Лабораторна робота №3

ПЕРЕРИВАННЯ. РОБОТА З ТАЙМЕРАМИ

Мета: отримати базові навички організації паралельних процесів з використанням переривань. Ознайомитись з особливостями конфігурації та використання таймерів.

Теоретичні відомості

Грамотним підходом до створення працездатних програм для мікроконтролерів є використання паралельних процесів. Уявімо ситуацію, коли існує задача, наприклад, дізнатися стан кнопки – як у попередній лабораторній роботі. Для цього викликається функція, яка повертає результат. Але для обробки сигналу кнопки потрібен певний час, зазвичай кілька мілісекунд. Протягом цього часу основна програма не працює, не виконує ніяких дій. Але що робити, якщо стан кнопки необхідно знати постійно? Наприклад, у стані очікування, коли вона буде натиснута. Ще складнішою буде ситуація, якщо таких кнопок декілька. Якщо почергово викликати функцію їх опитування, то велику частину часу основна програма буде на паузі, що призведе до пропуску інших подій. Виходом є обробка завдань паралельними процесами. Паралельні процеси – це дії, які виконуються одночасно. В реальності, повністю одночасно завдання можна виконувати тільки на мультипроцесорних системах. Але реальні завдання не вимагають для роботи всього процесорного часу. Одним із механізмів організації паралельних процесів є використання переривань. Переривання (англ. Interrupt) – це створюваний певним чином процес, який тимчасово перемикає мікропроцесор на виконання іншої програми з наступним відновленням роботи програми, яка була перервана. Перериванням в мікроконтролерах відведена особлива роль. Вони дозволяють мікроконтролеру моментально реагувати на певні події, підвищуючи тим самим його оперативність. Переривання виникає в результаті настання певної очікуваної події. В такому випадку, мікроконтролер тимчасово припиняє виконання основного коду програми та переходить до виконання підпрограми, яка знаходиться в обробнику переривання (англ. Handler). Після завершення цієї підпрограми продовжується виконання основного коду програми. Цей принцип умовно показано на рис. 3.1.

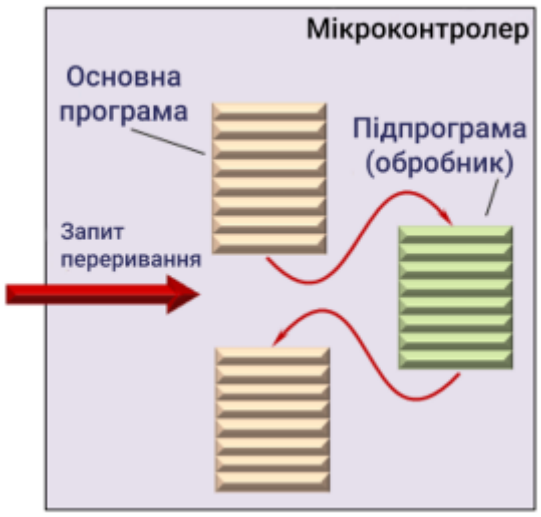

Рис. 3.1. Переривання

Існує поняття зовнішніх і внутрішніх переривань. Зовнішнє переривання викликається зовнішньою подією (наприклад, натисненням кнопки користувачем або замиканням контактів датчика). Внутрішнє переривання обумовлено деякою подією, пов'язаною, наприклад, з прийомом або передачею даних через інтерфейс, завершенням рахунку таймером тощо. Також розрізняють апаратне і програмне переривання. Апаратне переривання виникає в результаті

впливу зовнішніх сигналів або апаратної установки регістрових прапорців подій. Програмне переривання формується за допомогою коду програми шляхом програмної установки прапорців подій або безпосереднього виклику підпрограми обробки переривання. У прикладі з кнопкою, можна запрограмувати апаратний таймер так, що сигнал переривання з нього буде надходити циклічно, із заданим періодом. Відповідно, циклічно відбуватимуться переривання по таймеру. Таким чином, до програми додається функція, яка сама по собі викликається, наприклад, кожну 1 мс. У цій функції розміщується алгоритм обробки сигналу (наприклад, фільтрація брязкоту аналогічно до функції button_pressed() із попередньої лабораторної роботи). В результаті, в основній програмі можуть постійно виконуватись необхідні дії, які з періодом 1 мс будуть перериватися на короткий час для обробки сигналу кнопки. А про стан кнопки можна судити за програмними ознаками (наприклад, за станом змінної flag). Обробка сигналу кнопки в цьому випадку відбувається у фоновому режимі, непомітно для основної програми. Отже, апаратне переривання – це сигнал про певну подію. У разі появи такого сигналу виконання програми зупиняється, і управління переходить до обробника переривання. Після завершення його роботи управління повертається до коду основної програми, на те саме місце, де вона була перервана. З точки зору програми переривання – це виклик функції по зовнішній, не пов'язаній безпосередньо з програмним кодом, події. Під час визначення поняття "переривання" слово "подія" до цього моменту вживалось в загальноприйнятому значенні. Тобто ситуація, коли щось трапилося. Але в системі STM32 поряд з перериванням (interrupt) існує ще одне суворе поняття – апаратна подія (event). Їх часто плутають:

- Event – це подія (апаратна або програмна), на яку можуть реагувати ядро або периферійні блоки. Іншими словами, це щось, що сталося з апаратним вузлом мікроконтролера. Наприклад, переповнився таймер, на зовнішньому вході змінився рівень сигналу тощо. Одним з варіантів реакції на подію може бути переривання.
- Interrupt – це переривання роботи програми і перехід управління в спеціалізовану область – обробник переривання.

Апаратна подія – це те, що може викликати переривання. Крім переривань, події можуть активувати і інші можливості мікроконтролера. Кожне переривання викликається подією, але не кожна подія викликає переривання.

- Переривання без події неможливі.
- Події без переривань можливі.

Окремо виділяють зовнішні переривання. Зовнішні переривання (англ. External interrupts, EXTI) – це такі переривання, які обробляються внаслідок виникнення деяких подій на певному виводі порту мікроконтролера (рис. 3.2). Подій може бути декілька, не дивлячись на всього 2 логічних стани. Їх можна розділити на 2 групи:

1) Обробка зовнішніх переривань (External interrupts);

2) Обробка зовнішніх подій (External events).

Різниця тут лише в тому, що в першому випадку викликається обробник переривань, а в другому – тільки піднімається відповідний прапорець. Кожна група ділиться на три види подій:

1. Виявлено висхідний (передній) фронт (зміна рівня 0 на рівень 1);

2. Виявлено спадний (задній) фронт (зміна рівня 1 на рівень 0);

3. Виявлено обидва вказані фронти.

Одночасно існує можливість обробити 16 виводів контролера, але всі вони повинні бути з різними номерами. Тобто можна обробити одночасно ніжки PA1 і PC2, але неможливо обробити PA1 і PC1. Окрім виводів портів GPIO, існує можливість обробити зовнішні події від програмованого детектора напруг, від годинника реального часу (RTC), від подій на інтерфейсах USB та Ethernet.

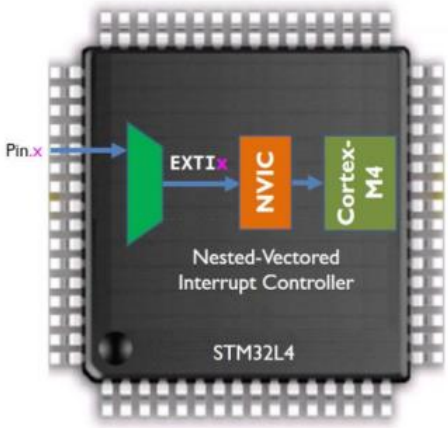

Рис. 3.2. Зовнішнє переривання на виводі мікроконтролера

Обробка і управління перериваннями виконується апаратним вузлом мікроконтролера – контролером пріоритетних векторних переривань (англ. Nested Vectored Interrupt Controller, NVIC). У разі виникнення дозволеної події контролер перериває виконання програми, зберігає в стеку необхідні дані і передає керування за адресою функції обробки переривання. Після виходу з функції-обробника управління повертається на адресу, за якою була перервана основна програма. Схема обробки переривань показана на рис. 3.3.

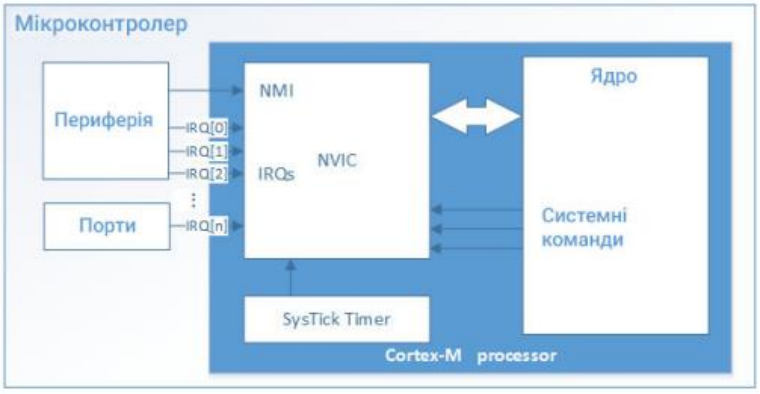

Рис. 3.3. Схема обробки переривань в мікропроцесорах Cortex

У випадку ініціації переривання NVIC перемикає ядро в режим обробки переривання. Після переходу в режим обробки переривання регістри ядра поміщаються до стеку. Безпосередньо під час запису значень регістрів до стеку здійснюється вибірка початкової адреси функції обробки переривання. По завершенню обробки переривання всі дії виконуються у зворотному порядку: зчитується вміст стека і, паралельно, здійснюється вибірка адреси повернення. Усі можливі переривання, підтримувані NVIC, записуються в таблицю векторів переривань. По суті, таблиця векторів переривань є списком адрес функцій обробників переривань. Номер у списку відповідає номеру переривання. У мікроконтролерах STM32 підтримується до 68 апаратних переривань. Кожному з них відповідає своя адреса-вектор. За цією адресою передається керування програми у разі виникненні переривання. Конкретні адреси для кожного джерела можна подивитися в документації на мікроконтролер

Основні властивості NVIC:

- Контролер пріоритетний. Тобто обробка переривання може бути перервана іншим перериванням з більш високим пріоритетом. Переривання з таким же або більш низьким пріоритетом чекатимуть закінчення обробки активного переривання. Пріоритети можна розбивати на групи і підгрупи.
- Для кожного переривання може бути заданий пріоритет. Це число від 0 до 15, 0 – найвищий пріоритет. Для установки пріоритетів існує спеціальний блок регістрів.
- Можуть бути заборонені одразу всі переривання – глобальна заборона.
- Можуть бути заборонені або дозволені окремі переривання. Для цього існує регістр дозволу або маски переривань.
- Існує блок регістрів очікування обробки переривань, кожен біт яких відповідає конкретному перериванню і показує, чи очікує воно обробки.
- Є регістри діючих переривань, які показують, чи знаходяться переривання в стадії обробки.
- Будь-яке переривання може бути викликано програмно. Для цього існує регістр програмного виклику переривань. Переривання викликається записом в нього відповідного номера.
- Кожному з переривань відповідає своя фіксована адреса-вектор, за якою передається керування для обробки переривання.

Для роботи з апаратними перериваннями обов'язково необхідно виконати наступні дії:

1. Дозволити потрібне переривання в контролері переривань. Якщо необхідно, то задати для нього пріоритет.

2. Дозволити глобальні переривання мікроконтролера.

3. Налаштувати пристрій (наприклад, таймер) на формування запиту переривання – події.

4. Створити функцію обробки переривання.

Для організації паралельних процесів часто необхідно формувати циклічні переривання з заданим періодом. Найбільш простим способом є робити це за допомогою апаратних таймерів. Таймери STM32 – багатофункціональні пристрої. За допомогою кожного з них можна реалізувати:

- Лічильник імпульсів (а значить і часу) з автоматичним перезавантаженням.
- Захоплення вхідного сигналу (4 канали);
- Виявлення фронту вхідного сигналу, запам'ятовування часу, генерацію події;
- Вимірювання часових параметрів вхідного ШІМ-сигналу: періоду і тривалості імпульсів;
- Інтерфейс енкодера. Вимірювання параметрів імпульсів енкодера;
- Порівняння кодів таймера (4 канали);
- Генерація події за випадковим збігом коду таймера з заданим значенням;
- Формування ШІМ-сигналу;
- Формування одиночних імпульсів.

У STM32F411VET6 є **10 таймерів**, які поділяються на **загального призначення, основні (базові) та спеціалізовані (розширені)**.

TIM1 – 16-бітний, підтримує ШІМ, одно- та багатофазний режим, може працювати для керування двигунами.

TIM2, TIM3, TIM4, TIM5 – 16-бітні, підтримують лічильник, вимірювання часу, генерацію сигналів, ШІМ, затримки.

TIM6, TIM7 – прості 16-бітні таймери, зазвичай використовуються для системних затримок або керування DAC.

TIM9, TIM10, TIM11 – 16-бітні таймери, які використовуються для генерації точних імпульсів та ШІМ.

Всі таймери мають однакову архітектуру. Розширений таймер відрізняється наявністю додаткових апаратних вузлів для формування протифазних сигналів ШІМ. В результаті його можна конфігурувати на роботу в режимі 6-канального ШІМ і керувати ним трьома напівмостовими підсилювачами потужності. Але для простих задач, таких як формування циклічних переривань, всі таймери мають однакову архітектуру. У таймерів великий вибір джерел тактування, є переддільники, забезпечується цифрова фільтрація сигналів, можлива синхронізація між собою тощо. Але зараз буде розглядатись лише режим лічильника з перезавантаженням. Саме в цьому режимі найзручніше формувати циклічні переривання.

Функціональна схема таймера досить складна. Її частина, необхідна для організації циклічних переривань, показана на рис. 3.4. З повною схемою можна ознайомитись в документації до мікроконтролера.

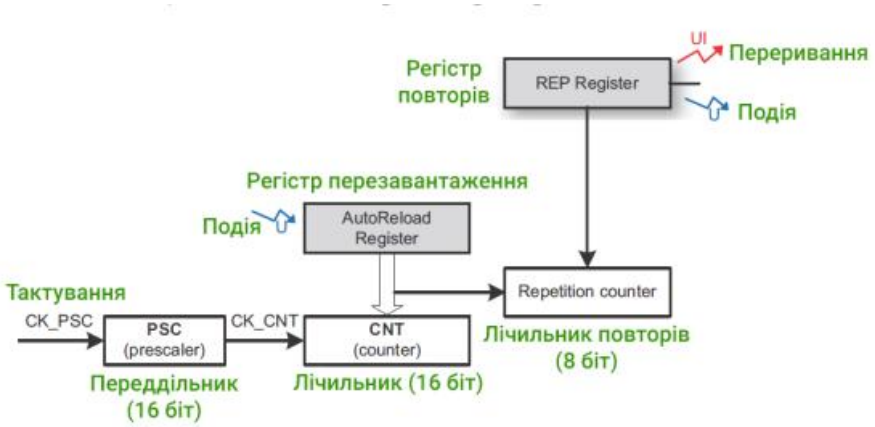

Рис. 3.4. Частина блок-діаграми таймера загального призначення

Відлік імпульсів або часу відбувається на 16-ти розрядному лічильнику CNT. Коли код лічильника досягає значення регістра перезавантаження, лічильник скидається в 0. Таким чином, лічильник рахує за циклом від 0 до значення регістра перезавантаження. Частота сигналу тактування таймера може бути зменшена за допомогою 16-ти розрядного переддільника PSC.

Перезавантаження лічильника формує подію (переривання). Частота її появи також може бути зменшена лічильником повторів (8 розрядів). Коефіцієнт ділення задається в регістрі повторів. Код лічильника використовується іншими вузлами таймера, наприклад, для формування ШІМ.

В якості джерела тактування можуть бути обрані:

- Внутрішні синхросигнали шин APB1 і APB2.
  - Для таймера TIM1 використовується синхросигнал шини APB2;
  - Для таймерів TIM2-TIM4 використовується синхросигнал шини APB1.
- Зовнішнє тактування, режим 1. Використовується вихідний сигнал іншого таймера або зовнішній сигнал входів захоплення.
- Зовнішнє тактування, режим 2. Використовується вивід мікроконтролера ETR.

- Особливий режим синхронізації - інтерфейс підключення енкодера.

Під час використання таймера в якості лічильника імпульсів можна вибрати один з режимів підрахунку:

- прямий підрахунок;
- зворотній підрахунок;
- двонаправлений.

У випадку прямого підрахунку вміст лічильника з кожним імпульсом тактування збільшується на 1. Коли він досягає значення регістра перезавантаження, то лічильник скидається. Таким чином, таймер рахує за циклом від 0 до значення перезавантаження. У момент перезавантаження формується переривання.

У режимі зворотної (реверсивного) підрахунку з кожним вхідним імпульсом вміст лічильника зменшується на 1. При досягненні 0 в лічильник завантажується значення регістра перезавантаження і реверсивний підрахунок триває. Таймер рахує за циклом від значення перезавантаження до 0. У момент перезавантаження формується переривання.

Двонаправлений режим означає, що лічильник рахує в прямому напрямку від 0 до значення перезавантаження, а потім переходить в реверсивний режим і підрахунок ведеться до 0. Під час зміни напрямку підрахунку і скидання генерується переривання.

Детально розглянемо деякі налаштування таймерів.

Prescaler (PSC). Це регістр переддільника. Переддільник ділить частоту тактування таймера, яка надходить на основний лічильник. По суті, Prescaler разом із вхідною частотою визначає роздільну здатність таймера.

Лічильник переддільника рахує вхідні імпульси від 0 до значення цього регістра. У разі рівності кодів лічильника і регістра, лічильник скидається і починає рахувати заново. У момент скидання формується імпульс тактування основного лічильника таймера. Таким чином, значення регістра переддільника визначає коефіцієнт ділення частоти вхідного сигналу.

Лічильник і регістр переддільника 16-ти розрядні. Тобто максимальний коефіцієнт ділення 65536.

Треба пам'ятати, що реальний коефіцієнт ділення на 1 більше, ніж значення регістра переддільника. Наприклад:

Значення регістру переддільника	Коефіцієнт ділення
0	1
999	1000
65535 (максимальне значення)	65536

Counter mode. Режим підрахунку, що визначає, в який бік рахувати. Доступні варіанти:

− Up - прямий підрахунок.

− Down - зворотній (реверсивний) підрахунок.

− Center Aligned mode 1 - двонаправлений підрахунок, переривання генерується в момент, коли лічильник рахує в зворотну сторону і доходить до 0.

− Center Aligned mode 2 - двонаправлений підрахунок, переривання генерується, коли лічильник рахує в прямому напрямку і досягає значення перезавантаження.

− Center Aligned mode 3 - двонаправлений рахунок, переривання генерується, в обох випадках - при досягненні 0 і значення перезавантаження.

Counter Period (Auto Reload Register). Регістр перезавантаження. Його значення задає період роботи таймера. Потрібно пам'ятати, що на час періоду впливає також і режим підрахунку. Лічильник 16-ти розрядний. Це означає, що для односпрямованого підрахунку період може тривати від 1 до 65536 довжин імпульсів переддільника. Реальна тривалість періоду на 1 більше значення регістра перезавантаження (так само, як і для регістра переддільника)

Internal Clock Division (CKD). Подільник вхідної частоти для внутрішніх потреб таймера. Отримана частота використовується під час фільтрації зовнішніх сигналів, формування "мертвого часу" ШІМ тощо.

Repetition Counter (RCR). Регістр лічильника повторів. Присутній не в усіх таймерах. Лічильник повторів рахує імпульси подій на виході таймера і при 53 досягненні значення регістра повторів скидається та формує реальну подію. Тобто він ділить частоту генерації подій (переривань) таймера. Лічильник 8-ми розрядний. Коефіцієнт ділення на 1 більше значення регістра повторів і може бути в діапазоні 1 - 256.

Auto-reload preload. Регістр перезавантаження буферизований. Розробники мікроконтролера надають програмісту вибір – під час запису значення перезавантаження передавати його в регістр моментально або дочекатися крайнього стану лічильника.

Хід виконання роботи

1. Виконати попередні налаштування плати «STM32F4 - DISCO» для керування вбудованим та зовнішнім світлодіодами із використанням таймеру та кнопки. Схема підключення елементів показана на рис. 3.5.

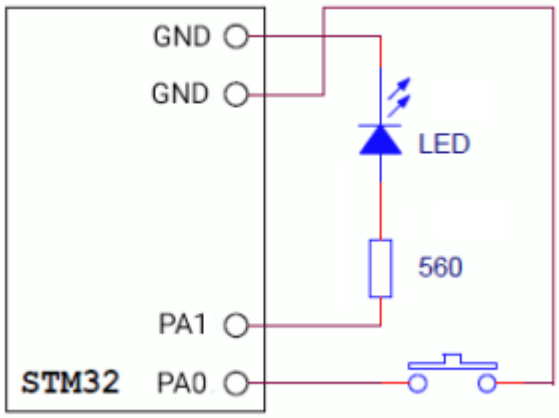

Рис. 3.5. Підключення зовнішнього світлодіоду та кнопки

1.1. Запустити середовище програмування STM32CubeIDE та створити новий проект.
1.2. Самостійно провести початкові налаштування виводів GPIO для підключення кнопки та користувацького світлодіода (за аналогією до попередніх робіт).
1.3. Встановити джерело тактування від зовнішнього кварцового резонатора та режим відлагодження «Serial Wire».
1.4. Налаштувати вивід підключення зовнішнього світлодіода PA1 для роботи в режимі активного виходу. Встановити на виводі низький рівень за замовчуванням та підтягування до землі.
1.5. Для обох активованих виходів (PA1, PA9) встановити швидкість роботи «High».
1.6. На вкладці Clock Configuration налаштувати тактування мікроконтролеру. Встановити тактування від зовнішнього кварцового резонатора та основну частоту тактування 100

МГц. Переконатись, що частота тактування таймерів на шині APB1 встановлена на рівні 100 МГц (рис. 3.6).

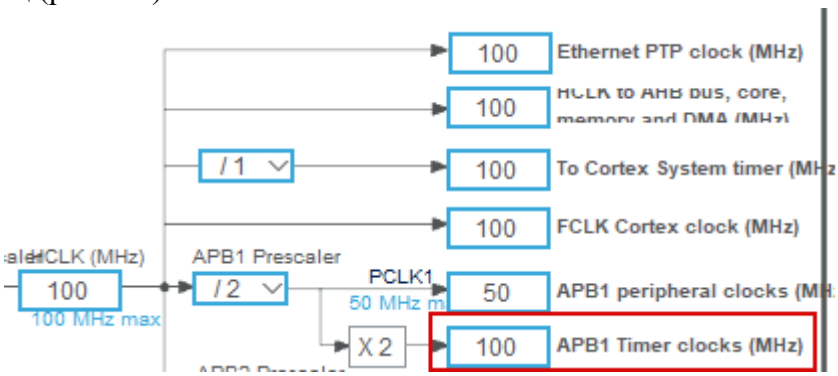

Рис. 3.6. Частоти тактування на шинах APB1 та APB2

1.7. На вкладці Pinout & Configuration в блоці Timers активувати таймер загального призначення TIM2. Для цього обрати даний таймер у випадаючому меню та встановити наступні налаштування (рис. 3.7):

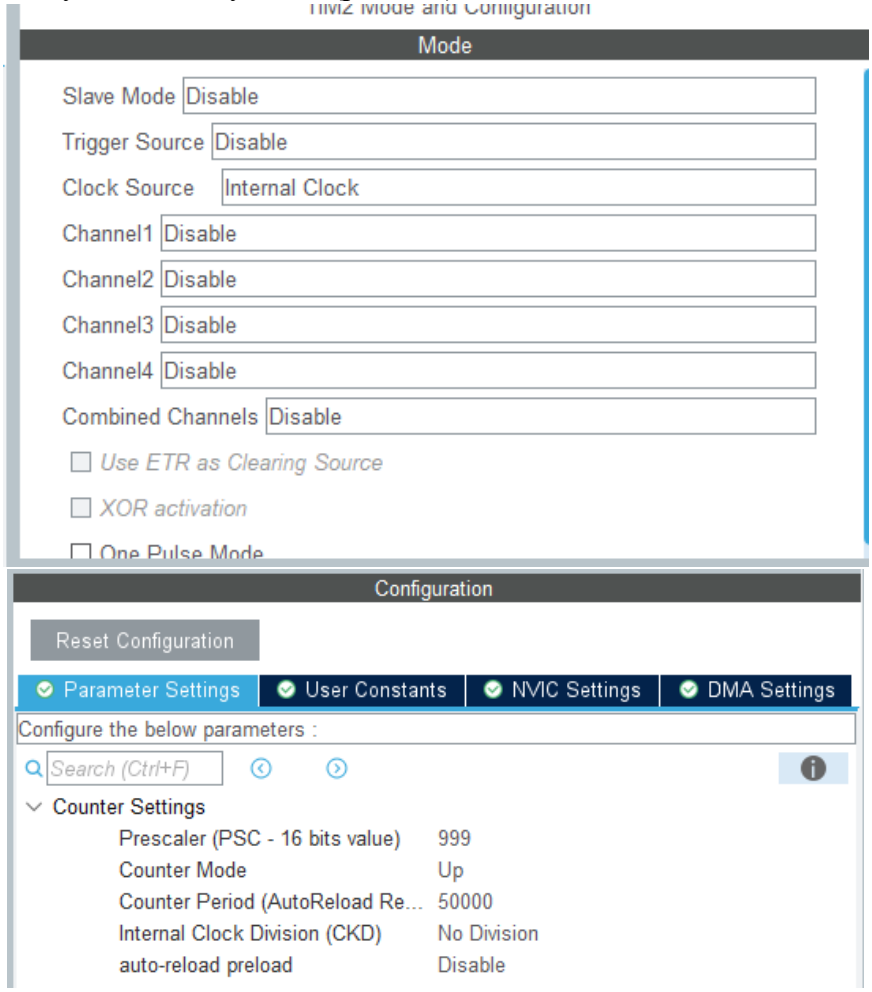

Рис. 3.7. Налаштування таймера для генерування переривання з періодом 0,5 с

Частота тактування таймера попередньо була встановлена на рівні 100 мГц. За допомогою регістра переддільника Prescaler (PSC) дана частота округлюється до більш зручного значення. В нашому випадку, якщо задати значення PSC = 1000 - 1 = 999, то частота після переддільника буде дорівнювати 100 000 000/1000 = 100 000 Гц, що відповідає періоду в 10 мкс. Далі в регістр перезавантаження AutoReload Register записується значення 50 000. Таймер буде рахувати імпульси з періодом в 10 мкс, поки не досягне даного значення. В результаті, отримаємо необхідний період переповнення таймеру: 50 000 * 10 мкс = 500 мс = 0,5с.

1.8. На вкладці конфігурації таймера NVIC Settings дозволити переривання за таймером TIM2 (рис. 3.8).

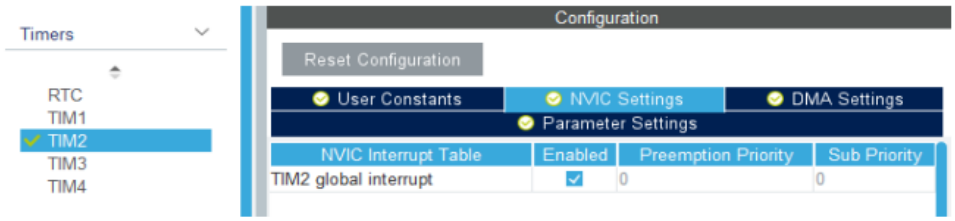

Рис. 3.7. Увімкнення переривань за таймером TIM2

1.9. Провести генерацію коду. Самостійно ознайомитись із отриманим кодом ініціалізації у файлі main.c.

1.10. Підключити плату STM32 «STM32F4 - DISCO» до ПК. Провести компіляцію проекту та завантажити програму до мікроконтролера. Критерієм правильності налаштувань будуть вимкнені користувацький та зовнішній світлодіоди.

2. Написати програму для почергового блимання користувацьким та зовнішнім світлодіодами з періодом 0,5 с. Для переключення стану світлодіодів використовувати переривання від таймера TIM2.

2.1. Написати функцію – обробник переривань. Для цього в директорії проекту Core –> Src відкрити файл stm32f4xx_it.c. Це файл, який містить код для роботи з перериваннями.

2.2. В кінці файлу знайти функцію TIM2_IRQHandler() – це і є обробник переривань таймера TIM2. Модифікувати її наступним чином:

```
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 BEGIN TIM2_IRQn 1 */
  if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_9) == HAL_GPIO_ReadPin(GPIOA,
 GPIO_PIN_1))
  {
   HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_9);
  }
  else
  {
   HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_9);
   HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_1);
  }

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

Дана функція буде викликатись у випадку виникнення переривання за таймером TIM2. В нашому випадку, кожні 0,5 секунд. Оскільки на початку роботи програми обидва світлодіоди вимкнені, у функції передбачена зміна стану користувацького світлодіода за даних умов (так як світлодіоди мають засвічуватись послідовно, а не одночасно). У подальшому виконується другий блок коду, в якому просто відбувається перемикання стану обох світлодіодів.

2.3. Повернутися до файлу main.c та запустити таймер TIM2 в режимі генерації переривань. Для цього використовується функція HAL_TIM_Base_Start_IT(), яку

необхідно розмістити в блоці Initialize all configured peripherals. В якості аргумента до функції передається вказівник на структуру параметрів конфігурації таймера – &htimX, де X – номер таймера. В нашому випадку, код запуску таймеру 2 в режимі генерації переривань буде виглядати наступним чином:

```
/* Initialize all configured peripherals */
 HAL_TIM_Base_Start_IT(&htim2);
 /* USER CODE BEGIN 2 */
```

2.4.. Основний цикл while(1) в даному випадку порожній. Програма просто знаходиться в ньому. По відношенню до основної програми, світлодіоди блимають в фоновому режимі, самі по собі. Якщо у циклі будуть виконуватись якісь дії, це ніяк не позначиться на рівномірному миготінні світлодіодів. Єдина умова – не вимикати переривання.

2.5.Завантажити програму до мікроконтролера та перевірити правильність її роботи (рівномірне почергове запалення світлодіодів з інтервалом 0,5 с).

3. Модифікувати програму таким чином, щоб світлодіоди миготіли лише тоді, коли натиснута кнопка.

3.1 Виконати дії для запуску таймера лише у випадку натиснутої кнопки. Для цього необхідно перемістити функцію старту таймера в режимі генерації переривань HAL_TIM_Base_Start_IT() з блоку ініціалізації Initialize all configured peripherals до основного циклу while(1) та написати відповідну умову увімкнення таймера. Якщо умова (кнопка натиснута) не виконується, необхідно вимкнути таймер за допомогою зворотної функції HAL_TIM_Base_Stop_IT() та повернути світлодіоди до початкового (вимкненого) стану. Це можна зробити з використанням наступного коду:

```
while (1)
{
	if(!HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0))
	 {
	HAL_TIM_Base_Start_IT(&htim2);
	 }
	 else
	 {
		HAL_TIM_Base_Stop_IT(&htim2);
		HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_RESET);
		HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET);
	 }
}
```

3.2.Завантажити програму до мікроконтролера та перевірити правильність її роботи. Світлодіоди мають миготіти лише коли кнопка натиснута. У разі відпускання кнопки світлодіоди мають одразу гаснути.

4. Модифікувати програму таким чином, щоб натискання на кнопку викликало зовнішнє переривання (EXTI) для увімкнення/вимкнення режиму миготіння світлодіодів. Передбачити алгоритм фільтрації брязкоту.

4.1 Налаштувати вивід PA0, до якого підключено кнопку, на роботу в режимі генерації зовнішніх переривань. Для цього відкрити вкладку з конфігурацією мікроконтролера (файл *Назва проекту*.ioc) та на графічній моделі мікроконтролера змінити призначення виводу PA0 на генерацію зовнішніх переривань за лінією 0, обравши режим «GPIO_EXTI0». В налаштуваннях виводу обрати генерацію зовнішніх переривань за висхідним та спадним фронтами. Встановити підтягування до низького рівня (рис. 3. 8).

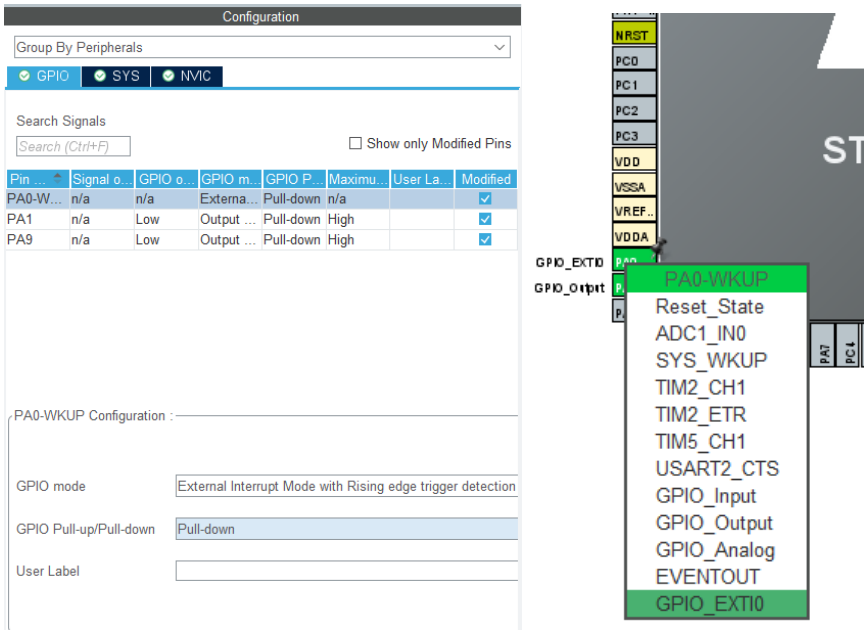

Рис. 3.8. Налаштування виводу для генерації EXTI за обома фронтами

4.2. На вкладці NVIC налаштувань виводів GPIO дозволити зовнішні переривання за лінією 0 (рис. 3.9).

4.3. Зберегти проект та повторно згенерувати код. Перевірити правильність згенерованого коду (попередній код користувача має залишитись після регенерації).

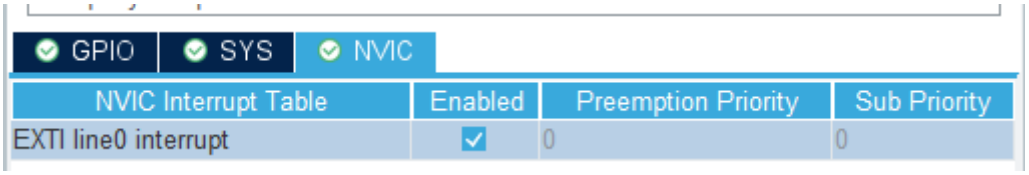

Рис. 3.9. Встановлення дозволу на зовнішні переривання

4.4. Алгоритм програмної фільтрації механічного брязкоту із використанням зовнішніх переривань передбачає виконання наступних кроків:

1) Створюються три змінні з кваліфікатором volatile, в яких будуть зберігатися ознака дії (в нашому випадку – чи повинні миготіти світлодіоди), ознака стану фільтрації брязкоту та момент виникнення переривання з метою подальшого формування часової затримки для фільтрації брязкоту.

2) Створюється функція HAL_GPIO_EXTI_Callback(). Дана функція повинна містити код, який буде виконано за запитом обробника переривання у разі його виникнення. Функцію потрібно розташувати в файлі main.c у блоці USER CODE BEGIN 4

3) Як тільки кнопка натискається, переривання на даному виводі забороняються. Це можна зробити за допомогою функції HAL_NVIC_DisableIRQ(), вказавши в якості аргументу необхідну лінію. Після цього мікроконтролеру байдуже, бряжчить кнопка чи ні, оскільки переривання відключені і, відповідно, реакції на них немає.

4) Ознака стану фільтрації брязкоту стає активною.

5) Фіксується момент старту фільтрації брязкоту (момент виникнення переривання). Це можна зробити на за допомогою функції HAL_GetTick(), яка

повертає поточну кількість мілісекунд, що пройшли з моменту запуску мікроконтролера.

6) Змінюється стан ознаки дії (в нашому випадку – активна ознака дії буде вмикати миготіння світлодіодами в головному циклі while(1), а неактивна – вимикати його).

7) В нескінченному циклі while(1) перевіряється ознака стану фільтрації брязкоту та час, який пройшов з моменту виникнення переривання. Якщо час перевищує задане значення затримки (як правило, 30-300 мс), фільтрація вважається завершеною.

8) Для виходу із режиму фільтрації необхідно:

− Очистити відповідний виводу біт регістру EXTI_PR. Якщо переривання дозволено в контролері переривань NVIC, то установка біта в регістрі EXTI_PR призведе до виклику відповідного обробника переривання. Після виклику обробника відповідний біт не скидається автоматично, це потрібно зробити вручну, інакше одразу після виходу з обробника переривання він буде викликаний знову, і так до нескінченності.

− Очистити відповідний лінії біт регістру NVIC_ICPRx. Біти даного регістру вказують на те, що на певній лінії виникло переривання, яке ще не опрацьоване. Відповідно, якщо не скинути вказаний біт, переривання буде автоматично запускатись знову.

− Увімкнути зовнішні переривання на лінії. Це можна зробити за допомогою функції HAL_NVIC_EnableIRQ(), до якої в якості аргументу передати номер відповідної лінії.

− Скинути ознаку фільтрації брязкоту до вимкненого стану.

4.5.Для реалізації описаного алгоритму, в файлі main.c створити наступні змінні:

```
/* USER CODE BEGIN PV */
volatile uint8_t blink_on = 0;
volatile uint8_t flag_irq = 0;
volatile uint32_t time_irq = 0;
/* USER CODE END PV */
```

4.6.Написати функцію обробки переривання по лінії 0. Дана функція буде автоматично викликатись обробником переривань. Код функції:

```
/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
if(GPIO_Pin == GPIO_PIN_0)
{
      HAL_NVIC_DisableIRQ(EXTI0_IRQn); // одразу вимикаємо переривання на виводі
      flag_irq = 1;     //вмикаємо ознаку фільтрації брязкоту
      time_irq = HAL_GetTick(); //фіксуємо момент увімкнення фільтрації
      blink_on = !blink_on;   //змінюємо ознаку стану миготіння світлодіодів
}
}
/* USER CODE END 4 */
```

4.7.Модифікувати код головної програми наступним чином:

```
/* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */
```

```
/* USER CODE BEGIN 3 */
// вмикаємо/вимикаємо миготіння в залежності від стану ознаки  blink_on
if(blink_on)
{
HAL_TIM_Base_Start_IT(&htim2);
}
else
{
HAL_TIM_Base_Stop_IT(&htim2);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_RESET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET);
}
// завершуємо фільтрацію брязкоту, якщо пройшло 200 мс
if(flag_irq && (HAL_GetTick() - time_irq) > 200)
{
  __HAL_GPIO_EXTI_CLEAR_IT(GPIO_PIN_0);  // очищуємо біт EXTI_PR
  NVIC_ClearPendingIRQ(EXTI0_IRQn); // очищуємо біт NVIC_ICPRx
  HAL_NVIC_EnableIRQ(EXTI0_IRQn);   // вмикаємо зовнішні   переривання
  flag_irq = 0; //вимикаємо ознаку фільтрації брязкоту
 }
}
/* USER CODE END 3 */
```

4.8. Завантажити програму до мікроконтролера та перевірити правильність її роботи. Коротке натискання на кнопку має вмикати режим миготіння світлодіодів, повторне натискання – вимикати його. У разі довгого натискання, світлодіоди мають миготіти, доки кнопка натиснута.

Індивідуальні завдання

1. Модифікувати програму таким чином, щоб світлодіоди запалювались/загасали одночасно, а період блимання становив 1 с.

2. Модифікувати програму таки чином, додати 4 режими роботи. Перший режим це п.1 індивідуального плану. Другий режим, це реалізація «біжучий вогник» на 4 світлодіодах котрі розташовані по колу на платі та режим миготіння зовнішнього та вбудованого світлодіода, причому частота миготіння одного 0,5с, а іншого 1с. 4ий режим зупиняє усі таймери та вимикає усі світлодіоди. Реалізація кожного режиму в окремому таймері(окрім вимикання). При переході на інший режим таймер попереднього зупиняються. Переключення режимів відбувається натисканням користувацької кнопки використовуючи зовнішнє переривання.

Зміст звіту

1. Тема та мета лабораторної роботи.
2. Скріншот конфігурації мікропроцесора (файл .ioc), налаштування кожного таймера(3шт) та зовнішнього переривання EXTI0.
3. Код файлу main.c функції main та HAL_GPIO_EXTI_Callback та код файлу stm32f4xx_it.c по кожному перериванню (4 шт).
4. Висновки.

Контрольні запитання

1. Що таке «переривання»? Поясніть мету та призначення переривань.
2. Опишіть основні види переривань.
3. Поясніть різницю між поняттями «переривання» та «подія».
4. Розкрийте поняття «зовнішні переривання». Які особливості EXTI?
5. Що таке NVIC? Поясніть принципи роботи NVIC.
6. Назвіть основні властивості NVIC.
7. Що таке таймери? Поясніть призначення та можливості таймерів.
8. Поясніть призначення та особливості роботи з регістром Prescaler.

9. Назвіть основні режими підрахунку таймера в режимі лічильника.
10.Поясніть призначення регістра Auto Reload Register.
11.Яким чином необхідно налаштувати таймер для генерації циклічних переривань з періодом 2 с?
12.Опишіть алгоритм фільтрації брязкоту з використанням переривань.