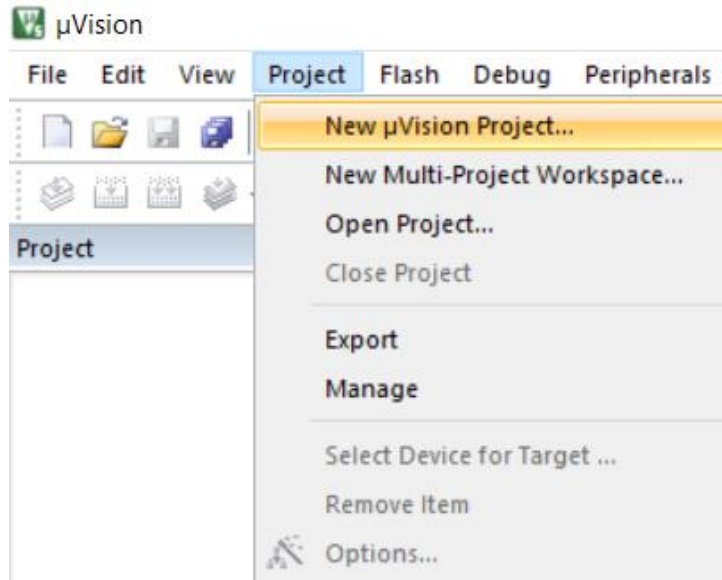


## Keil + GL Starter Kit

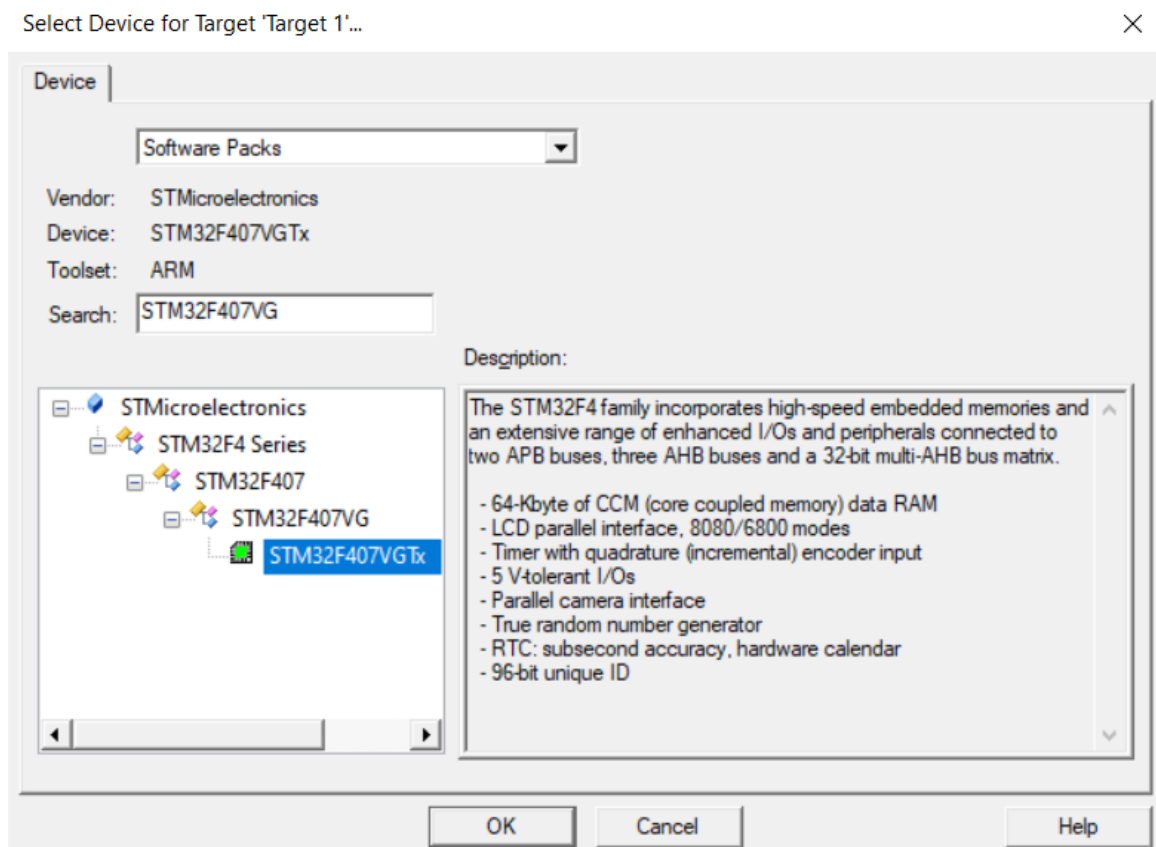
### Getting started

#### 1. Создание нового проекта

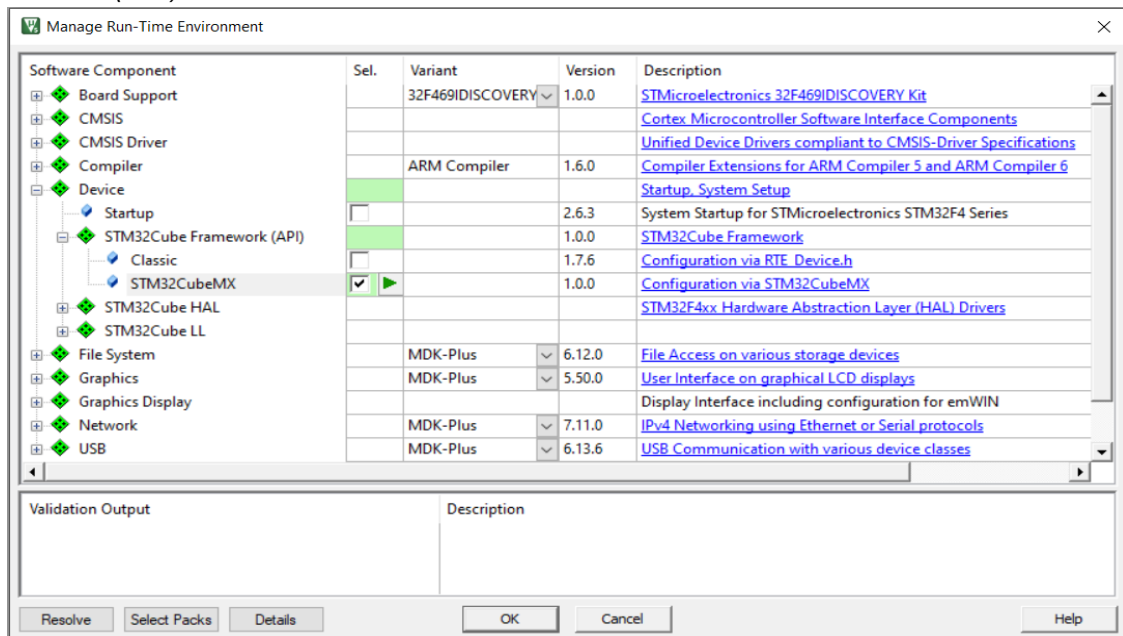
В Keil uVision5 *Project -> New uVision Project-> Ввести имя проекта (рекомендуется создать для него отдельную директорию)*



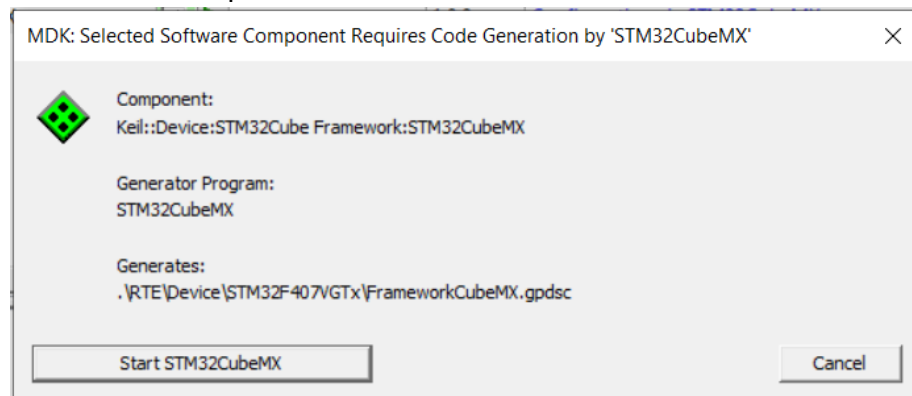
В *Select Device for Target* во вкладке *Search* выбрать *STM32F407VG*



В *Manage Run - Time Environment* во вкладке *Device* выбрать *STM32Cube Framework(API)* -> *STM32CubeMX*



В появившемся окне выбираем *Start STM32CubeMX*.



CubeMX будет иметь следующий вид



## 2. Обзор CubeMX

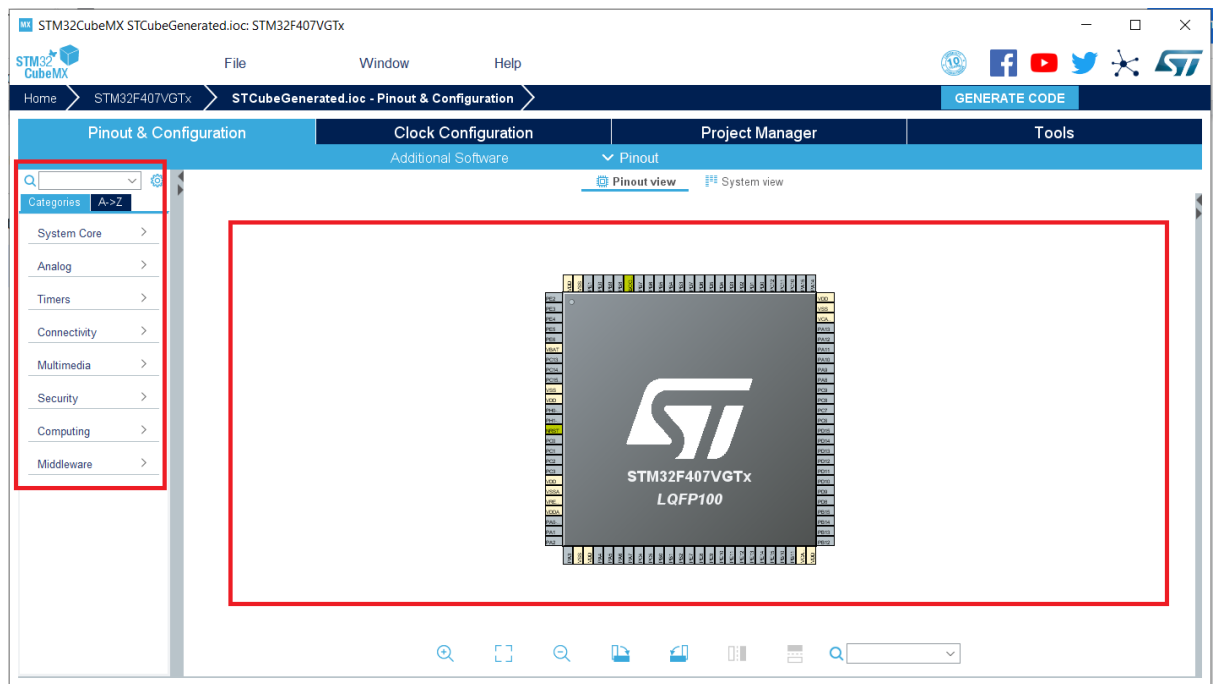
CubeMX состоит из четырех вкладок:

- *Pinout & Configuration* - настройка портов ввода/вывода и периферии;
- *Clock Configuration* - настройка тактирования - выбор тактового генератора, настройка PLL, частот шин, периферии;
- *Project Manager* - можно задать минимальный размер stack/heap, используемую версию Firmware Package, а также опции кодогенерации.
- *Tools* - можно приблизительно рассчитать потребление тока контроллером.

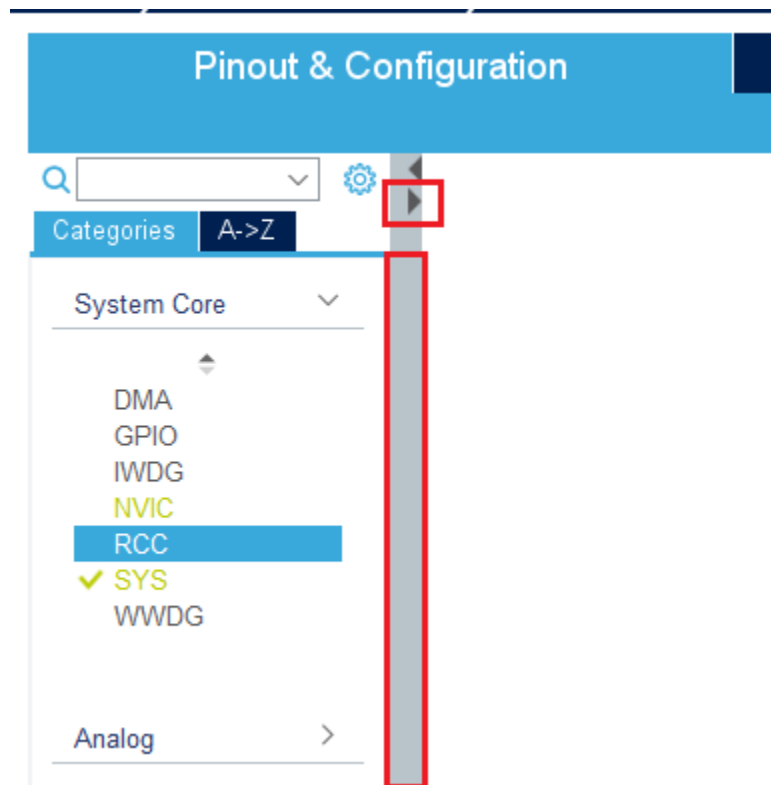
## 2.1 Pinout & Configuration

По умолчанию окно разделено на две части:

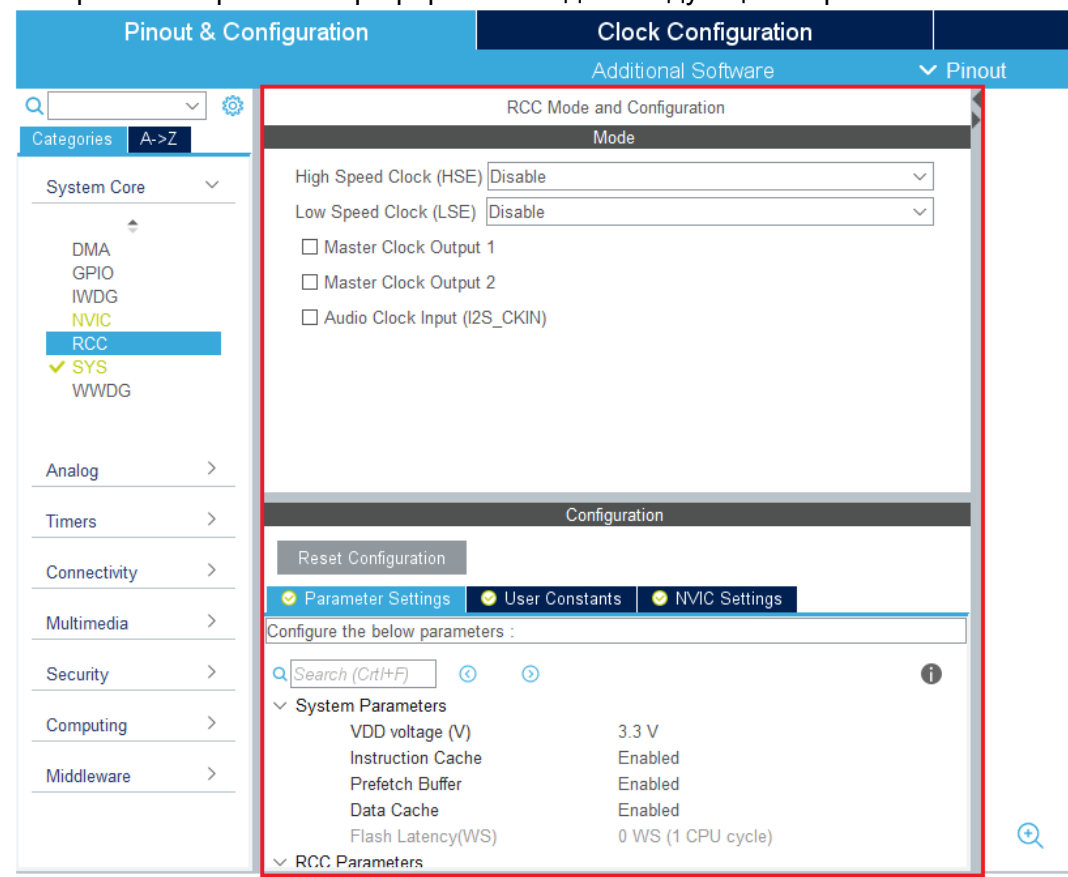
- *Peripherals Configuration* - периферия контроллера, сгруппированная по категориям (*Categories*). Также можно переключить список на отображение в алфавитном порядке (*A->Z*);
- *Pinout view* - отображение контроллера в выбранном корпусе (LQFP100) со всеми пинами (портами ввода/вывода).



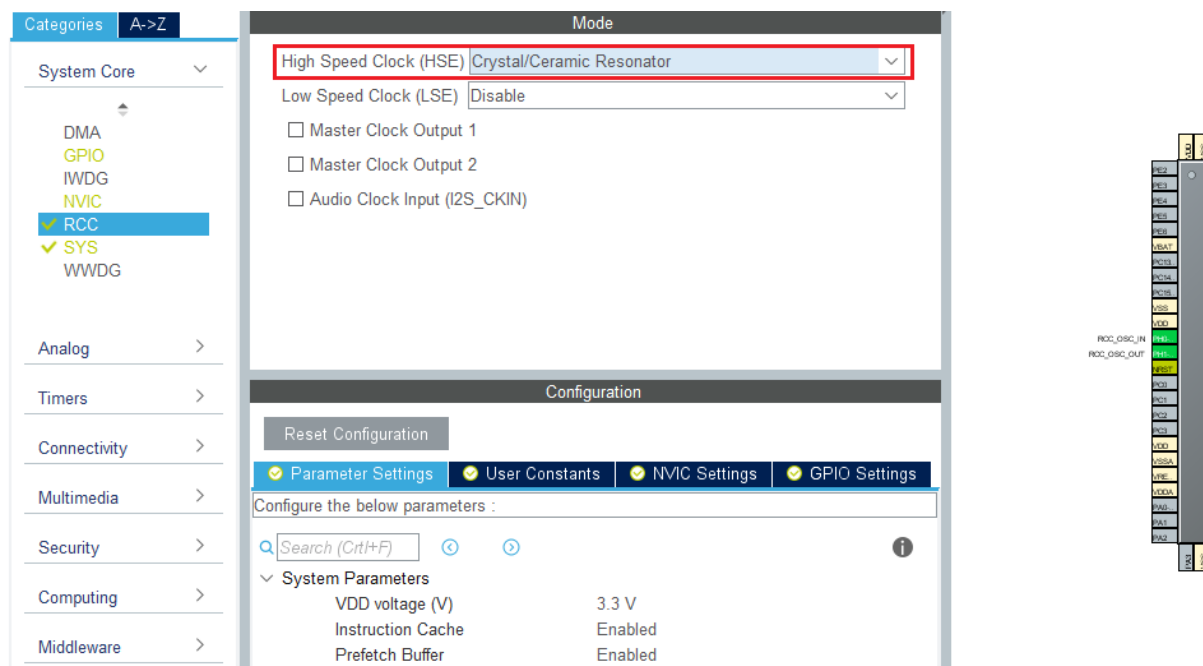
После включения/сброса MCU тактируется от внутреннего RC-генератора (HSI - High Speed Internal oscillator), который не отличается высокой стабильностью частоты. StarterKit имеет внешний кварц на 8 МГц. Переключим источник тактовых импульсов на HSE (High Speed External oscillator) - внешний кварцевый резонатор. В *Peripherals Configuration* выберем *System Core* -> *RCC*. По умолчанию окно настройки периферии скрыто. Для того, чтобы его открыть, нужно кликнуть на вертикальную полосу-разделитель между окнами *Peripherals Configuration* и *Pinout view* либо нажать на треугольник на той же полосе-разделителе



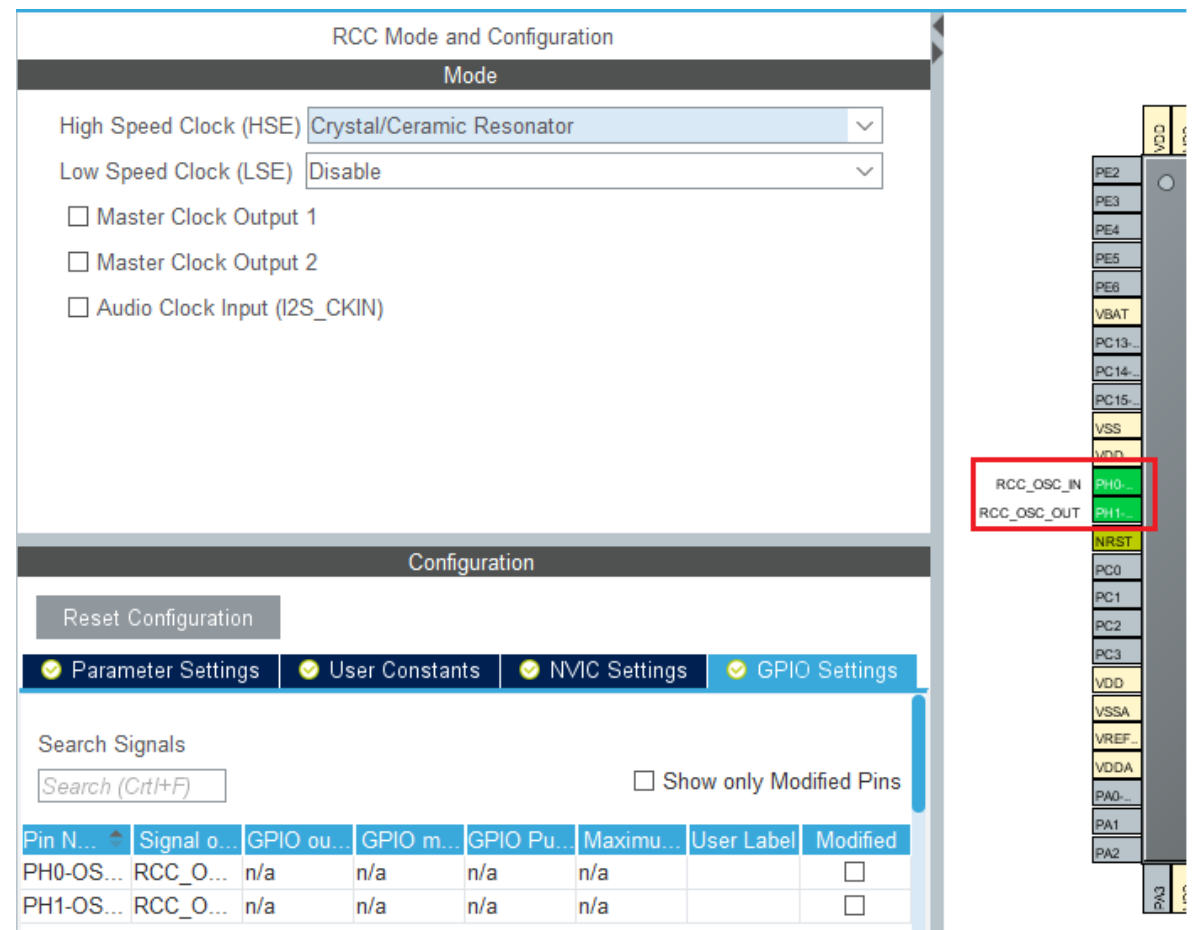
Окно настройки выбранной периферии выглядит следующим образом



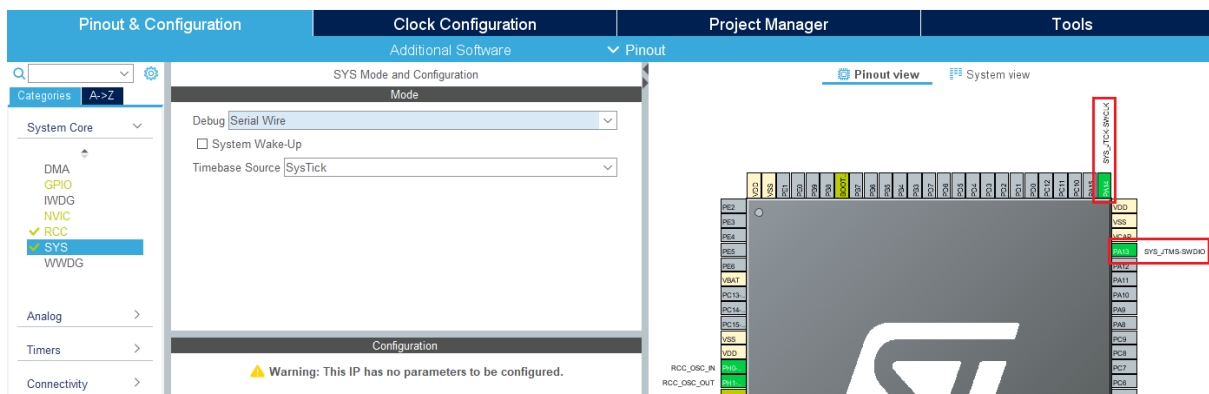
Включим тактирование от HSE. Для этого в выпадающем списке *High Speed Clock (HSE)* выберем *Crystal/Ceramic Resonator*



После этих действий в *Pinout View* пины PH0 и PH1 (OSC\_IN и OSC\_OUT соответственно) стали зелеными т.е. эти пины стали задействованы



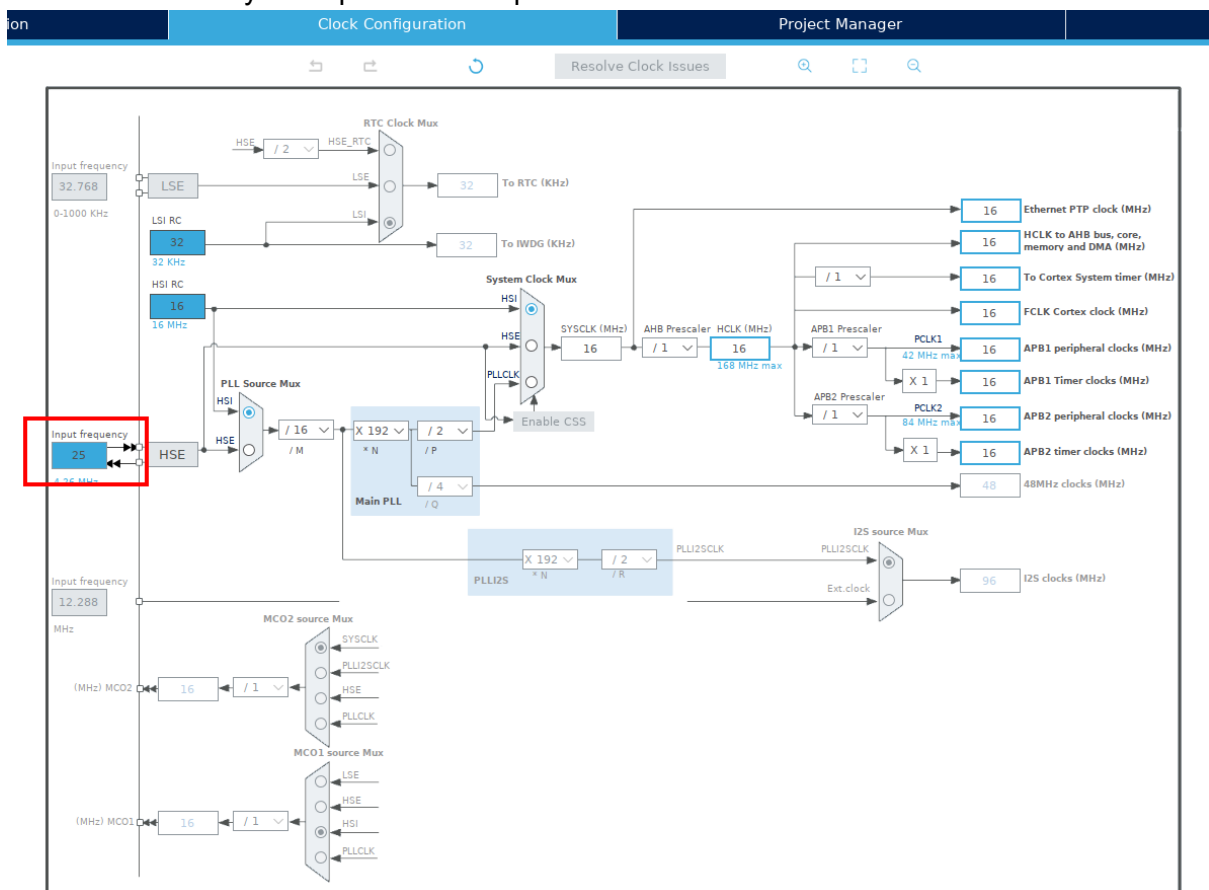
Далее необходимо разрешить отладку по SWD (последовательный отладочный интерфейс). Для этого в *Peripherals Configuration* переходим в категорию SYS. В выпадающем списке *Debug* выбираем *Serial Wire*



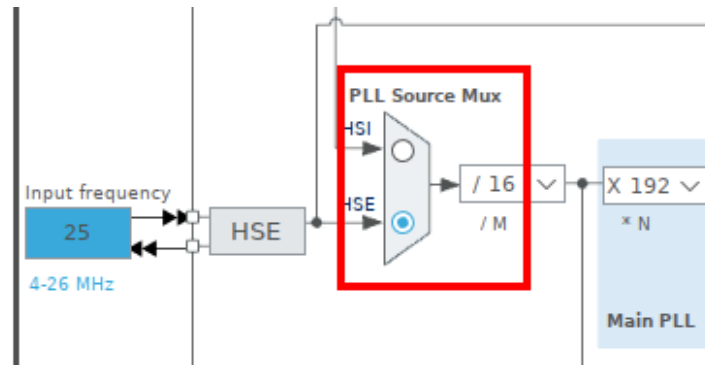
Так же видно, что в *Pinout View* пины PA13 (SWDIO) и PA14 (SWCLK) стали задействованы

### 3.1. Clock Configuration

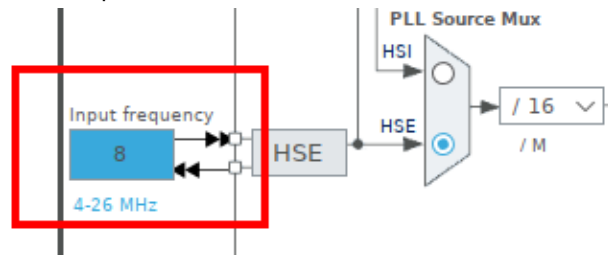
Показывает схему тактирования выбранного MCU



Так как мы включили тактирование от HSE, то блок *HSE* стал активным. Но источником тактовых импульсов по-прежнему является HSI. Для переключения на HSE в блоке *PLL Source Mux* выберем HSE (просто кликнуть на “кружочек” напротив HSE)



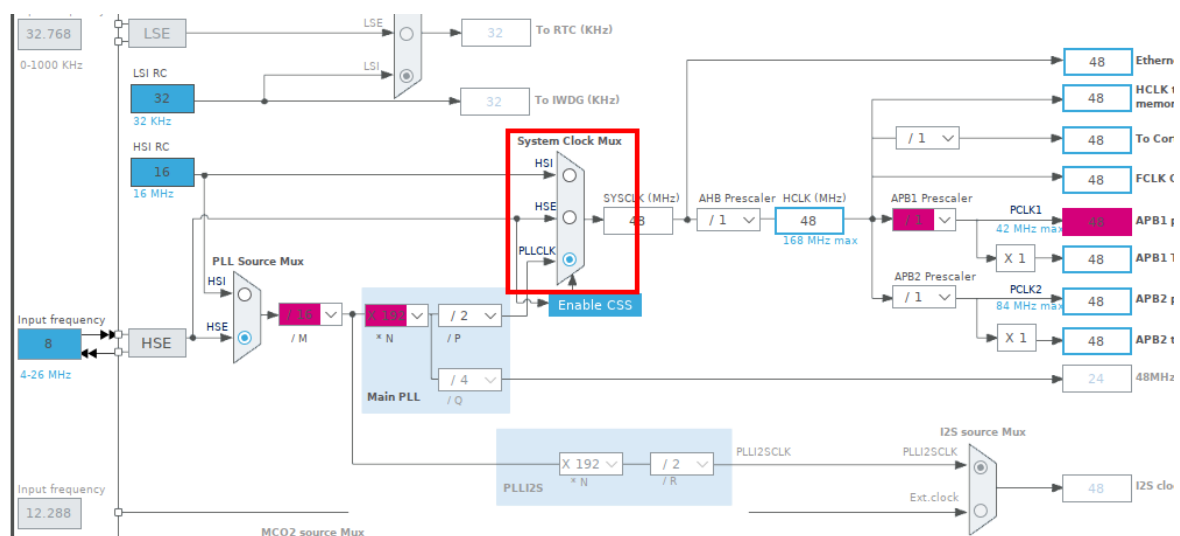
Как упоминалось ранее, на StarterKit стоит кварц на 8 МГц. Исправим *Input frequency*, т.к. по умолчанию стоит 25 МГц



MCU имеет 3 шины:

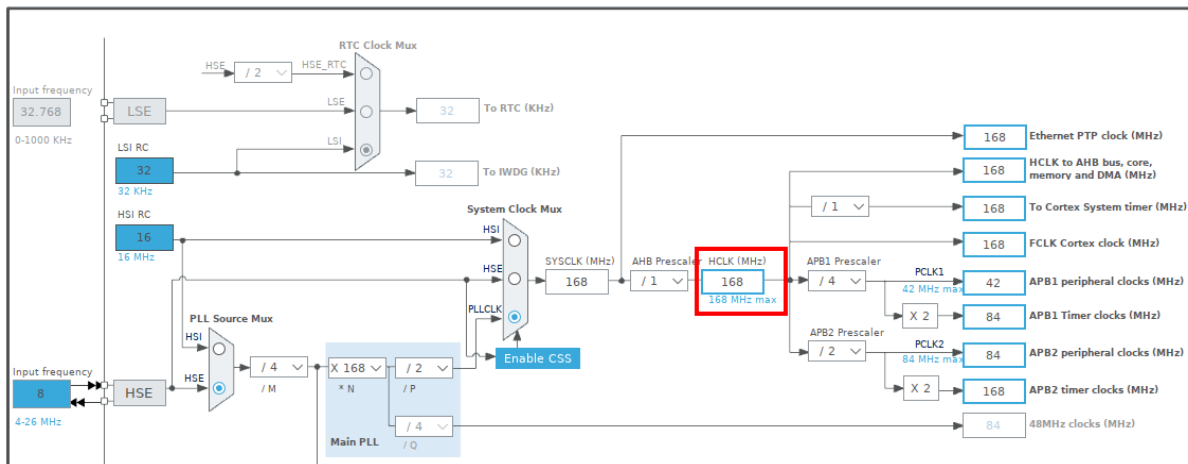
- АНВ - Advanced High-performance Bus. На этой шине работает ядро (core), DMA, память
- APB1/APB2 - Advanced Peripheral Bus. На этих шинах работают таймеры и почти вся периферия.

Максимальная частота АНВ - 168 МГц. Для повышения частоты тактового генератора (HSE = 8 МГц) задействуем PLL (Phase-Locked Loop). Для этого *System Clock Mux* выберем *PLLCLK*.



Красные значения делителей и частот означают, что частоты, после выбранных коэффициентов деления получились либо ниже допустимого предела, либо выше.

Коэффициенты деления можно подобрать вручную. А можно эту задачу передать *Clock Configuration*. Так, например, если мы хотим получить частоту AHB = 168МГц, можно в *HCLK* ввести *168* и нажать *Enter*. Коэффициенты деления будут пересчитаны автоматически



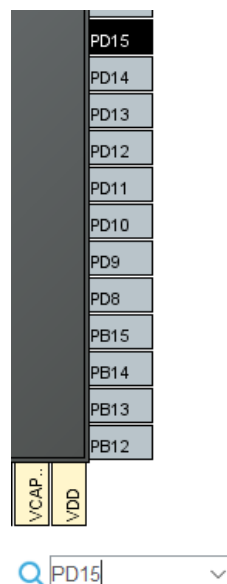
Настройка тактирования окончена. Теперь попробуем выполнить первую задачу - помигать светодиодом.

#### 4. Мигание светодиодом

StarterKit имеет 4 светодиода:

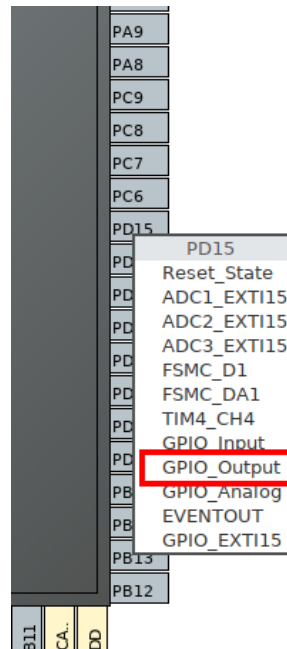
- PD12 - зеленый;
- PD13 - оранжевый;
- PD14 - красный;
- PD15 - синий.

Помигаем, к примеру, синим светодиодом. Для этого нужно настроить пин #15 порта PORTD (PD15). Переходим на вкладку *Pinout & Configuration*. В *Pinout view* нужно найти пин PD15. Для этого можно воспользоваться поиском. В окне поиска введем *PD15*

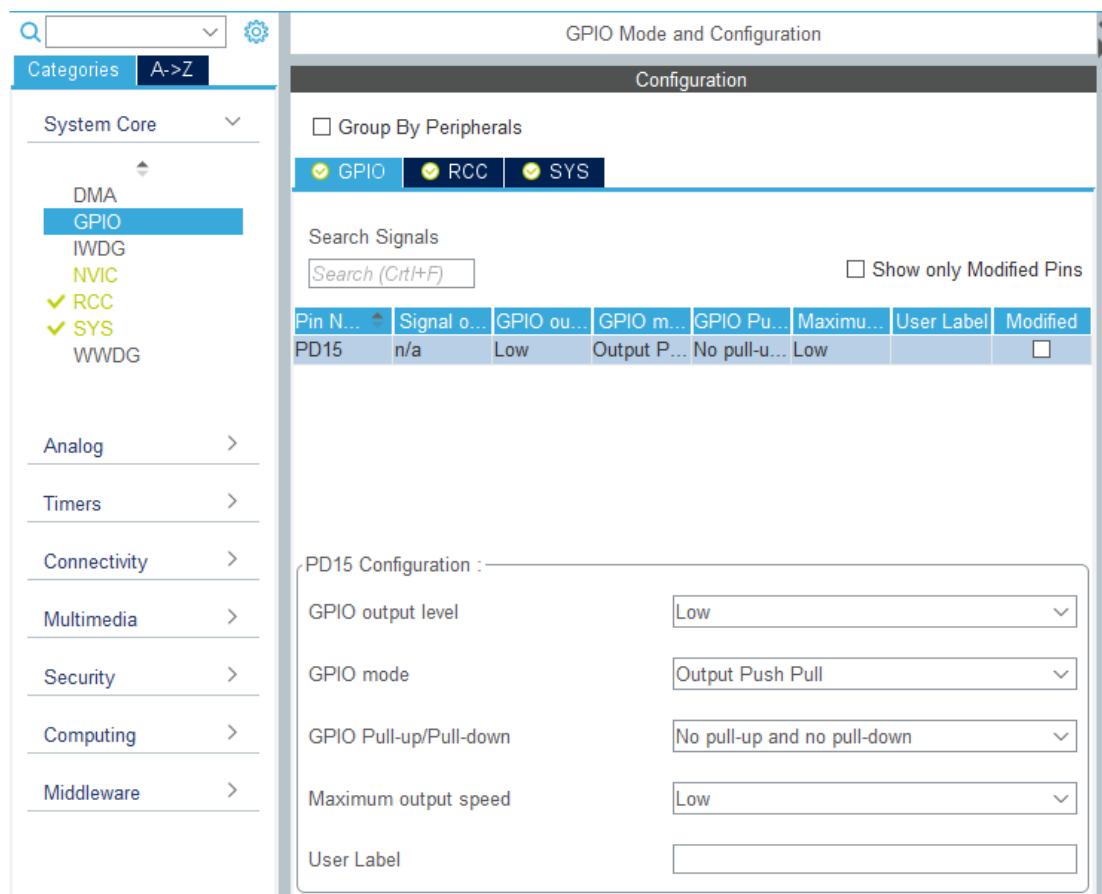


Для управления светодиодом пин PD15 должен быть настроен на выход. Для этого в *Pinout view* кликнем на *PD15* и из списка выберем *GPIO\_Output*

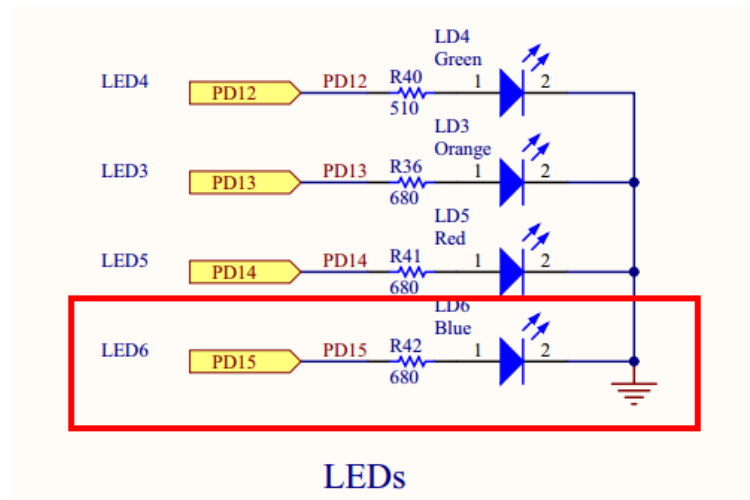




Далее переходим в *Peripheral Configuration* в категорию *System Core* в раздел *GPIO*. В списке появился PD15. После того, как мы выделим этот пин в списке, у нас появятся поля для настройки.



По схеме синий светодиод, подключенный к PD15, не содержит подтягивающего резистора



Поэтому режим работы пина (GPIO mode) должен быть *push-pull*

PD15 Configuration :

GPIO output level	Low
GPIO mode	Output Push Pull
GPIO Pull-up/Pull-down	No pull-up and no pull-down
Maximum output speed	Low
User Label	

Остальные настройки можно оставить без изменений.

Теперь все готово для генерации кода. Для этого в *Device Configuration Tool* перейдем на вкладку *Project Manager* в *Code Generator* и поставим галочку напротив *Keep User Code when re-generating*

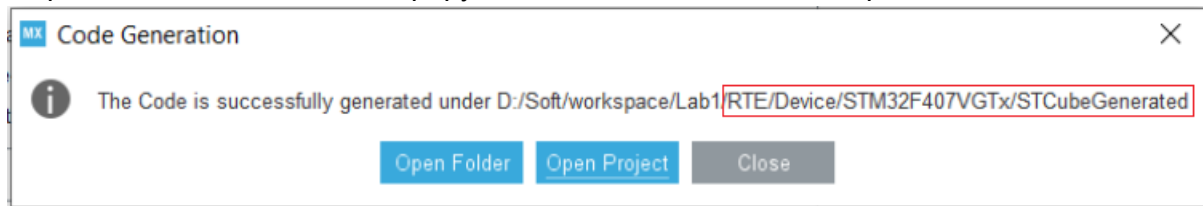
Pinout & Configuration	Clock Configuration	Project Manager
<p>STM32Cube MCU packages and embedded software packs</p> <p><input type="radio"/> Copy all used libraries into the project folder</p> <p><input checked="" type="radio"/> Copy only the necessary library files</p> <p><input type="radio"/> Add necessary library files as reference in the toolchain project configuration file</p>		
<p>Generated files</p> <p><input type="checkbox"/> Generate peripheral initialization as a pair of '.c/.h' files per peripheral</p> <p><input type="checkbox"/> Backup previously generated files when re-generating</p> <p><input checked="" type="checkbox"/> Keep User Code when re-generating</p> <p><input checked="" type="checkbox"/> Delete previously generated files when not re-generated</p>		
<p>HAL Settings</p> <p><input type="checkbox"/> Set all free pins as analog (to optimize the power consumption)</p> <p><input type="checkbox"/> Enable Full Assert</p>		
<p>Template Settings</p> <p>Select a template to generate customized code</p> <p style="text-align: right;"><a href="#">Settings...</a></p>		

Остальные настройки - по своему усмотрению.

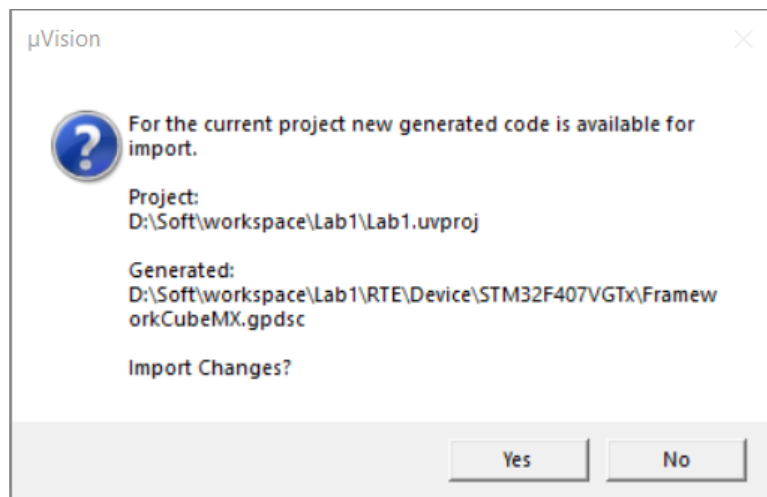
Для генерации кода нажмите **GENERATE CODE**



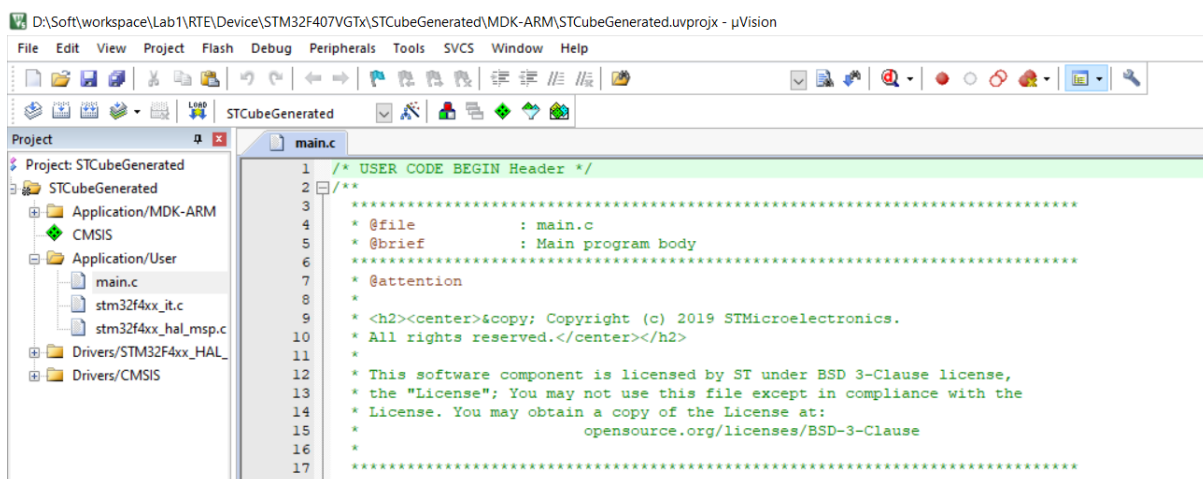
Обратите внимание, код генерируется в подкаталоге вашего проекта.



Нажмите *Open Project* чтобы перейти в этот подкаталог. uVision предложит портировать сгенерированный код в ваш изначальный каталог, однако, будут портированы не все зависимости, поэтому проще продолжить работу в созданном при генерации подкаталоге.



В *Project* откроем файл *Application/User/main.c*



Инициализация *PD15* выполняется в функции *static void MX\_GPIO\_Init(void)*

```
main.c
151  */
152  static void MX_GPIO_Init(void)
153  {
154      GPIO_InitTypeDef GPIO_InitStructure = {0};
155
156      /* GPIO Ports Clock Enable */
157      __HAL_RCC_GPIOH_CLK_ENABLE();
158      __HAL_RCC_GPIOD_CLK_ENABLE();
159      __HAL_RCC_GPIOA_CLK_ENABLE();
160
161      /*Configure GPIO pin Output Level */
162      HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);
163
164      /*Configure GPIO pin : PD15 */
165      GPIO_InitStructure.Pin = GPIO_PIN_15;
166      GPIO_InitStructure.Mode = GPIO_MODE_OUTPUT_PP;
167      GPIO_InitStructure.Pull = GPIO_NOPULL;
168      GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_LOW;
169      HAL_GPIO_Init(GPIOD, &GPIO_InitStructure);
170
171  }
```

Теперь в цикле *while* добавим код для мигания светодиодом

```
88  /* Initialize all configured peripherals */
89  MX_GPIO_Init();
90  /* USER CODE BEGIN 2 */
91
92  /* USER CODE END 2 */
93
94  /* Infinite loop */
95  /* USER CODE BEGIN WHILE */
96  while (1)
97  {
98      HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_15);
99      HAL_Delay(500);
100  /* USER CODE END WHILE */
101
102  /* USER CODE BEGIN 3 */
103  }
104  /* USER CODE END 3 */
105  }
```

```
HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_15);
HAL_Delay(500);
```

**ВАЖНО:** для того, чтобы написанный код сохранялся после регенераций кода, он должен находиться между подобными комментариями

```
/* USER CODE BEGIN 2 */
```

```
/* USER CODE END 2 */
```

Собираем проект (*Project -> Build Target*). Если проект собрался успешно, то в *Console* должен быть следующий вывод

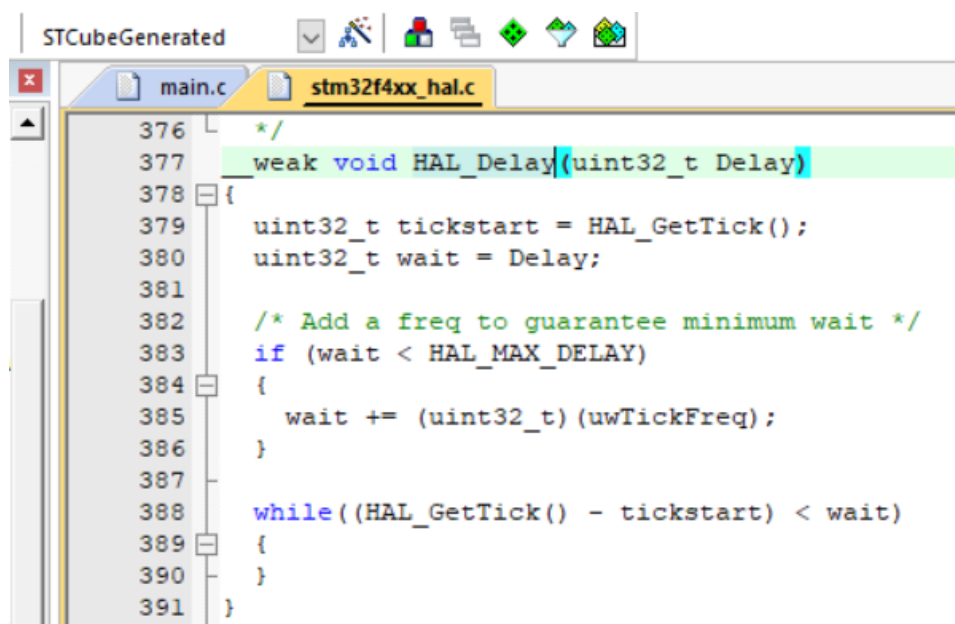
```
Build Output
compiling stm32f4xx_hal_flash_ex.c...
compiling stm32f4xx_hal_pwr_ex.c...
compiling stm32f4xx_hal_flash_ramfunc.c...
compiling stm32f4xx_hal_dma_ex.c...
compiling stm32f4xx_hal_cortex.c...
compiling system_stm32f4xx.c...
compiling stm32f4xx_hal_exti.c...
compiling stm32f4xx_hal.c...
linking...
Program Size: Code=2764 RO-data=440 RW-data=16 ZI-data=1024
FromELF: creating hex file...
"STCubeGenerated\STCubeGenerated.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:15
<
```

## 5. Мигание светодиодом, используя таймер

Несмотря на то, что в предыдущем примере мы смогли помигать светодиодом, в нашей программе есть существенный недостаток - блокирующая задержка. Для того, чтобы частота мигания светодиода была различима глазу, в предыдущем примере мы добавили задержку в 500 мс после каждой смены состояния выхода *PD15*:

```
HAL_Delay(500);
```

Если мы посмотрим на реализацию функции `__weak void HAL_Delay(uint32_t Delay)` мы увидим, что она является блокирующей:



```
STCubeGenerated
main.c  stm32f4xx_hal.c
376  /*
377  __weak void HAL_Delay(uint32_t Delay)
378  {
379      uint32_t tickstart = HAL_GetTick();
380      uint32_t wait = Delay;
381
382      /* Add a freq to guarantee minimum wait */
383      if (wait < HAL_MAX_DELAY)
384      {
385          wait += (uint32_t)(uwTickFreq);
386      }
387
388      while((HAL_GetTick() - tickstart) < wait)
389      {
390      }
391  }
```

Все 500 мс контроллер только проверяет условие в цикле:

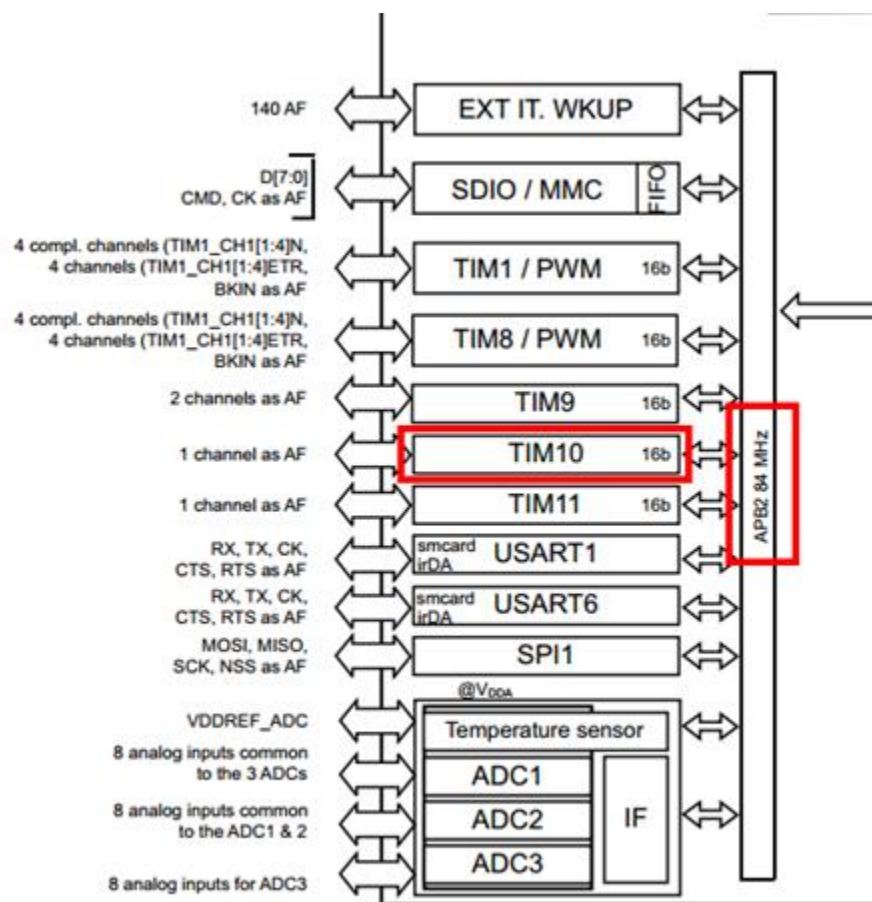
```

while((HAL_GetTick() - tickstart) < wait)
{
}

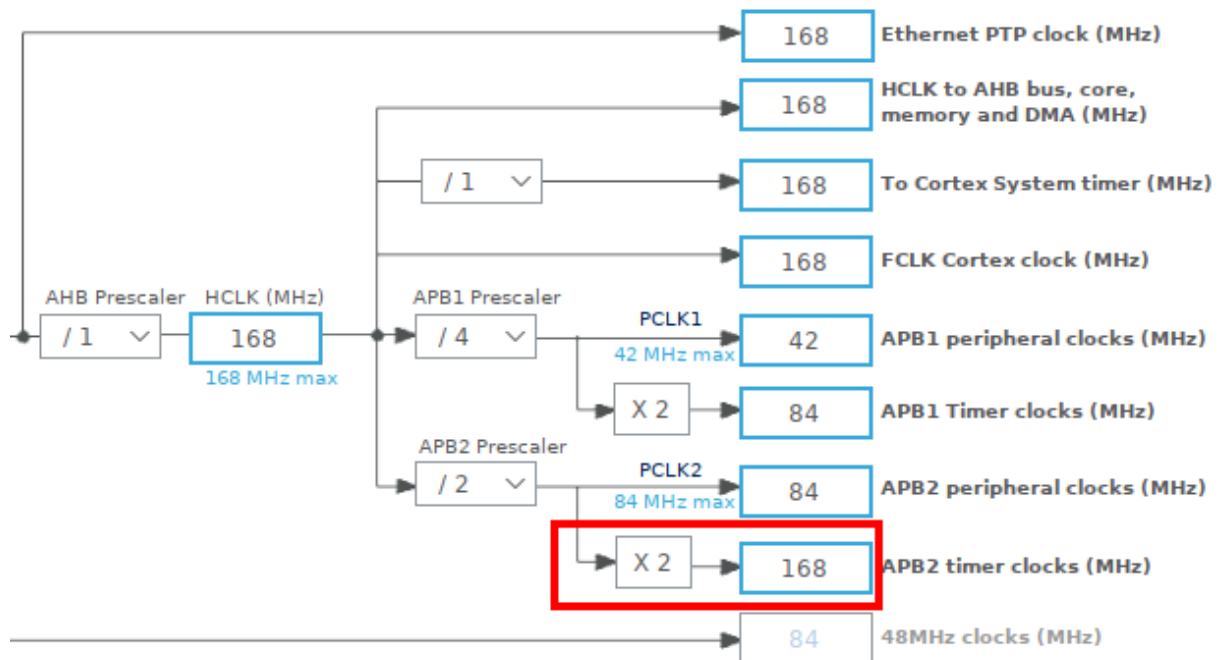
```

Исправим это. Сделаем так, чтобы состояние пина *PD15* менялось в прерывании таймера. Все остальное время контроллер может быть занят другими делами.

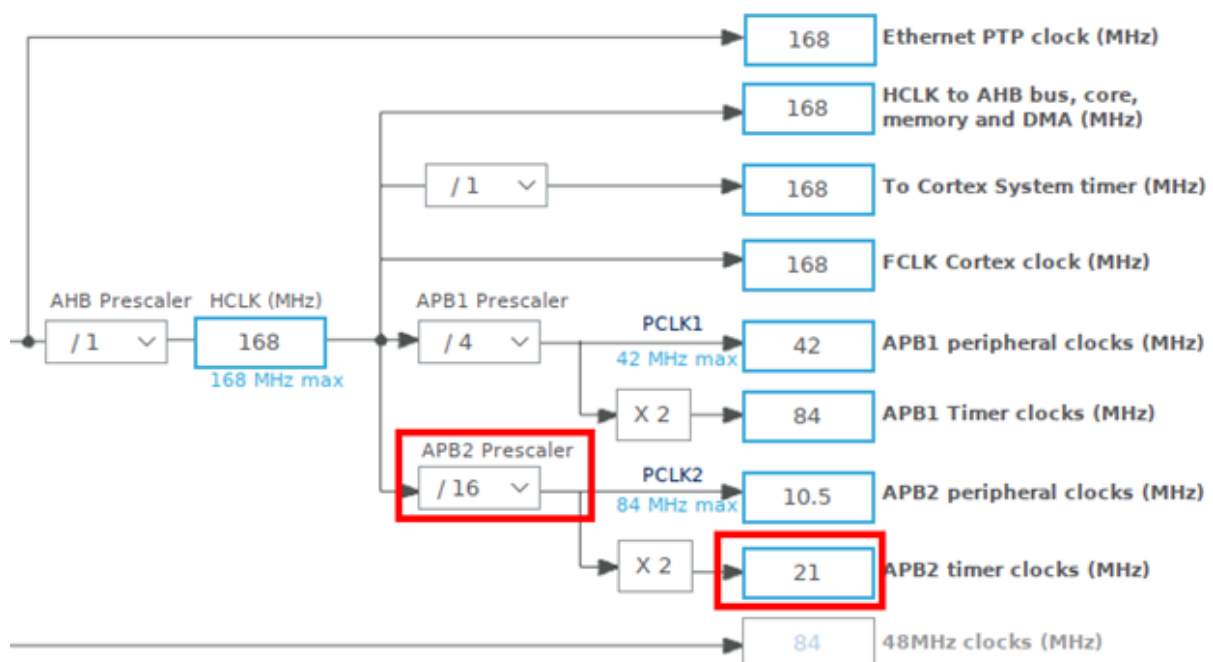
Для нашей задачи вполне подойдет самый простой таймер - general-purpose timer - к примеру, *TIM10*. Для начала нужно определить к какой шине подключен *TIM10*. Для этого откроем datasheet (именно datasheet, а не reference manual), раздел *Device overview*, *STM32F40xxx block diagram*



Из диаграммы видно, что *TIM10* подключен к шине *APB2*. Перейдем на вкладку *Clock Configuration* в *STM32CubeIDE*. Частота шины *APB2* = 84 МГц, но частота тактирования таймеров шины *APB2* еще умножается на 2, поэтому таймеры этой шины будут работать на частоте 168 МГц

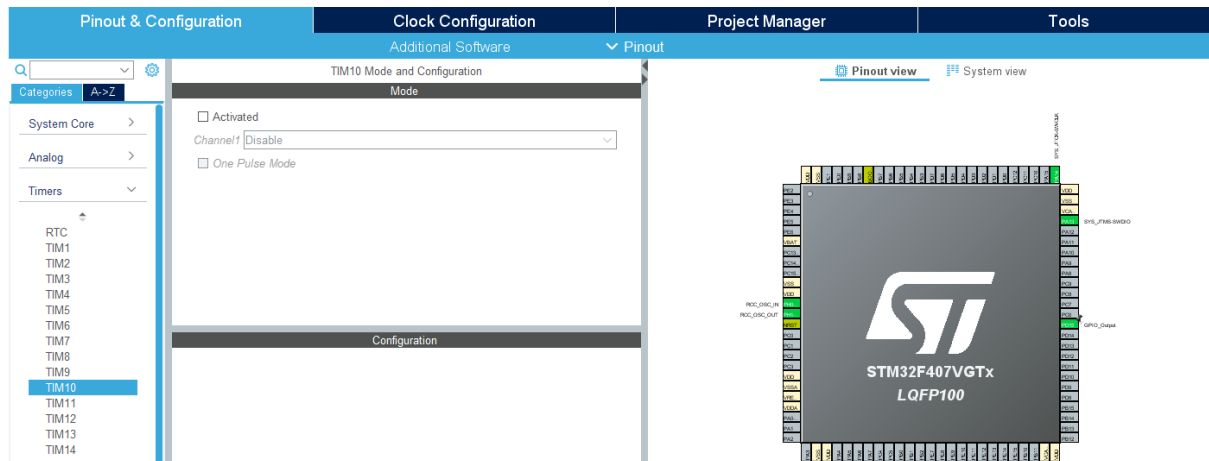


Получаем, что один тик  $TIM10 = 1/168000000 \text{ Гц} = 0.00000000595 \text{ с} = 0.00595 \text{ мкс}$ . Мы хотим, чтобы пин  $PD15$  менял состояние каждые 500 мс. Нет необходимости, чтобы  $TIM10$  работал на такой большой частоте. К тому же, мы пока не используем никакую периферию, подключенную к шине  $APB2$ . Поэтому мы можем уменьшить частоту  $APB2$  увеличив значение делителя (prescaler) этой шины. Установим максимально возможный делитель = 16

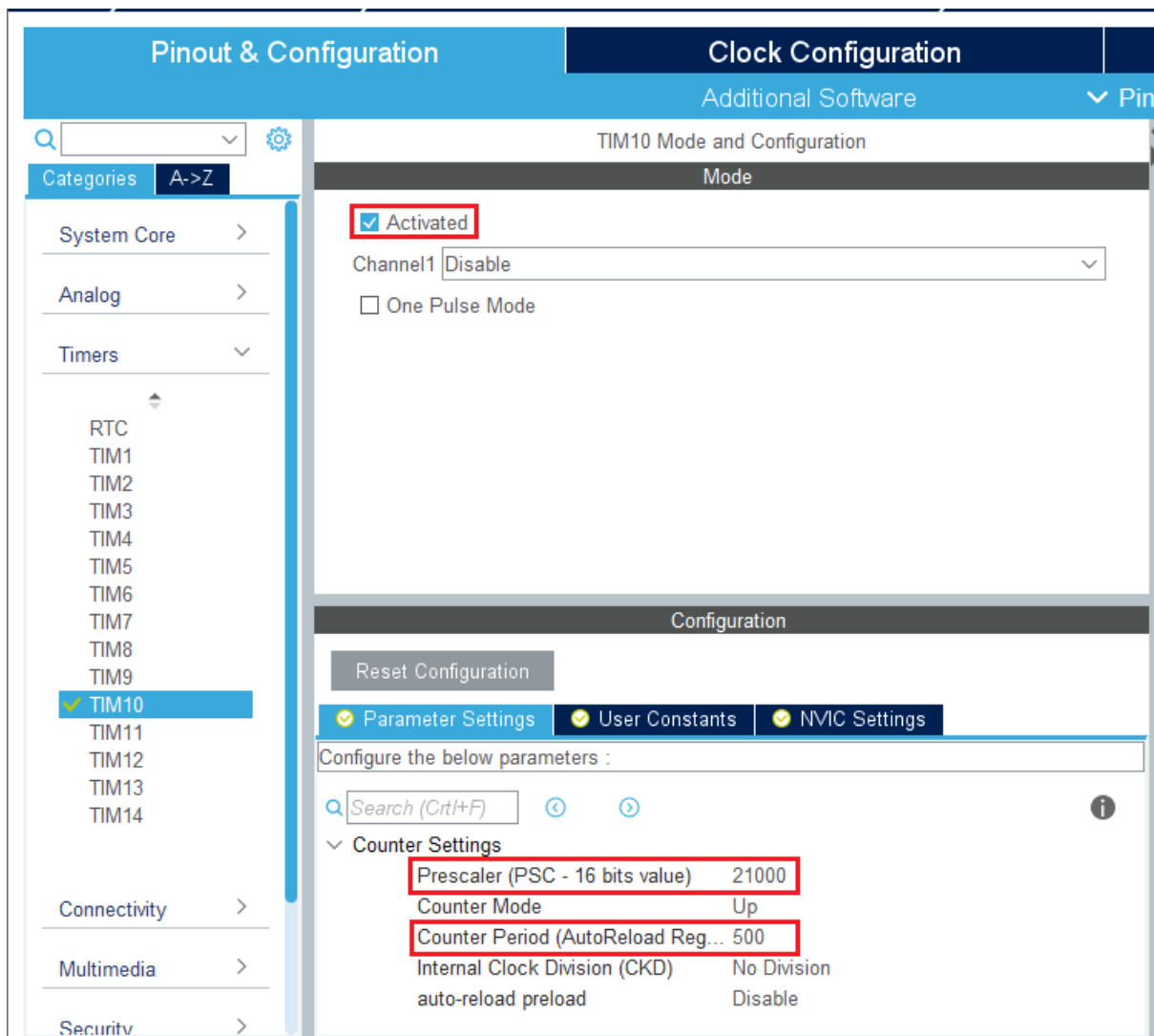


Теперь  $TIM10$  будет работать на частоте 21 МГц. Следовательно, один тик  $TIM10 = 1/21000000 \text{ Гц} = 0.0000000476 \text{ с} = 0.0476 \text{ мкс}$ .

В CubeMX перейдем на вкладку *Pinout & Configurations*. В *Categories* -> *Timers* выберем *TIM10* (если вы закрыли CubeMX, найдите в проекте файл с расширением *STM32CubeMX (.ioc)* и откройте его).



Настроим *TIM10* следующим образом

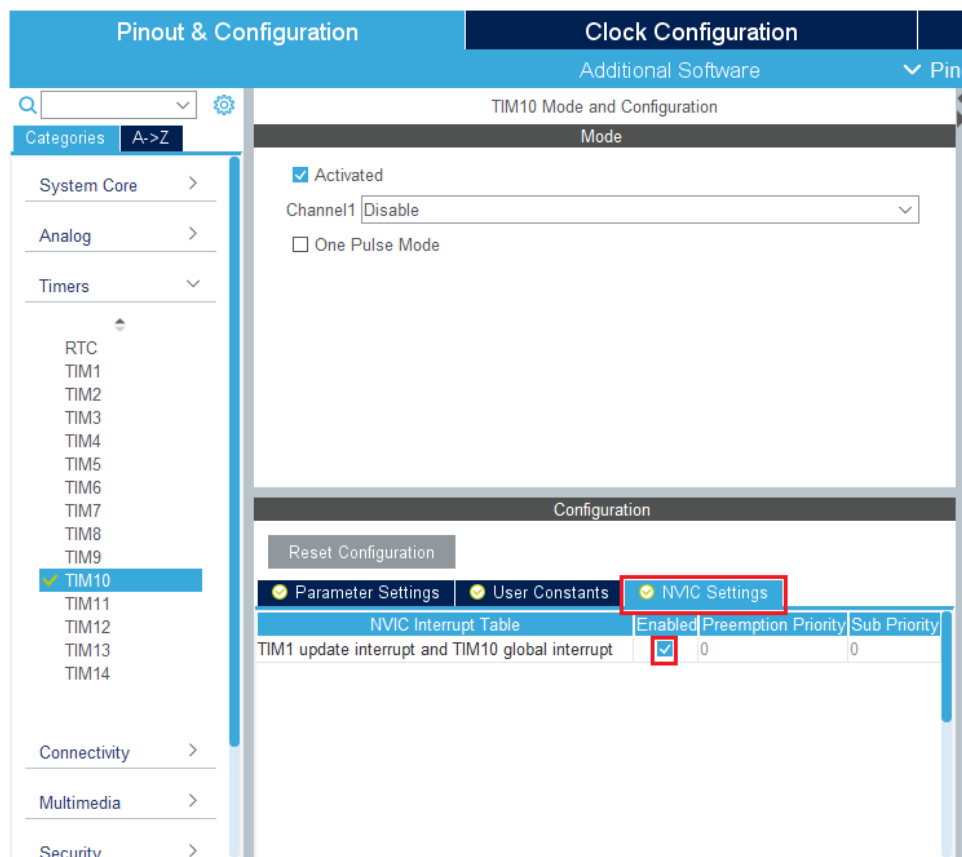




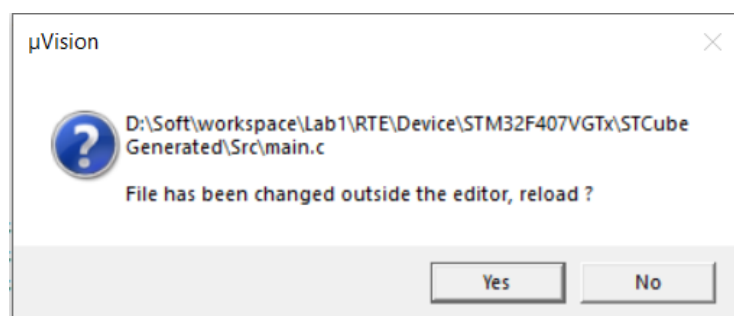
Мы задали делитель *Prescaler (PSC)* = 21000. Таким образом, *TIM10* будет работать на частоте:  $21000000 \text{ Гц} / 21000 \text{ (PSC)} = 1000 \text{ Гц}$ . Это значит, что счетчик таймера *TIM10* (регистр *TIM\_CNT*) будет инкрементироваться с частотой 1000 Гц т.е. Каждую  $1/1000 \text{ Гц} = 0.001 \text{ с} = 1 \text{ мс}$ .

*Counter Period (AutoReload Register)* = 500. Это значит, что таймер будет инкрементировать свой счетчик (*TIM\_CNT*) пока не достигнет значения 500. После этого таймер перезапустится и начнет счет с 0. Т.к таймер инкрементируется каждую 1 мс, то *TIM\_CNT* достигнет значения 500 через:  $1 \text{ мс} * 500 = 500 \text{ мс}$ .

Теперь разрешим прерывание от *TIM10*. Для этого перейдем во вкладку *NVIC Settings* и разрешим *Update Interrupt*. Т.е. Прерывание будет срабатывать каждый раз, когда значение в регистре *TIM\_CNT* будет становиться равным значению в регистре *AutoReload (ARR)* т.е. через каждые 500 мс.



Перегенерируем код. На все последовавшие вопросы отвечаем утвердительно.



Откроем файл *main.c*. У нас появилась функция инициализации таймера *TIM10*

```
main.c
157 static void MX_TIM10_Init(void)
158 {
159     /* USER CODE BEGIN TIM10_Init 0 */
160
161     /* USER CODE END TIM10_Init 0 */
162
163     /* USER CODE BEGIN TIM10_Init 1 */
164
165     /* USER CODE END TIM10_Init 1 */
166     htim10.Instance = TIM10;
167     htim10.Init.Prescaler = 21000;
168     htim10.Init.CounterMode = TIM_COUNTERMODE_UP;
169     htim10.Init.Period = 500;
170     htim10.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
171     htim10.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
172     if (HAL_TIM_Base_Init(&htim10) != HAL_OK)
173     {
174         Error_Handler();
175     }
176     /* USER CODE BEGIN TIM10_Init 2 */
177
178     /* USER CODE END TIM10_Init 2 */
179
180 }
181
182
```

И вызов этой функции из *int main(void)*

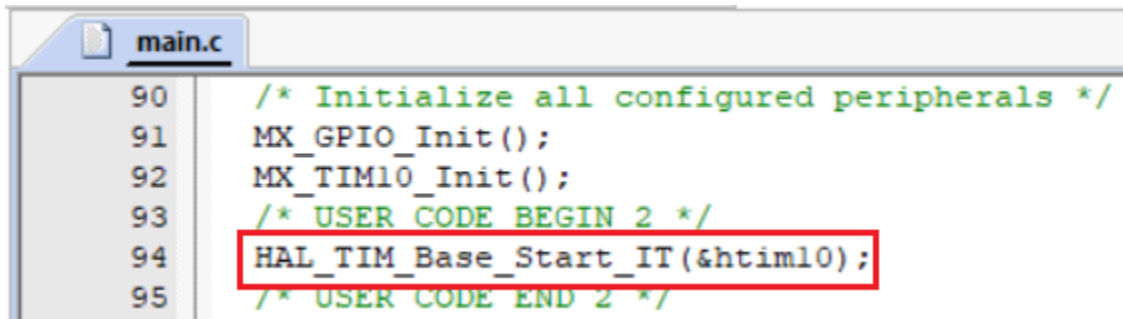
```
main.c
67 int main(void)
68 {
69     /* USER CODE BEGIN 1 */
70
71     /* USER CODE END 1 */
72
73
74     /* MCU Configuration----- */
75
76     /* Reset of all peripherals, Initializes the
77     HAL_Init();
78
79     /* USER CODE BEGIN Init */
80
81     /* USER CODE END Init */
82
83     /* Configure the system clock */
84     SystemClock_Config();
85
86     /* USER CODE BEGIN SysInit */
87
88     /* USER CODE END SysInit */
89
90     /* Initialize all configured peripherals */
91     MX_GPIO_Init();
92     MX_TIM10_Init();
93     /* USER CODE BEGIN 2 */

```

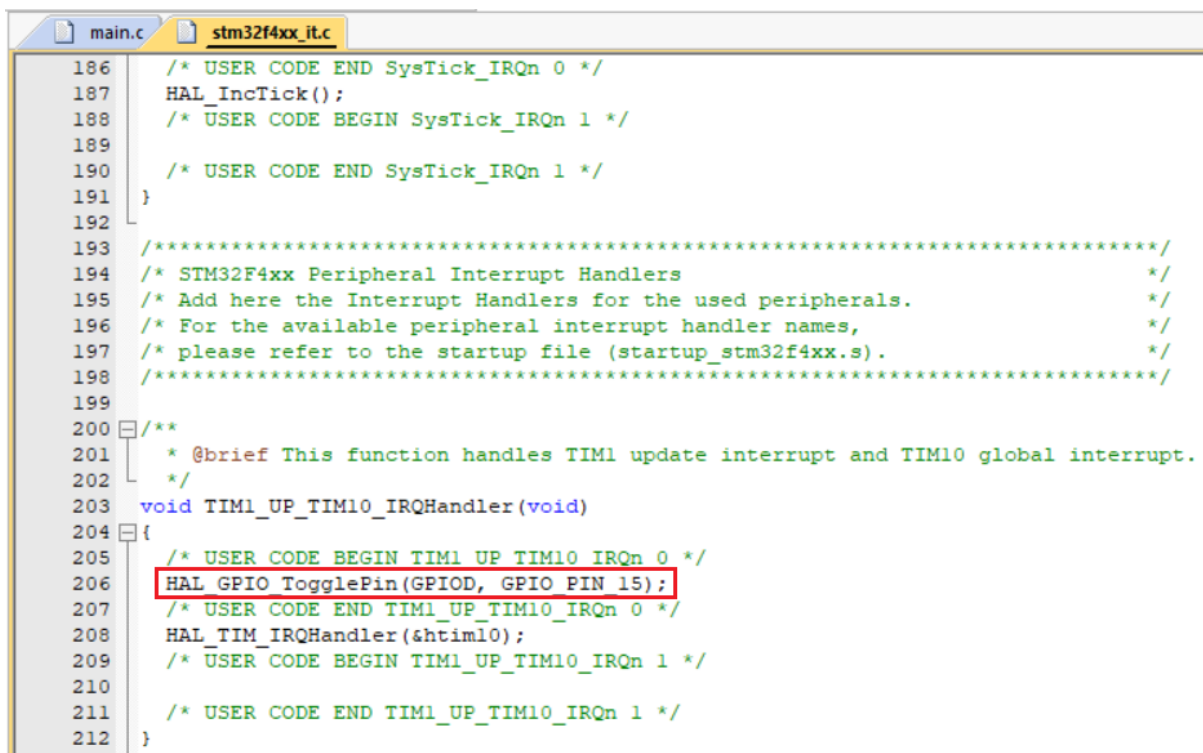
Теперь перенесем вызов изменения состояния *PD15* из цикла *while* в обработчик прерывания *void TIM1\_UP\_TIM10\_IRQHandler(void)* в файле *stm32f4xx\_it.c*. Также удалим блокирующую задержку из цикла *while*. Таким образом наш главный цикл *while* остался пустым.

После инициализации *TIM10* (т.е. после вызова функции *MX\_TIM10\_Init()*) запустим *TIM10* с разрешенным прерыванием

```
HAL_TIM_Base_Start_IT(&htim10);
```



```
main.c
90  /* Initialize all configured peripherals */
91  MX_GPIO_Init();
92  MX_TIM10_Init();
93  /* USER CODE BEGIN 2 */
94  HAL_TIM_Base_Start_IT(&htim10);
95  /* USER CODE END 2 */
```



```
main.c  stm32f4xx_it.c
186  /* USER CODE END SysTick_IRQn 0 */
187  HAL_IncTick();
188  /* USER CODE BEGIN SysTick_IRQn 1 */
189
190  /* USER CODE END SysTick_IRQn 1 */
191  }
192
193  /*****
194  /* STM32F4xx Peripheral Interrupt Handlers
195  /* Add here the Interrupt Handlers for the used peripherals.
196  /* For the available peripheral interrupt handler names,
197  /* please refer to the startup file (startup_stm32f4xx.s).
198  /*****
199
200  /**
201  * @brief This function handles TIM1 update interrupt and TIM10 global interrupt.
202  */
203  void TIM1_UP_TIM10_IRQHandler(void)
204  {
205  /* USER CODE BEGIN TIM1_UP_TIM10_IRQn 0 */
206  HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_15);
207  /* USER CODE END TIM1_UP_TIM10_IRQn 0 */
208  HAL_TIM_IRQHandler(&htim10);
209  /* USER CODE BEGIN TIM1_UP_TIM10_IRQn 1 */
210
211  /* USER CODE END TIM1_UP_TIM10_IRQn 1 */
212  }
```

Собираем проект и запускаем под отладкой

TODO: ADC