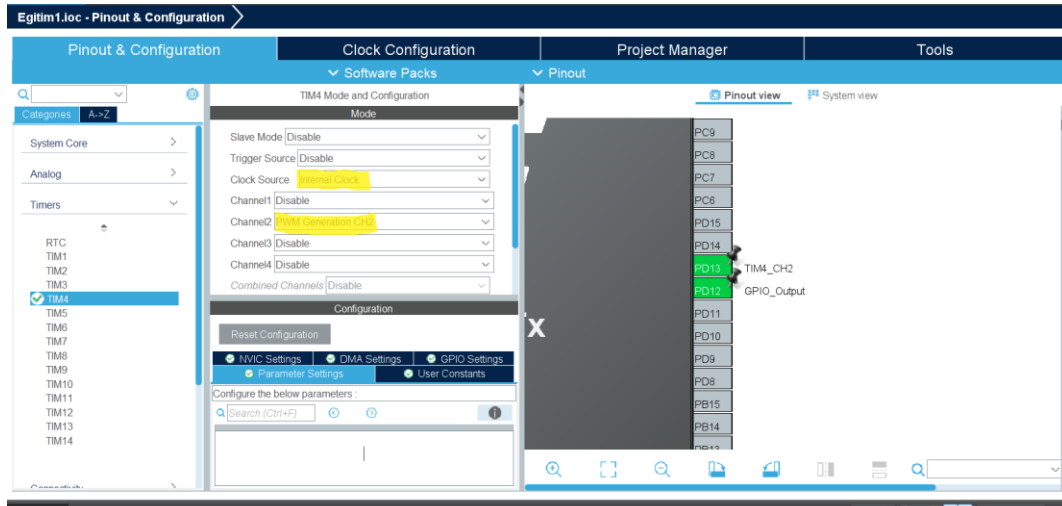
Run tuşuna basıp çalıştırıyoruz.

Timer ile PWM Verme ve Led Yakma



PD13 Timer 4'ün 2. Kanalına bağlıymış. Seçiyoruz.

Timers>>TIM4>>Clock Source ve Channel ayarlarını resimdeki gibi yapıyoruz.



Hatırlarsanız clock'umuz 16MHz çalışıyordu. Bizim buradaki amacımız bölücülerle clock hızını bölerek timer hızını 50 Hz yapmak.

Burada önemli olan 2 ayar var:

Counter Period: Bu ayar bizim pwm sinyalinin çözünürlüğü. Örneğin bu ayarı 1999 yaptığımızda 0 – 1999 aralığında pwm sinyali gönderebileceğiz.

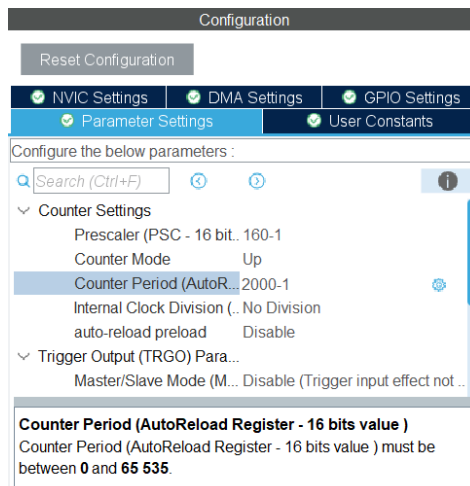
Prescaler: Bu bizim bölücümüz ve clock hızını bölmemize yarıyor.

Daha anlaşılabilir olması bakımından:

16MHz yani 16 000 000 Hz sinyali, Prescaler 160 ve Counter Period 2000 ile bölüp 50Hz sinyale ulaşıyoruz.

$$16\,000\,000 / 160 / 2000 = 50\text{ Hz}$$

Bu ayarları belirlerken saymaya sıfırdan başladığımızdan dolayı Prescaler'a 160 - 1 veya 159 ; Counter Period'a 2000 – 1 veya 1999 yazmalıyız.



ioc'yi kaydedip main.c ye geliyoruz.

PWM vermek için kullandığımız HAL kütüphanesindeki fonksiyonlar:

```
HAL_TIM_PWM_Start(&htimX, TIM_CHANNEL_Y);
```

```
__HAL_TIM_SET_COMPARE(&htimX, TIM_CHANNEL_Y, compare_value);
```

Turuncu ledi 1500/2000 yani %75 duty cycle ile yakan kod:

```
/* USER CODE BEGIN 2 */
    HAL_TIM_PWM_Start(&htim4,TIM_CHANNEL_2);
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    __HAL_TIM_SET_COMPARE(&htim4,TIM_CHANNEL_2,1500);|

    /* USER CODE END WHILE */

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

“User Code 2” bloğunun içinde HAL_TIM_PWM_Start fonksiyonu ile pwm sinyalini başlattık.

__HAL_TIM_SET_COMPARE fonksiyonu ile istediğimiz duty cycle değerinde PWM sinyalini gönderdik.

Çözünürlüğümüz 2000. 1500 verdiğimiz zaman %75 duty cycle değerine ulaştık.

NOT: Bazı motorlar belirli duty cycle aralıklarında çalışabilir. Örneğin bir servo motor %10 ve %70 duty cycle aralığında çalışabilir. %10 negatif yönde maximum açı, %70 pozitif yönde maximum açı, %40 ise başlangıç (sıfır) noktası olabilir.

Hazırlayanlar:

Yavuz Eren Doğan

Celal Yeldus