ГУАП

STM & XILINX

«Программирование» встраиваемых систем на базе микроконтроллеров и ПЛИС

Владимир Хрусталев
Email : v_crys@mail.ru

Часть I
Знакомство с STM32

План курса

1. Часть I. Программирование микроконтроллеров
   1. Основы STM32. Обзор средств разработки. Первая программа.
   2. **Таймеры и прерывания. Разрабатываем секундомер.**
   3. Внешние интерфейсы STM32. «Связываем» микроконтроллер с компьютером.

2. Часть II. Проектирование микросхем и их отладка на FPGA
   1. Знакомимся с FPGA и SystemVerilog. Обзор средств разработки. Первые опыты.
   2. Цифровая схемотехника. Синхронная и комбинаторная логика. Конечные автоматы. Разрабатываем светофор.
   3. Верификация. Разработка аппаратного модуля UART.

3. Часть III. (опционально) О разработке процессоров. Архитектура и микроархитектура. «Живой» пример: процессор Syntacore SCR1

ГУАП

STM & XILINX

Занятие №2

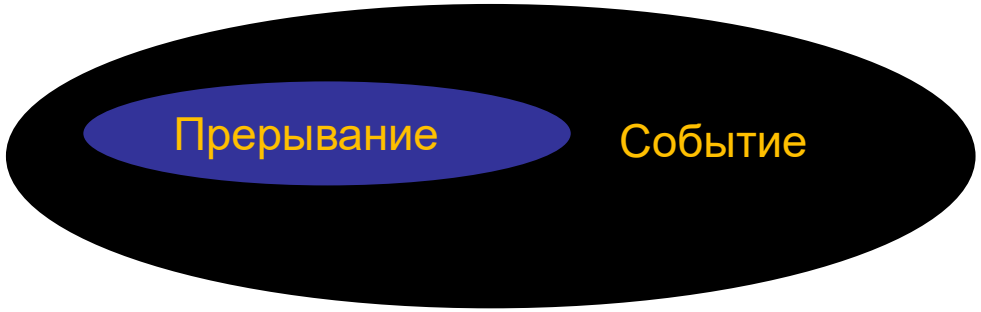
Таймеры и прерывания

Прерывания

Событие (event) – какое-либо произошедшее действие. Примеры событий в МК: нажата кнопка, таймер досчитал до нуля и т. д.

Прерывание (interrupt) – приостановка выполняемых действий и переключение на другие задачи. После завершения задач, возвращение к изначально выполняемым действиям.

Прерывания



Событие может вызывать прерывание, а может и не вызывать.
Прерывание всегда является обработкой события

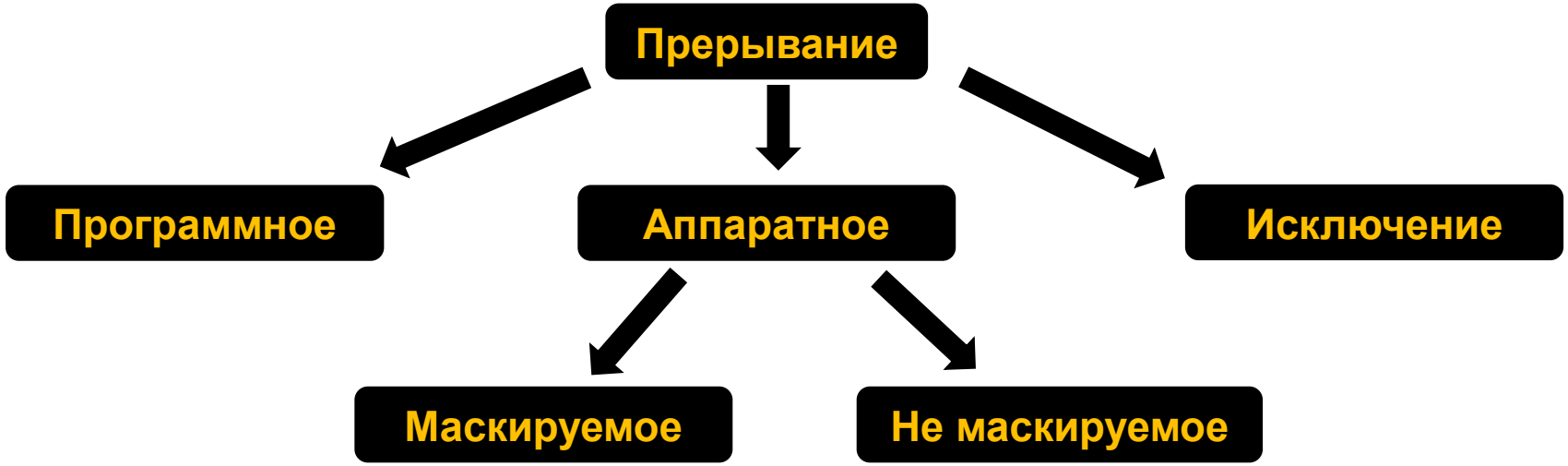
Приоритеты прерываний

В случае одновременного возникновения нескольких прерываний, они будут выполняться в очередности своих приоритетов. Прерывания с одинаковыми приоритетами выполняются последовательно.

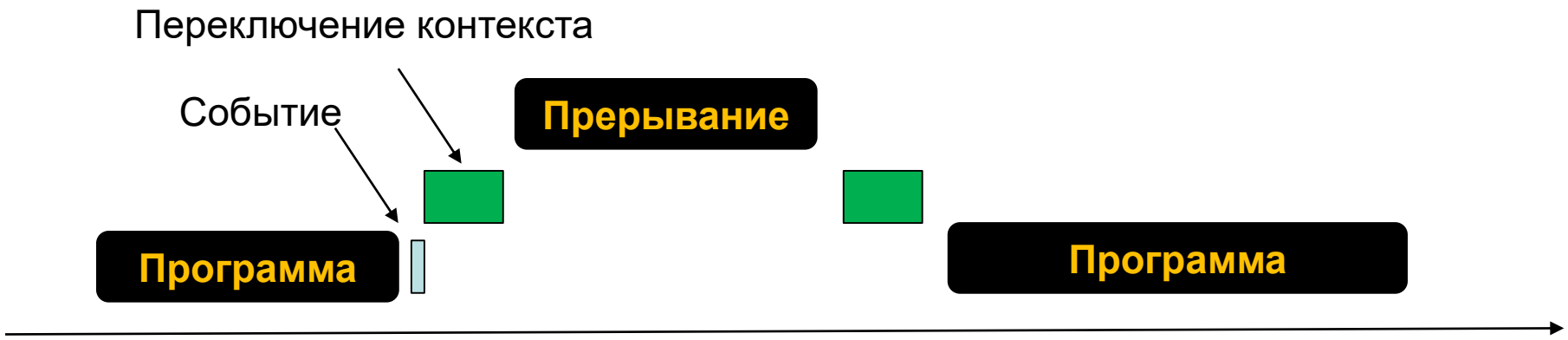
Если прерывание уже обрабатывается и приходит прерывание с наибольшим приоритетом, то происходит переключение на более приоритетное прерывание. После завершения более приоритетного прерывания будет завершаться прерванное низкоприоритетное прерывание.

NVIC – контроллер вложенных векторных прерываний – управляет порядком выполнения всех прерываний и содержит таблицу, в которой перечислены прерывания и их приоритеты.

Классификация прерываний



Контекст



Прерывание может возникнуть в любой момент. Основная программа хранит в регистрах временные значения, которые необходимы ей для вычислений. Подпрограмма прерываний также может использовать эти регистры. В связи с этим может возникнуть конфликт. Подпрограмма прерываний может изменить регистры, из-за чего основная программа потеряет свои временные значения. Для исключения таких ситуаций, при возникновении прерывания значения регистров сохраняются в стек, а после завершения прерывания, восстанавливаются из стека. Такая операция называется переключением контекста.

Контекст программы – состояние ее регистров в частном, и памяти, в общем случае.

Последовательность действий при прерывании

Событие

Выставляется флаг события

Проверяется маскирующий бит прерывания

Сохранение контекста

Прерывание

Особенности контроллера NVIC

- **Features**
- • 68 (not including the sixteen Cortex®-M3 interrupt lines)
- • 16 programmable priority levels (4 bits of interrupt priority are used)
- • Low-latency exception and interrupt handling
- • Power management control
- • Implementation of System Control Registers
- The NVIC and the processor core interface are closely

- **Поддержка вложенных прерываний**
- **Поддержка векторных прерываний**
- **Поддержка динамического изменения приоритетов**
- **Уменьшение времени реакции на прерывание**
- **Маскирование прерываний**

Таблица прерываний

Table 61. Vector table for connectivity line devices

Position	Priority	Type of priority	Acronym	Description	Address
-	-	-	-	Reserved	0x0000_0000
-	-3	fixed	Reset	Reset	0x0000_0004
-	-2	fixed	NMI	Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector.	0x0000_0008
-	-1	fixed	HardFault	All class of fault	0x0000_000C
-	0	settable	MemManage	Memory management	0x0000_0010
-	1	settable	BusFault	Pre-fetch fault, memory access fault	0x0000_0014
-	2	settable	UsageFault	Undefined instruction or illegal state	0x0000_0018
-	-	-	-	Reserved	0x0000_001C - 0x0000_002B
-	3	settable	SVCall	System service call via SWI instruction	0x0000_002C
-	4	settable	Debug Monitor	Debug Monitor	0x0000_0030
-	-	-	-	Reserved	0x0000_0034
-	5	settable	PendSV	Pendable request for system service	0x0000_0038
-	6	settable	SysTick	System tick timer	0x0000_003C
0	7	settable	WWDG	Window Watchdog interrupt	0x0000_0040
1	8	settable	PVD	PVD through EXTI Line detection interrupt	0x0000_0044
2	9	settable	TAMPER	Tamper interrupt	0x0000_0048
3	10	settable	RTC	RTC global interrupt	0x0000_004C
4	11	settable	FLASH	Flash global interrupt	0x0000_0050
5	12	settable	RCC	RCC global interrupt	0x0000_0054
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000_0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000_005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000_0060
9	16	settable	EXTI3	EXTI Line3 interrupt	0x0000_0064
10	17	settable	EXTI4	EXTI Line4 interrupt	0x0000_0068

Таймеры

Таймер – это счетчик, тактируемый от некоторого сигнала с заранее известной и стабильной частотой.

Таймер подсчитывает количество пришедших импульсов. Если импульсы следуют со строго заданным периодом, то легко перевести это количество импульсов во время.

Допустим, имеется часовой кварц с частотой 32768 Гц. Сигнал с кварца подается на 15-разрядный таймер. Для переполнения 15 разрядного таймера необходимо 2^15 = 32768 тактов. Таким образом переполнение 15 разрядного таймера будет происходить ровно раз в секунду.
Кварц с частотой 32768 используется для часов реального времени.

Для чего используются таймеры в МК

Казалось бы, ответ очевиден, для подсчета времени. Однако, это очень обобщенный ответ. Таймеры в современных МК имеют большое количество различных режимов работы и могут быть использованы для:

1. Генерации сигнала ШИМ. Генерация сигнала *(миандра)* на высокой частоте с различной скважностью *(отношением длительности веского состояния к низкому в течение периода)*
2. Захват сигнала. Иногда необходимо определить время между двумя фронтами сигнала. Для этого удобно использовать таймер в режиме захвата сигнала. Когда появляется фронт сигнала, срабатывает прерывание и можно сохранить текущее значение таймеры. На следующем фронте операцией вычитания можно определить прошедшее время. (импульсы от датчика Холла закрепленного на колесе велосипеда).
3. Режим сравнения. Аппаратно сравнивается значение таймера со значением некоторого регистра. В случае совпадения происходит заранее заданное событие.
4. Тактирование других таймеров

Виды таймеров внутри stm32f103

Таймер внутри ядра:

- Системный таймер (24-битный)

Таймеры, представленные в виде модулей:

- Базовый таймер
- Таймеры общего назначения (16-битные, имеют различные режимы работы и несколько каналов). Каждый канал может создавать определенное событие.
- Расширенные таймеры. (16-битные, имеют очень богатый функционал)
- Два сторожевых таймера IWDT, WWDT

The STM32xx Series devices, based on the Arm® cores[(a)], have various built-in timers outlined as follows;

- **General-purpose timers** can be used by any application for: output comparison (timing and delay generation), one-pulse mode, input capture (for external signal frequency measurement), sensor interface (encoder, hall sensor);
- **Advanced timers**: these timers have the most features. In addition to general purpose functions, they include several features related to motor control and digital power conversion applications: three complementary signals with deadtime insertion and emergency shut-down input;
- One or two **channel timers**: used as general-purpose timers with a limited number of channels.
- One or two channel timers with **complementary output**: same as the previous timer type with an additional deadtime generator on one channel. In some situations, this feature allows a general purpose timer to be used where an additional advanced timer would be necessary.
- **Basic timers** are used either as timebase timers or for triggering the DAC peripheral These timers don't have any input/output capabilities;
- **Low-power timers** are simple general purpose timers and are able to operate in low-power modes.They are used to generate a wake-up event for example;
- **High-resolution timers** are specialized timer peripherals designed to drive power conversion in lighting and power source applications. They can also be used in other fields that require very fine timing resolution. AN4885, AN4539 and AN4449 are practical examples of high-resolution timer use.

Параметры таймеров

Table 4. Timer features overview

Timer type	Counter resolution	Counter type	DMA	Channels	Output. channels	Synchronization	
						Master configuration	Slave configuration
Advanced Control	16 bit	Up, down and center aligned	Yes	4, 6(1)	3	Yes	Yes
General-purpose	16 bit 32 bit(2)	Up, down and center aligned	Yes	Up to 4	0	Yes	Yes
Basic	16 bit	Up	Yes	0	0	Yes	No
1-channel	16 bit	Up	No	1	0	Yes (OC signal)	No
2-channel	16 bit	Up(3)	No	2	0	Yes	Yes
1-channel with one complementary output	16 bit	Up	Yes	1	1	Yes (OC signal)	No
2-channel with one complementary output	16 bit	Up	Yes	2	1	Yes	Yes
Low-power timer	16 bit	Up	No	1(4)	0	Yes (OC signal)	No
High-resolution timer	16 bit	Up	Yes	Up to 12(4)	Up to 6	Yes	Yes

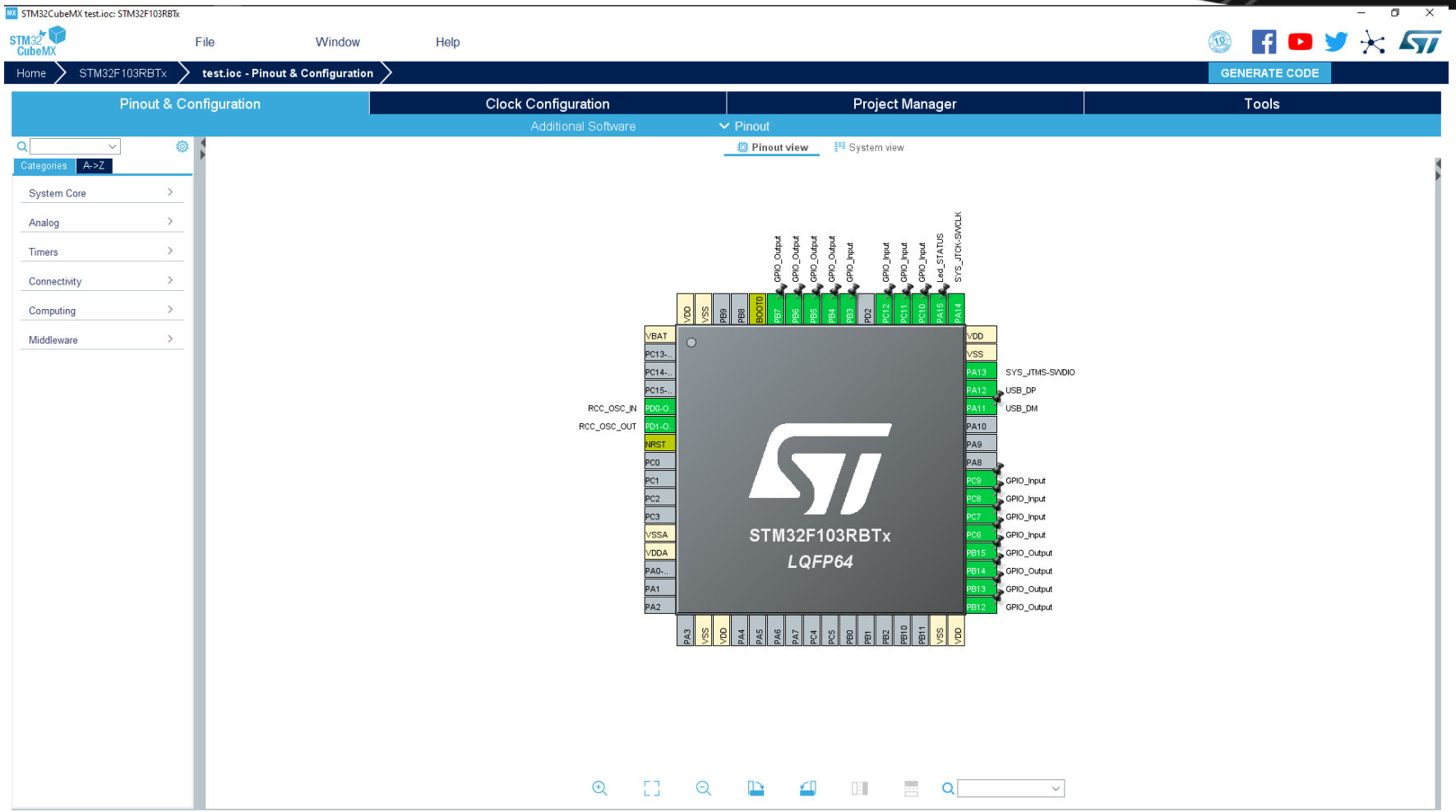
ГУАП

STM & XILINX

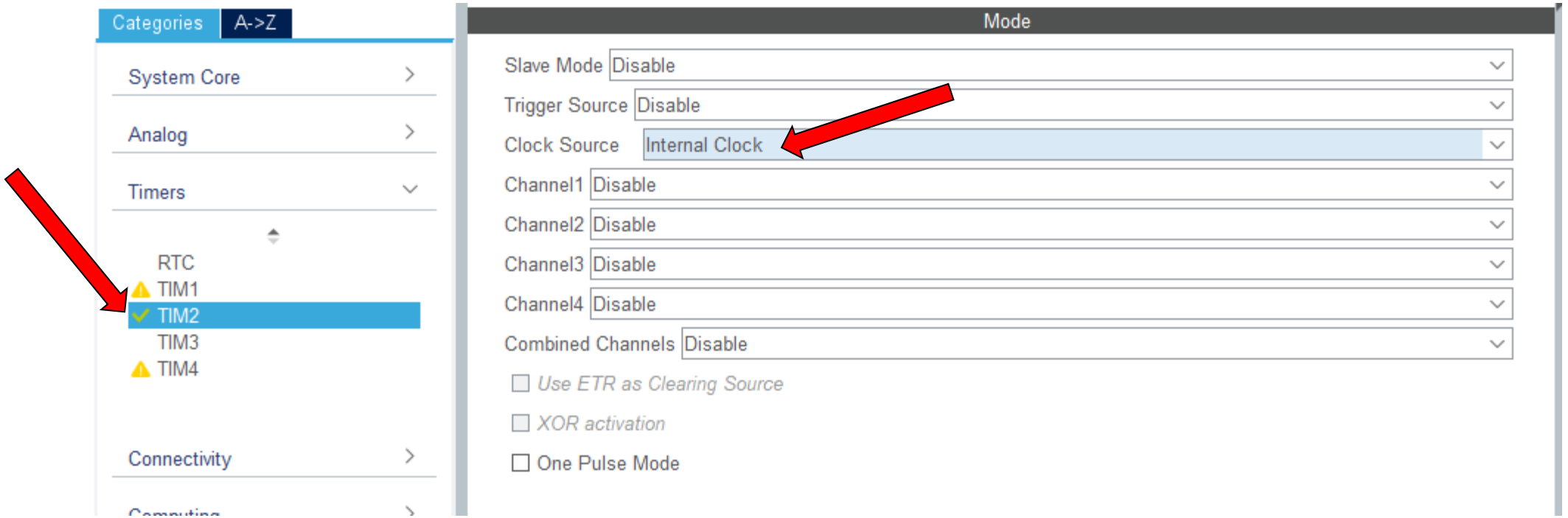
Занятие №2

Разрабатываем секундомер

Берем за основу проект с предыдущего урока



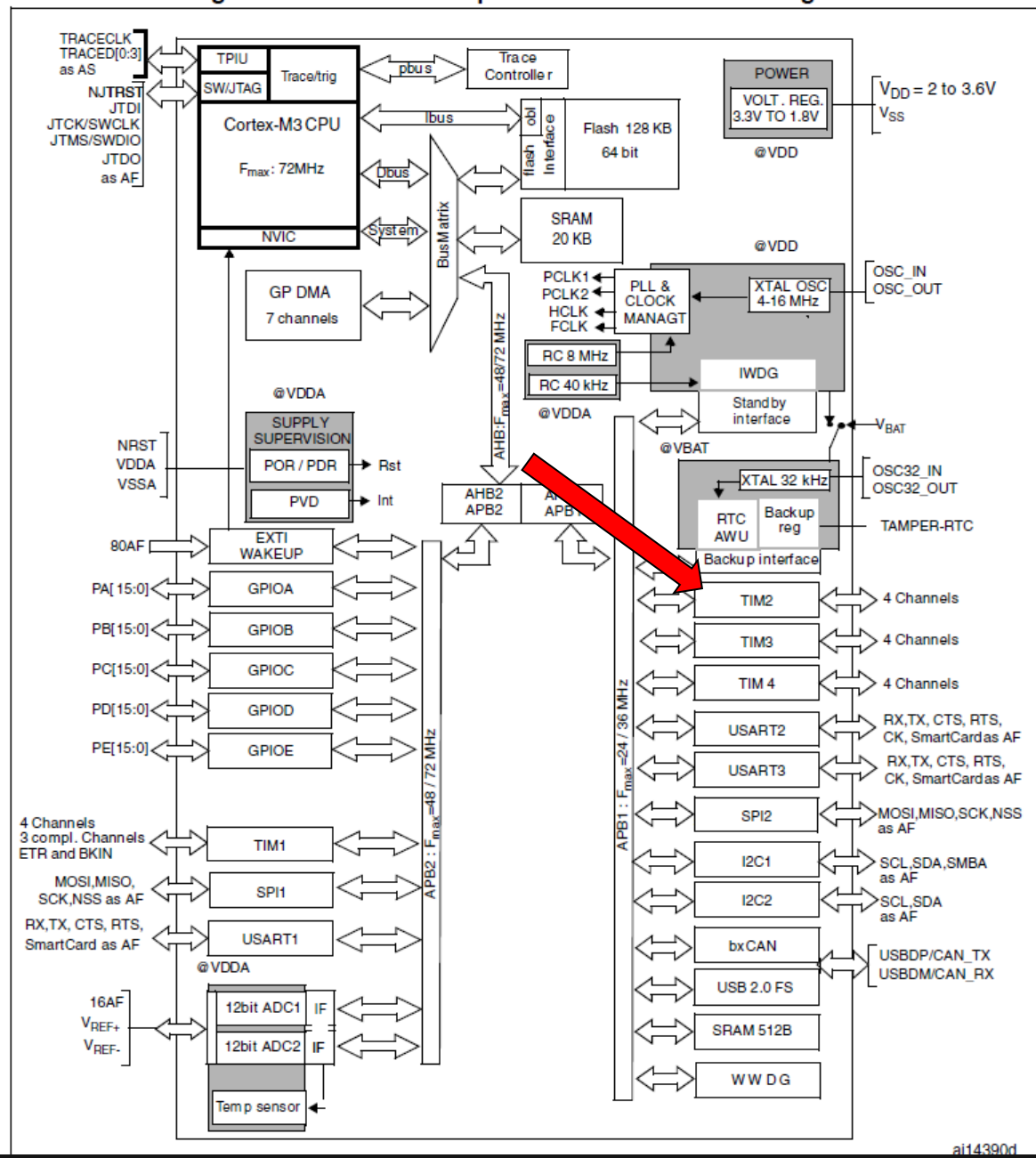
Включаем таймер (тактируем от внутренней шины APB)



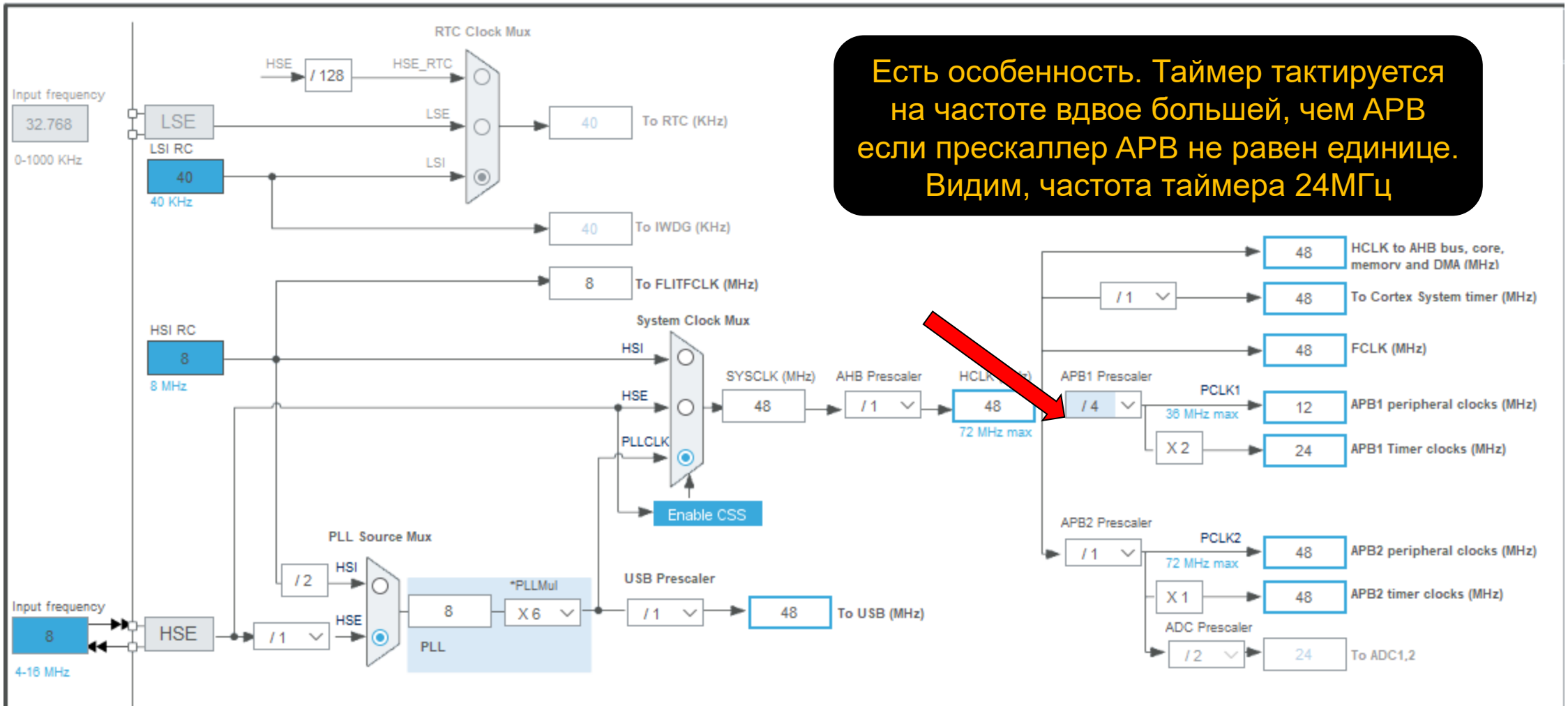
Смотрим по схеме МК куда подключен таймер

Таймер подключен к шине APB1. Тактироваться он будет от тактового импульса этой шины

Figure 1. STM32F103xx performance line block diagram



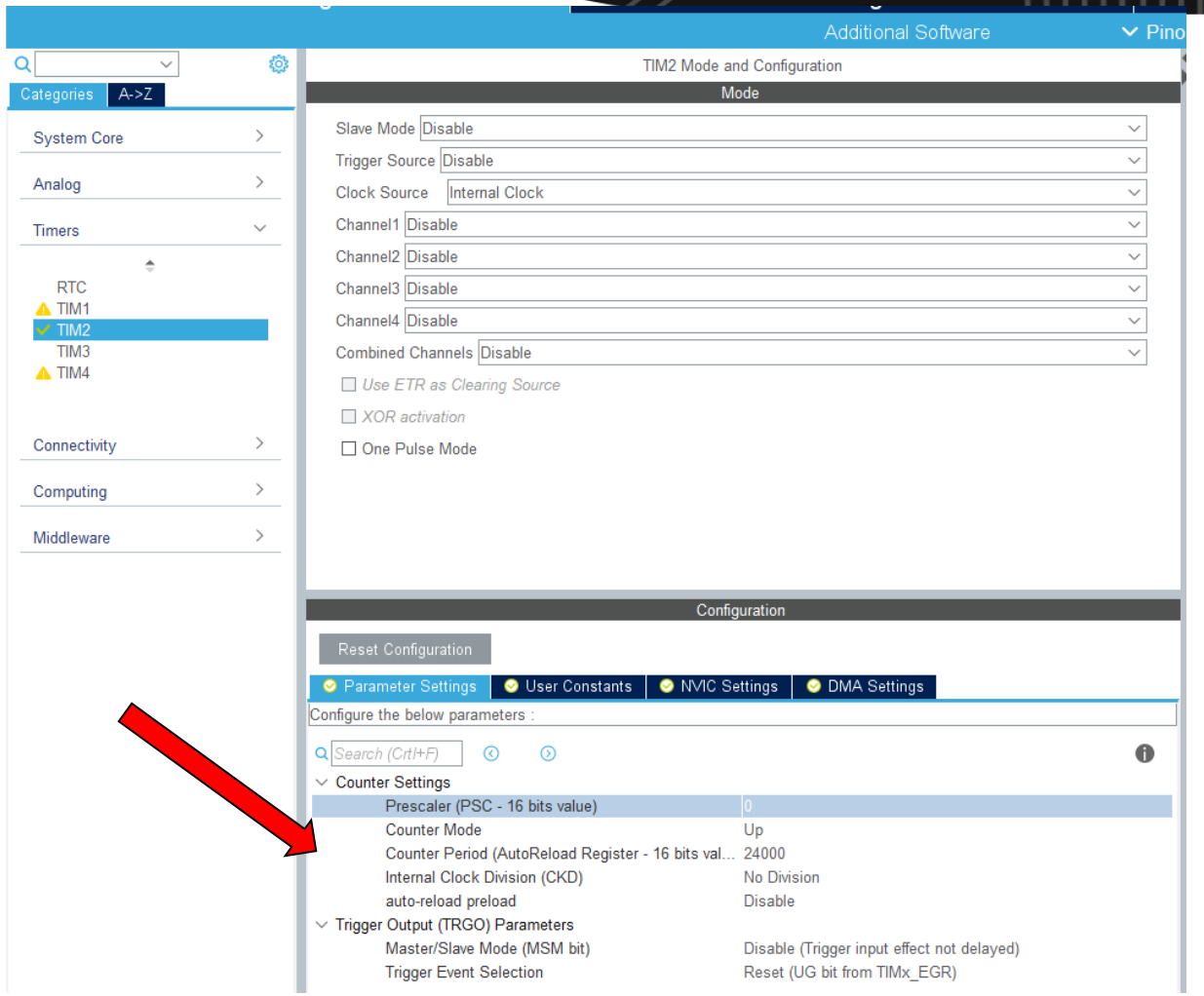
Определяем по дереву тактовых импульсов частоту APB1



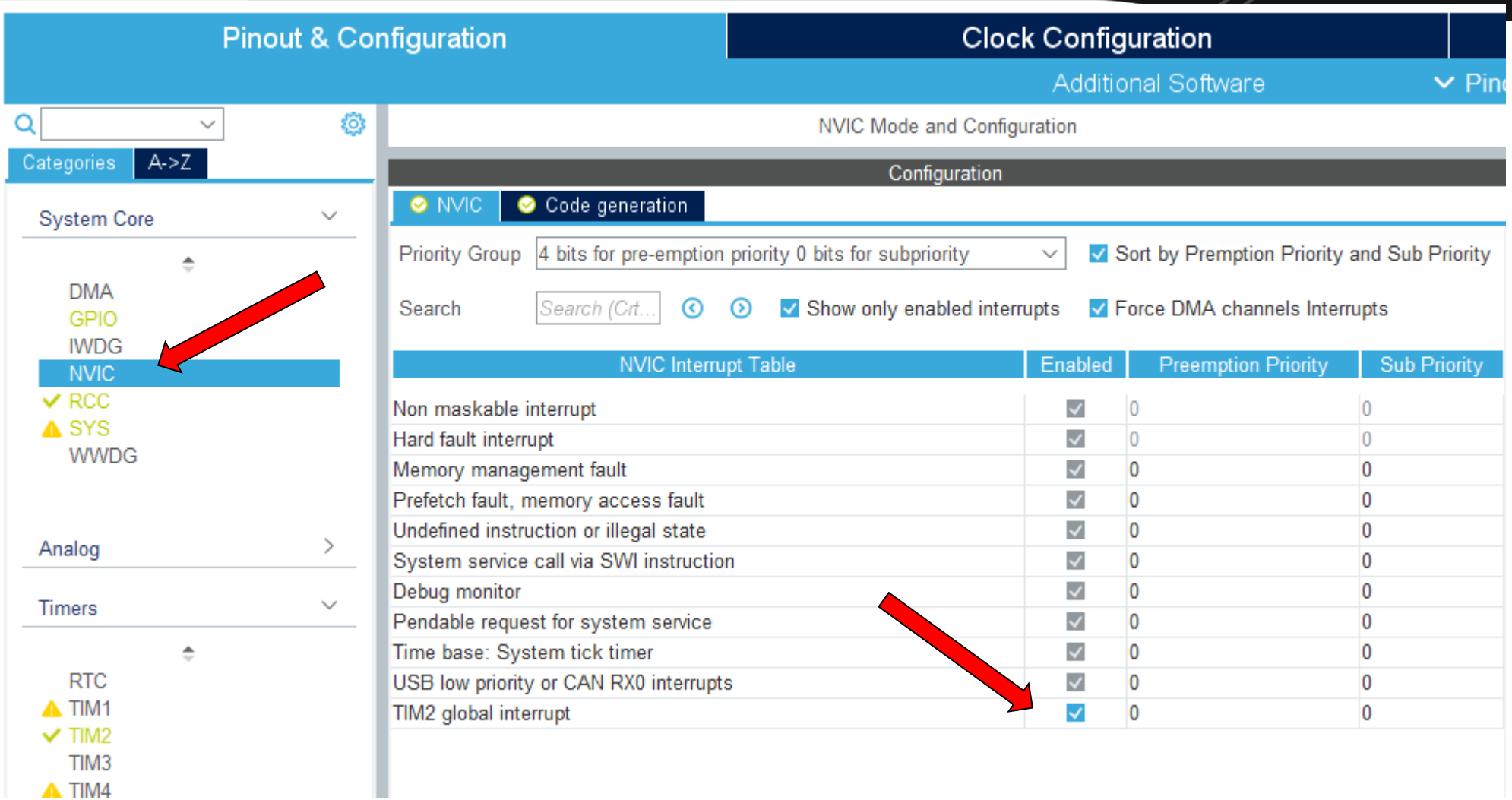
Есть особенность. Таймер тактируется на частоте вдвое большей, чем APB если прескаллер APB не равен единице. Видим, частота таймера 24МГц

Настраиваем таймер

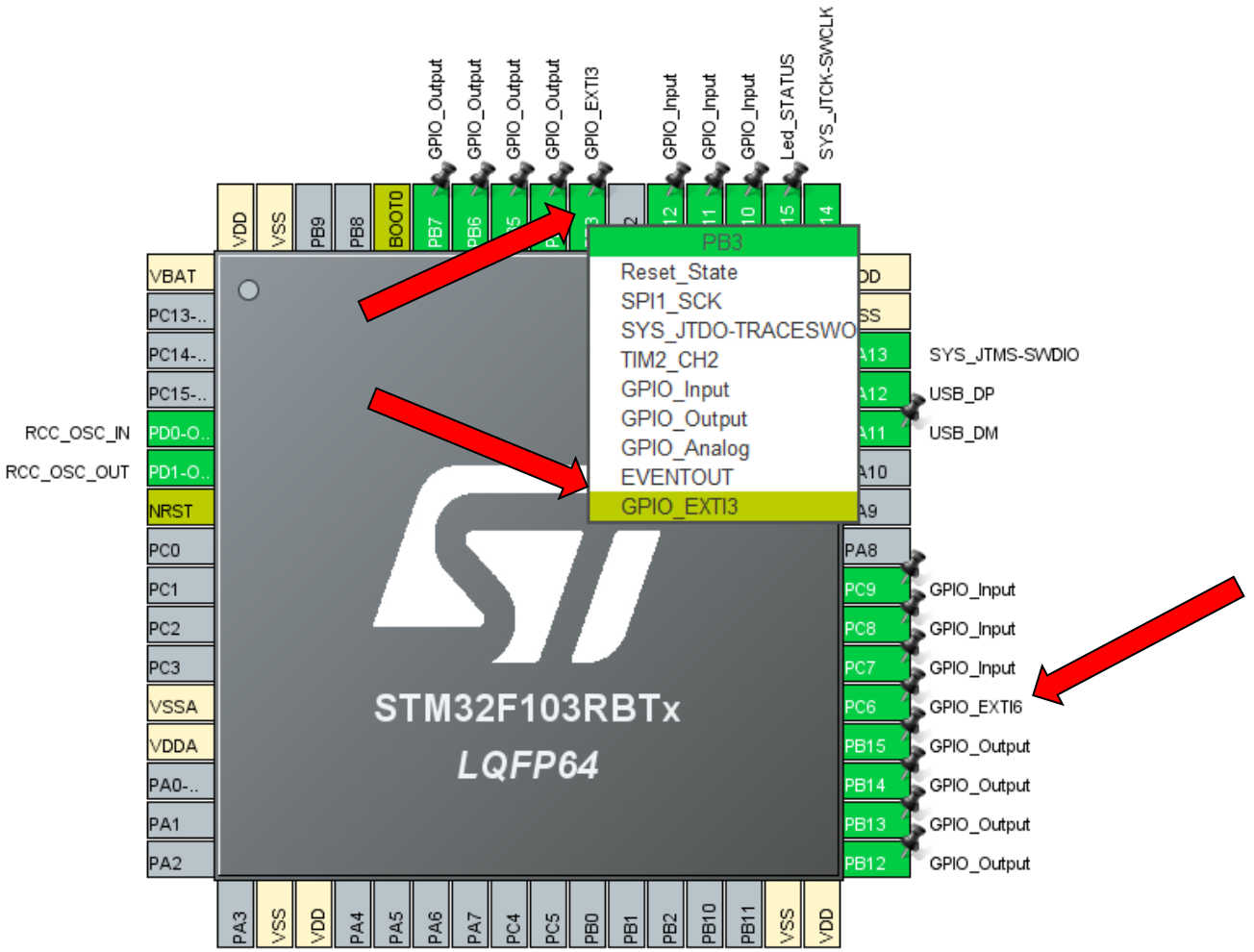
Таймер имеет встроенный предделитель, который позволяет поделить входную частоту. Однако, для нас в этом нет необходимости, так как таймер 16-разрядный и тактируется частотой 24МГц. Для того, чтобы таймер срабатывал раз в 1мс, необходимо установить период его срабатывания равным 24000.



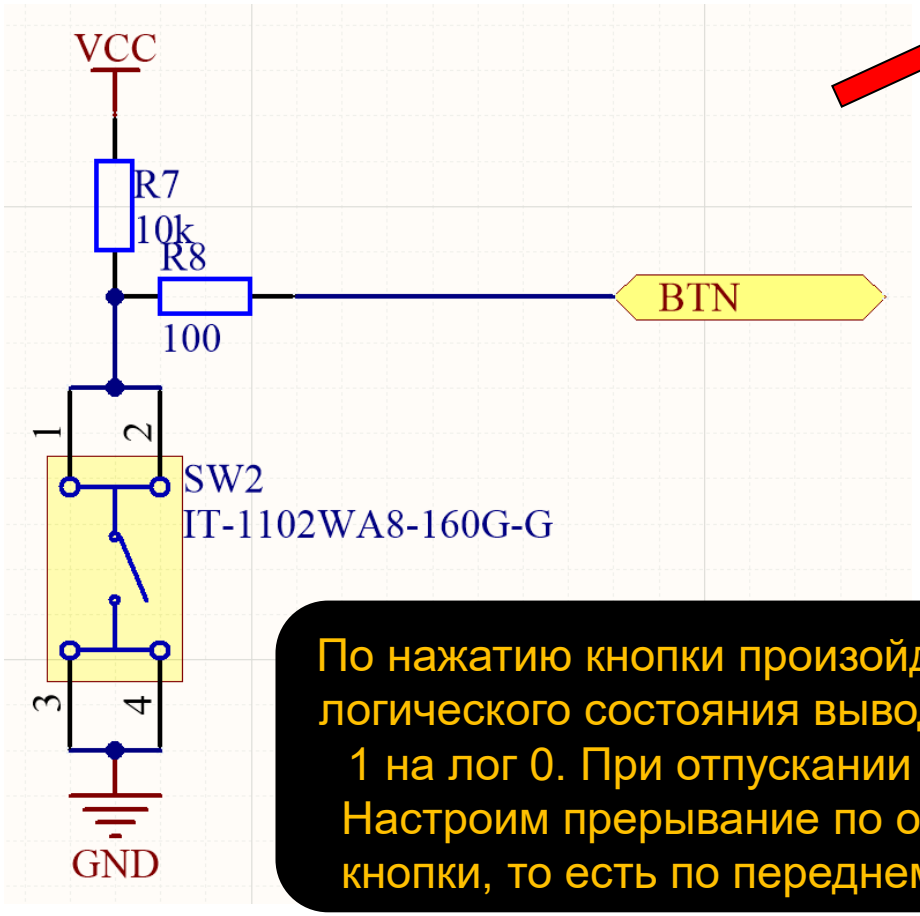
Включим прерывание по таймеру



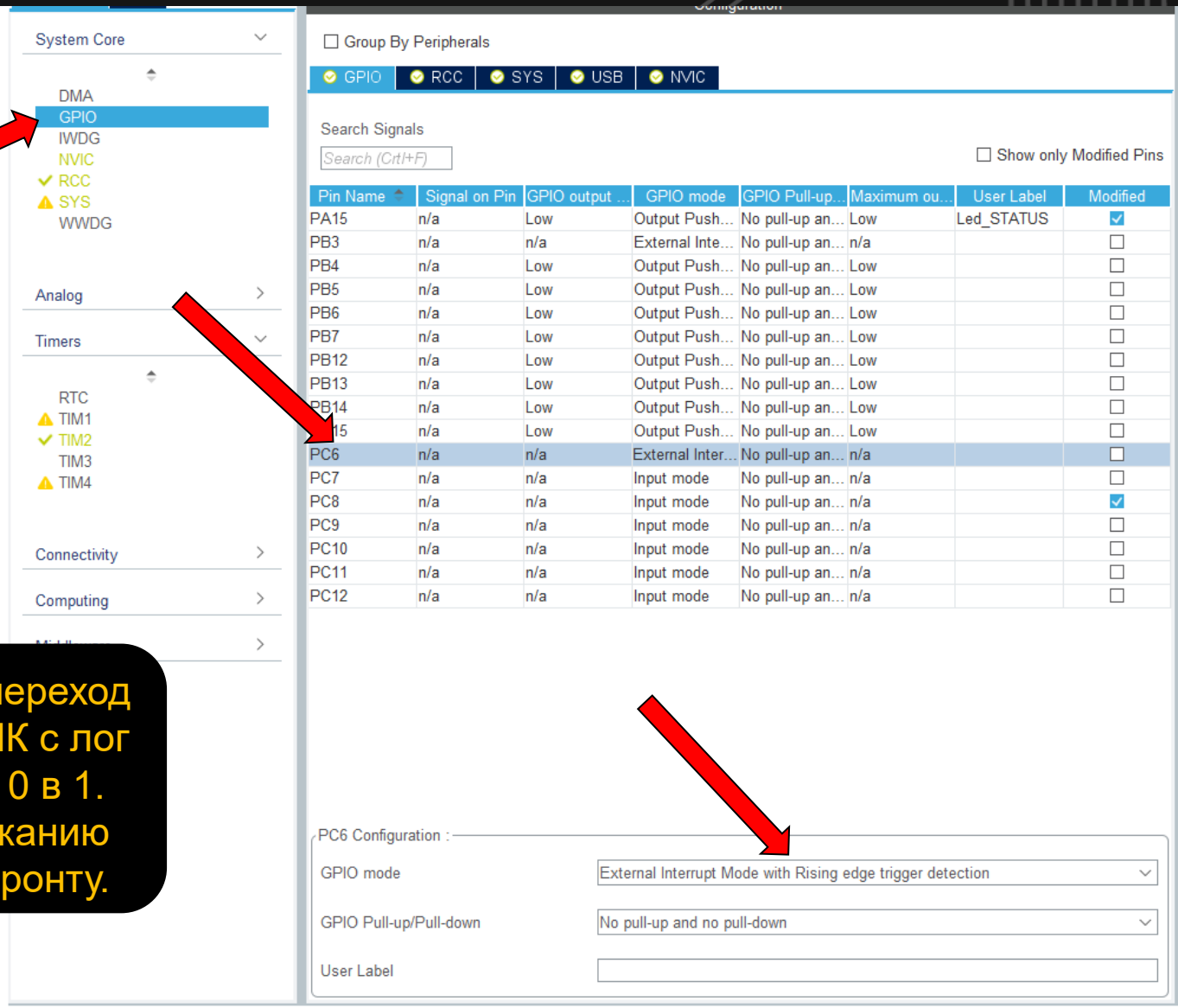
Теперь настроим прерывания по кнопкам PB3 и RC6



Настройки фронта, по которому будет срабатывать прерывание

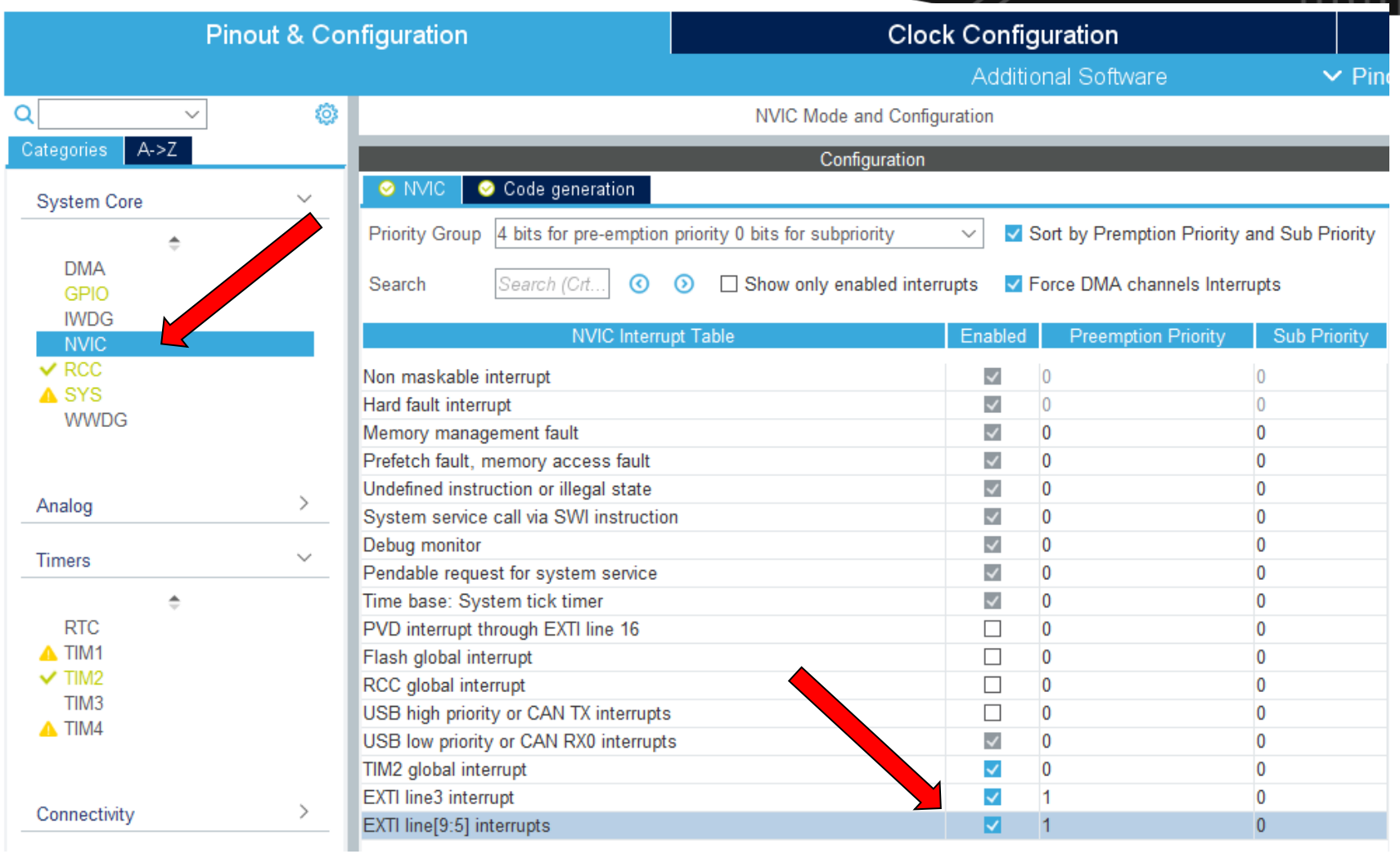


По нажатию кнопки произойдет переход логического состояния вывода МК с лог 1 на лог 0. При отпускании – из 0 в 1. Настроим прерывание по отпусканию кнопки, то есть по переднему фронту.

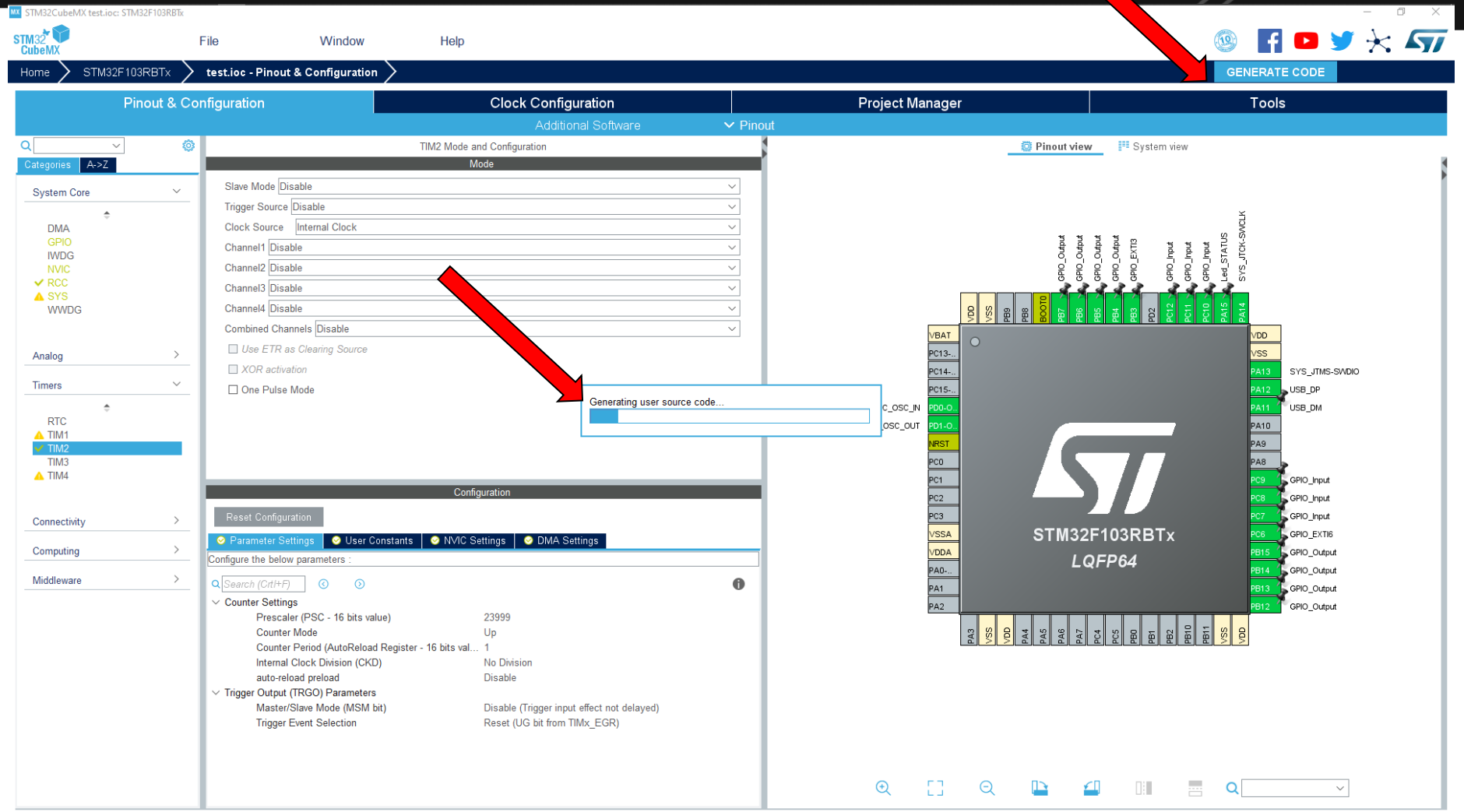


Включаем прерывание

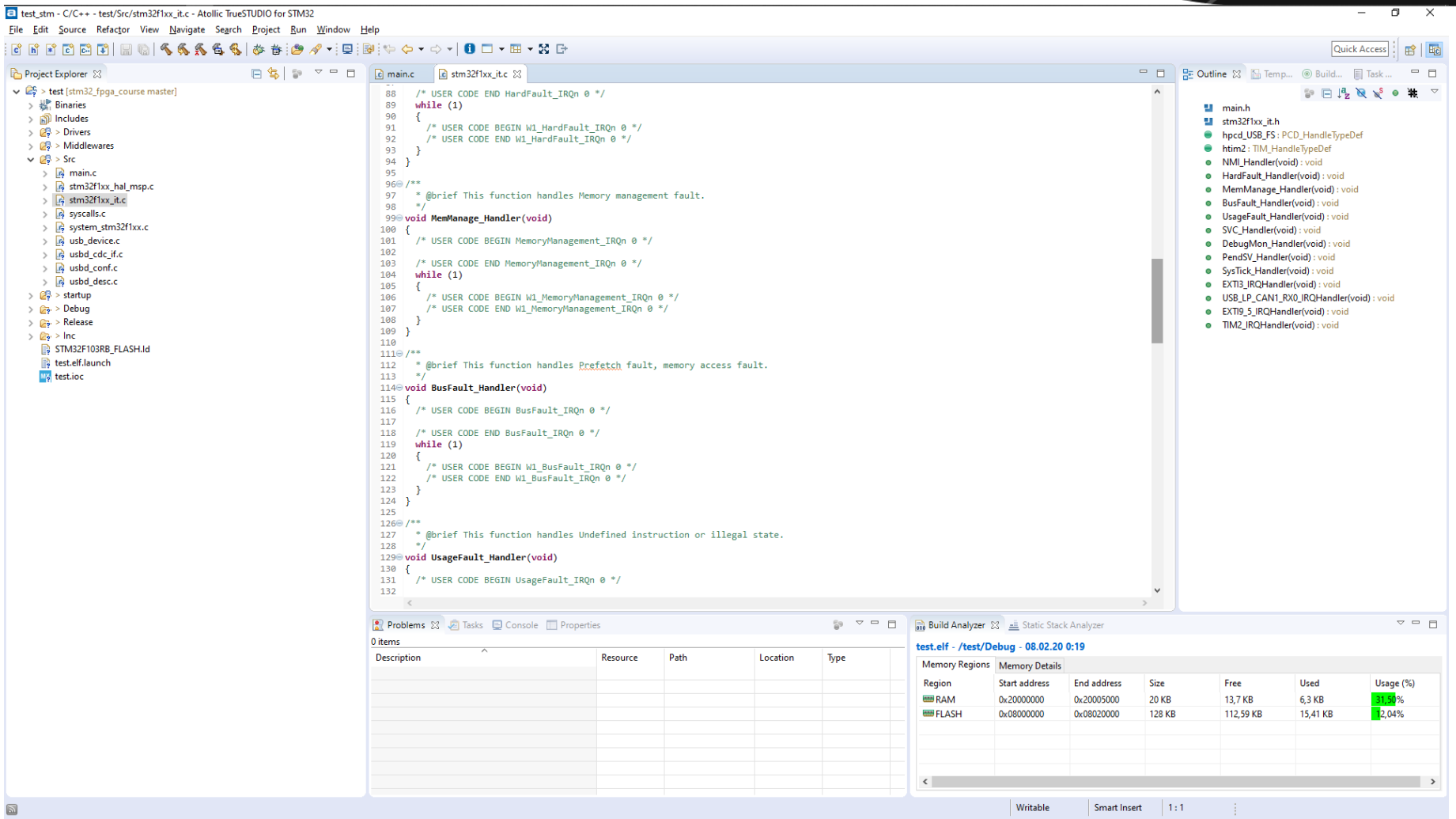
Зададим кнопкам меньший приоритет, чем таймеру (0 – более высокий приоритет чем 1)



Собираем проект



Открываем IDE (нас будет интересовать файл по работе с прерываниями stm32f1xx_it.c)



Находим обработчики прерываний

```c
void EXTI9_5_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI9_5_IRQn 0 */

  /* USER CODE END EXTI9_5_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_6);
  /* USER CODE BEGIN EXTI9_5_IRQn 1 */

  /* USER CODE END EXTI9_5_IRQn 1 */
}
```

```c
/**
  * @brief This function handles EX
 interrupt.
  */
void EXTI3_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI3_IRQn 0 */

  /* USER CODE END EXTI3_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_3);
  /* USER CODE BEGIN EXTI3_IRQn 1 */

  /* USER CODE END EXTI3_IRQn 1 */
}
```

```c
void TIM2_IRQHandler(void)
{
  /* USER CODE BEGIN TIM2_IRQn 0 */

  /* USER CODE END TIM2_IRQn 0 */
  HAL_TIM_IRQHandler(&htim2);
  /* USER CODE BEGIN TIM2_IRQn 1 */

  /* USER CODE END TIM2_IRQn 1 */
}
```

Включим таймер в main и создадим глобальные переменные для подсчета времени в _it.c

```
  /* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start_IT(&htim2);
  /* USER CODE END 2 */
```

```
/* Private variables ---------------------
-------------------------------------*/
/* USER CODE BEGIN PV */
  int glob_time = 0;
  int time_after_btn  = 0;

  int time_sec_ctr = 0;

  int time_s = 0;

  int flag_timer_pause = 0;
/* USER CODE END PV */
```

Будем мигать светодиодом статуса раз в 500 мс

```
void TIM2_IRQHandler(void)
{
  /* USER CODE BEGIN TIM2_IRQn 0 */

  /* USER CODE END TIM2_IRQn 0 */
  HAL_TIM_IRQHandler(&htim2);
  /* USER CODE BEGIN TIM2_IRQn 1 */
  if (glob_time > 500)
  {
    glob_time = 0;
    HAL_GPIO_TogglePin(Led_STATUS_GPIO_Port, Led_STATUS_Pin);
  } else glob_time++;

  /* USER CODE END TIM2_IRQn 1 */
}
```

Добавим счетчик секунд

```
void TIM2_IRQHandler(void)
{
        ///……

  if (time_sec_ctr >= 1000) {
    time_s++;
    time_sec_ctr = 0;

    if (time_s >= 256) time_s = 0;
  } else time_sec_ctr++;

  /* USER CODE END TIM2_IRQn 1 */
}
```

По кнопке будем сбрасывать счетчик секунд

```
void EXTI3_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI3_IRQn 0 */

  /* USER CODE END EXTI3_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_3);
  /* USER CODE BEGIN EXTI3_IRQn 1 */

  time_s = 0;
  time_sec_ctr = 0;

  /* USER CODE END EXTI3_IRQn 1 */
}
```

По другой кнопке будем ставить на паузу секундомер

Следует обратить внимание, что функция прерывания по второй кнопке отвечает сразу за несколько выводов

`void EXTI9_5_IRQHandler(void)`

Посмотрим содержимое функции HAL_GPIO_EXTI_IRQHandler

```
void EXTI9_5_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI9_5_IRQn 0 */

  /* USER CODE END EXTI9_5_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_6);
  /* USER CODE BEGIN EXTI9_5_IRQn 1 */

  /* USER CODE END EXTI9_5_IRQn 1 */
}
```

```
void HAL_GPIO_EXTI_IRQHandler(uint16_t GP
IO_Pin)
{
  /* EXTI line interrupt detected */
  if (__HAL_GPIO_EXTI_GET_IT(GPIO_Pin) !=
 0x00u)
  {
    __HAL_GPIO_EXTI_CLEAR_IT(GPIO_Pin);
    HAL_GPIO_EXTI_Callback(GPIO_Pin);
  }
}
```

Как может быть видно, в функции вывязывается функция и в нее в качестве параметра передается номер пина HAL_GPIO_EXTI_Callback(GPIO_Pin);

Директива __weak указывает компилятору, что это низкоприоритетное тело функции. То есть, если где-то в другом месте есть другое тело этой функции, то будет вызвано оно.

```
__weak void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  /* Prevent unused argument(s) compilation warning */
  UNUSED(GPIO_Pin);
  /* NOTE: This function Should not be modified, when the c
allback is needed,
           the HAL_GPIO_EXTI_Callback could be implemented
in the user file
   */
}
```

Переопределим тело функции

Такая обработка прерываний через переопределение тела функции наиболее корректна, так как не затрагиваются файлы библиотек и уменьшается вероятность ошибки в случае переноса проекта. (прерывания по таймеру так же имеют такую функцию)

При нажатии кнопки может возникнуть дребезг контактов. В данном случае он может быть критичен. Для борьбы с этим нежелательным эффектом используется проверка того, что от предыдущего нажатия кнопки уже прошло 50мс.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  if (GPIO_Pin == GPIO_PIN_6)
    if (time_after_btn > 50){
      time_after_btn = 0;
      flag_timer_pause = !flag_timer_pause;
    }
}
```

В таймере увеличивает счетчик времени после последнего нажатия кнопки

```
void TIM2_IRQHandler(void)
{
  ///……
  if (time_after_btn < 51)
    time_after_btn++;
  ///……
  /* USER CODE END TIM2_IRQn 1 */
}
```

Добавим управление индикацией

```
void EXTI3_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI3_IRQn 0 */

  /* USER CODE END EXTI3_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_3);
  /* USER CODE BEGIN EXTI3_IRQn 1 */

  time_s = 0;
  time_sec_ctr = 0;
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, 1);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, 1);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, 1);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, 1);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, 1);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, 1);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, 1);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, 1);
  /* USER CODE END EXTI3_IRQn 1 */
}
```

```c
void TIM2_IRQHandler(void)
{
  /* USER CODE BEGIN TIM2_IRQn 0 */

  /* USER CODE END TIM2_IRQn 0 */
  HAL_TIM_IRQHandler(&htim2);
  /* USER CODE BEGIN TIM2_IRQn 1 */

  if (glob_time > 500)
  {
    glob_time = 0;
    HAL_GPIO_TogglePin(Led_STATUS_GPIO_Port, Led_STATUS_Pin);
  } else glob_time++;

  if (time_after_btn < 51)
    time_after_btn++;

  if ((time_sec_ctr >= 1000) & (~flag_timer_pause)) {
    time_s++;
    time_sec_ctr = 0;

      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, !((time_s & 1)));
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, !((time_s >> 1) & 1));
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, !((time_s >> 2) & 1));
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, !((time_s >> 3) & 1));
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, !((time_s >> 4) & 1));
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, !((time_s >> 5) & 1));
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, !((time_s >> 6) & 1));
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, !((time_s >> 7) & 1));

    if (time_s >= 256) time_s = 0;
  } else time_sec_ctr++;

  /* USER CODE END TIM2_IRQn 1 */
}
```

ГУАП

STM & XILINX

Спасибо за внимание,
спасибо за старания!

GitHub https://github.com/v-crys/course_stm32_fpga