Лабораторна робота №2

ЗЧИТУВАННЯ ІНФОРМАЦІЇ З ДИСКРЕТНИХ ЛІНІЙ ПОРТІВ ЗАГАЛЬНОГО ПРИЗНАЧЕННЯ

Мета: отримати базові навички роботи з портами вводу/виводу загального призначення (GPIO) мікропроцесорної системи на базі ядра ARM Cortex-M4. Ознайомитись з особливостями підключення зовнішніх цифрових дискретних пристроїв введення до портів загального призначення.

Теоретичні відомості

З метою підключення зовнішніх пристроїв введення (кнопок, клавіатури тощо) або цифрових датчиків/вузлів, виводи портів загального призначення GPIO мають бути налаштовані для роботи в режимі входів. Загальна функціональна схема одного виводу GPIO показана на рис. 2.1. Червоним виділена частина, яка відповідає режиму роботи виводу на вхід (input driver).

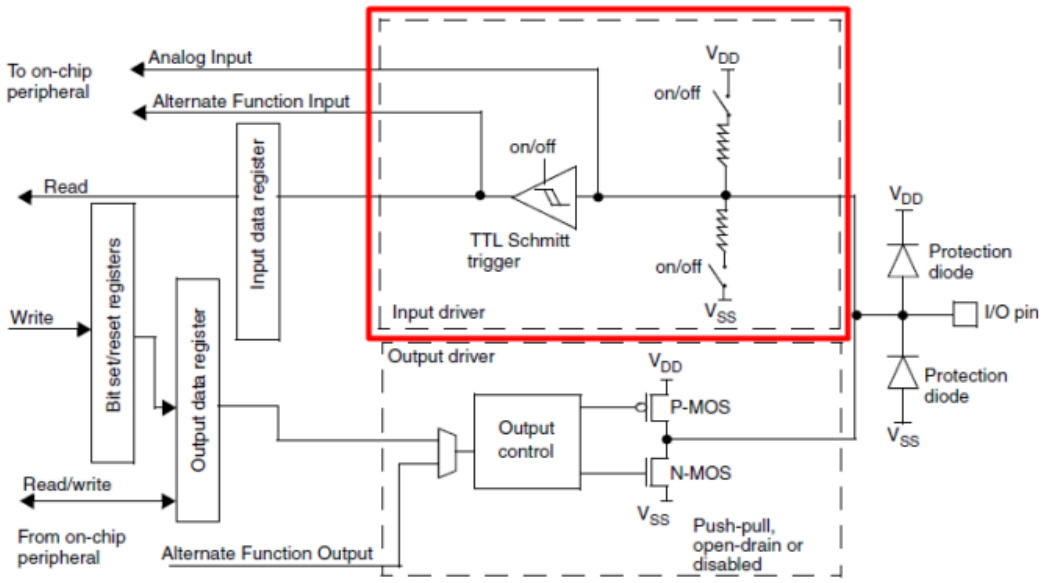

Рис. 2.1. Базова схема виводу загального призначення (виділено вхід)

Відповідно до схеми, до виводу підключені 2 захисних діоди: один на землю Vss, інший на шину живлення Vdd. Перший діод замикає через себе вхідну напругу негативної полярності, другий обмежує її на рівні Vdd (3,3 В). Сигнали з напругою, яка перевищує ці межі, повинні підключатися через обмежувальні резистори. Вхідний струм не повинен перевищувати ± 5 мА – це граничний струм захисних діодів. Також необхідно пам'ятати, що вхідний струм через захисні діоди потрапляє на джерело живлення плати. Якщо він перевищує струм споживання плати, то напруга живлення мікроконтролера буде підвищуватися і може досягти неприпустимого значення. Це стандартна проблема в подібних схемах. Але більша частина входів мікроконтролера STM32F411VET6 допускає подачу напруги +5 В. Такі входи називають «толерантними» до напруги 5 В. В схемі для таких виводів верхній захисний діод підключений не до живлення Vdd, а до обмежувача напруги Vdd_ft. Толерантні виводи позначаються FT (five volt tolerant). Напруга таких входів обмежується до рівня на 4 В вище Vdd. Це зроблено для підключення сигналів з рівнями ТТЛ-логіки (0-5 В). Драйвер входу містить в собі схему управління підтягуючими резисторами та тригер Шмітта. Якщо вивід використовується периферійними пристроями з аналоговим входом (наприклад, АЦП), то такі пристрої підключаються безпосередньо до входу. Периферійні пристрої з цифровим входом і регістр вхідних даних підключаються до входу через тригер Шмітта. Тригер Шмітта на вході

– типове рішення для мікроконтролерів. Він визначає вхідні рівні напруг перемикання, формує на своєму виході стандартні рівні цифрового сигналу (для КМОН-логіки 0 – 3.3 В) з хорошими фронтами, незалежно від якості вхідного сигналу. А також захищає від шуму на вході і дозволяє здійснити узгодження цифрової частини схеми з аналогової без використання додаткових елементів. Тригер Шмітта активний у всіх вхідних режимах, крім режиму аналогового входу. У режимі аналогового входу тригер Шмітта відключається і на його виході формується логічний 0 незалежно від сигналу на вході.

Регістр вхідних даних IDR, підключений до виходу тригера Шмітта, доступний тільки для читання. Він може бути прочитаний незалежно від режиму роботи відповідного виводу і містить значення, яке відповідає реальному сигналу на виводі мікроконтролера. Таким чином, можна зчитувати даніз виводу мікроконтролера навіть тоді, коли він працює в режимі виходу. Якщо вивід налаштований як аналоговий вхід, то у випадку читанні з регістра у відповідному розряді буде записано 0 (тригер Шмітта відключений). Під час використання виводу в якості входу, до нього можуть бути підключені підтягуючі резистори зі значенням опору близько 40 кОм. Підтягування може виконуватись або до живлення Vdd, або до землі Vss. Наявність вбудованих підтягуючих резисторів дозволяє скоротити кількість компонентів на платі і спростити пристрій в цілому. Вони використовуються для того, щоб жорстко задати рівень сигналу на вході, який може в певні моменти роботи пристрою залишатися непідключеним ("висіти у повітрі"). На рис. 2.2 показано стандартну схему підключення кнопки до виводу мікроконтролера: вивід PA0 працює в режимі input pull-up та підключений через кнопку до загального проводу (землі GND). Коли кнопка натиснута, на виводі формується логічний 0, оскільки він замикається на землю. У випадку розімкнутої кнопки підтягуючий резистор pull-up (підтягування до джерела живлення) встановить на вході логічну 1. Таким чином, стан ніжки PA0 завжди буде чітко визначеним – вивід ніколи не буде «висіти в повітрі».

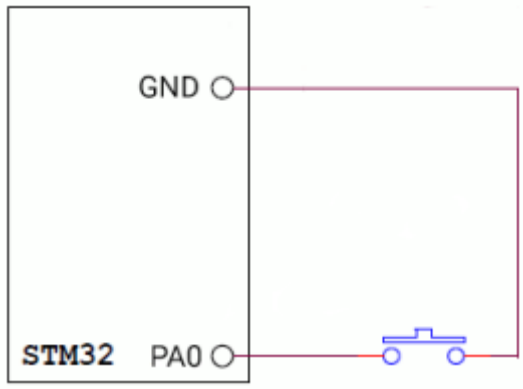

Рис. 2.2. Схема підключення кнопки до виводу мікроконтролера

Варто пам’ятати, що опір внутрішніх підтягуючих резисторів досить великий. З іншого боку, для механічних кнопок може бути обмежений мінімальний струм. При струмі нижче мінімального не гарантовано низький опір замкнутих контактів натиснутої кнопки. Крім того, цей опір може бути нестабільним і стати джерелом підвищеного шуму. В таких випадках рекомендовано використовувати зовнішні підтягуючі резистори для підключення кнопок.

В таблиці 2.1 наведені електричні характеристики виводів портів у режимі входу. Як видно з таблиці, не можна навантажувати вивід струмом понад 5 мА.

Таблиця 2.1. Гранично-допустимі вхідні параметри портів

Параметр	**Значення**
Напруга на толерантних входах	Не менше -0,3 В, не більше Vdd + 4 В У випадку живлення 3,3 В: - 0,3 … 7,3 В

Напруга на звичайних входах	Не менше - 0,3 В, не більше 4 В
Максимальний вхідний струм через захисні діоди	Не більш ніж ±5 мА.

Як правило, реальні сигнали на входах мікроконтролера мають спотворену форму. Неідеальність фронтів дискретних сигналів або незначні коливання амплітуди не є суттєвою проблемою. Значно гіршими є принципові спотворення сигналів, що не дозволяють визначити їх стан простим читанням входу.

Для механічних контактів, таких як кнопки, перемикачі, реле тощо, головною проблемою є брязкіт контактів. Контакти – це механічні елементи. Поверхні у них нерівні, часто покриті брудом, оксидами тощо. Під час комутації контакти відскакують один від одного. Це призводить до хаотичної комутації електричного кола до тих пір, поки не встановиться стабільний контакт. Даний перехідний процес і називається брязкотом контактів.

Під час натискання і відпускання кнопки вона переходить в потрібний стан не одразу. Якийсь час контакти кнопки «бряжчать» між собою, що буде сприйнято мікроконтролером як багаторазові імпульси. Кількість цих імпульсів може перевищувати тисячі. Наочно брязкіт можна побачити на осцилограмі, на якій показано момент відпускання кнопки (рис. 2.3).

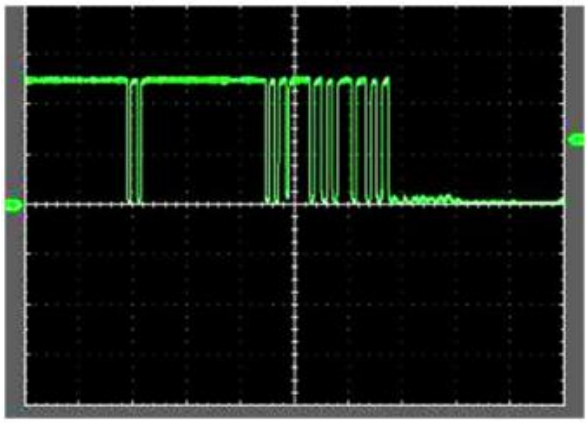

Рис. 2.3. Осцилограма брязкоту контактів

Перехідні процеси протікають дуже швидко і зникають за декілька мілісекунд. Але, обробляючи сигнал від кнопки на високошвидкісному мікроконтролері, існує висока ймовірність зіткнутися з явищем брязкоту, що повинно бути враховано під час програмування. Часова діаграма сигналу комутації механічного контакту показана на рис. 2.4. У такого сигналу неможливо виділити фронт перемикання простим читанням входу. Наприклад, неможливо визначити, скільки натискань кнопки було зроблено.

Рис. 2.4. Діаграма перемикання механічного контакту

У такого сигналу неможливо виділити фронт перемикання простим читанням входу. Наприклад, неможливо визначити, скільки натискань кнопки було зроблено. Тому використовуються різноманітні апаратні та програмні методи усунення брязкоту.

Апаратне усунення брязкоту – це схемотехнічні рішення, які дозволяють усунути даний небажаний ефект. Найчастіше це простий RC-ланцюг, або навіть лише один конденсатор. Принцип роботи такої схеми: конденсатору потрібен певний час для зарядки (або розрядки). Поки конденсатор повністю не зарядиться, на вхід мікроконтролера не надійде необхідний сигнал. Цього часу вистачає на те, щоб перехідний процес встиг завершитися. Таким чином і виконується пригнічення брязкоту. Проста схема апаратного усунення брязкоту контактів приведена на рис. 2.5.

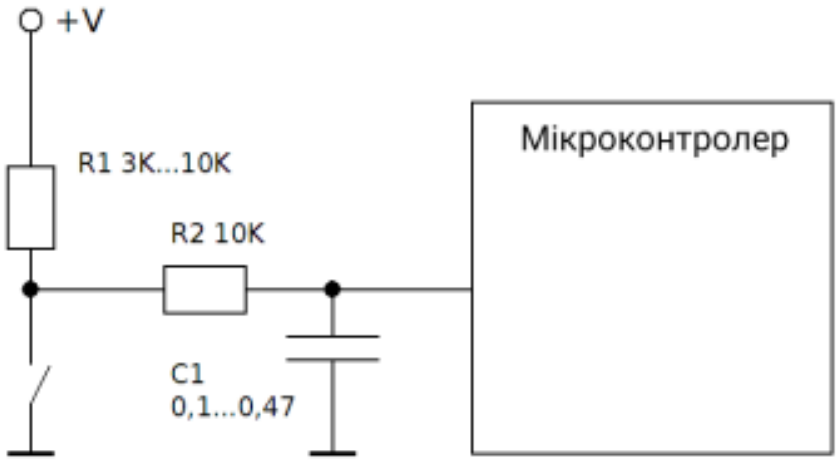

Рис. 2.5. Апаратне усунення брязкоту

У разі використання мікропроцесорної техніки, в переважній більшості випадків немає сенсу ускладнювати схему пристрою і вбудовувати в неї апаратні елементи усунення брязкоту. Використовувати апаратні рішення в пристроях на мікроконтролерах застосовується, наприклад, якщо мікроконтролер малопотужний і навіть найменше витрачання його ресурсів на допоміжні задачі небажане. В інших випадках, значно оптимальніше організувати програмне усунення брязкоту. Найпростіше і найпоширеніше програмне рішення для боротьби з брязкотом – введення тимчасової затримки.

Такий алгоритм передбачає відмову від реагування на занадто часте перемикання сигналу. Якщо кнопка була натиснута протягом занадто короткого часу (наприклад, 1 мс), це є відхиленням від норми. Очевидно, що кнопка не може спрацювати на настільки короткий час, особливо якщо натискається людиною. Відповідно, таке спрацювання можна вважати брязкотом або іншою імпульсною завадою.

Просте програмне вирішення для такого випадку – необхідно дочекатися, поки стан кнопки буде стійким протягом певного проміжку часу, і тільки після цього приймати рішення. Даний спосіб називається очікуванням стійкого стану сигналу. Вважається, що сигнал знаходиться на певному логічному рівні, якщо протягом заданої кількості зчитувань поспіль його стан не змінювався. Тобто, всі результати зчитування сигналу були однаковими.

Наприклад, через кожну 1 мс зчитується стан входу. Час підтвердження становить 10 мс. Якщо 10 зчитувань поспіль відліки були низького рівня, значить стан сигналу вважається рівним логічному 0. Якщо серед 10 читань зустрілася хоча б одна логічна 1, то відлік починається заново.

Такий метод усуває брязкіт контактів та одиночні імпульсні завади. Але у випадку регулярних імпульсних завад метод втрачає ефективність. Приклад такого випадку показано на рис. 2.6.

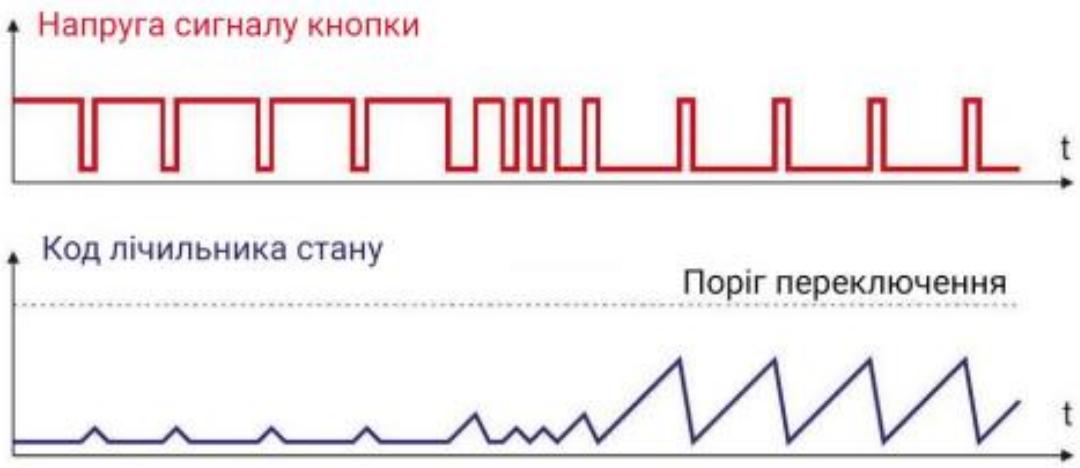

Рис. 2.8. Діаграма усереднення сигналу кнопки

На верхній діаграмі показано сигнал з брязкотом та регулярними імпульсними завадами. На нижній – значення лічильника підтвердження стану. Короткі імпульси перешкоди скидають його, і значення лічильника не досягає порогу перемикання. Якщо повернутись до попереднього прикладу, у випадку наявності 9 зчитувань стану низького рівня і одного читання стану високого рівня за 10 мс, лічильник ніколи не досягне 10. Вхідний рівень в такому випадку ніколи не визначиться, як низький.

Але візуально на рис. 2.6 можна легко визначити, що в середині діаграми стан сигналу змінився – кнопка перемкнулась. На рисунку видно, що в першій половині діаграми середній рівень сигналу високий, а в другій половині – низький. На основі цього можна створити другий алгоритм – усереднення сигналу (рис. 2.7):

- Створюється лічильник середнього значення сигналу;
- У циклі з певним часом зчитується стан сигналу;

Якщо стан дорівнює логічній 1, то до лічильника додається одиниця, якщо логічному 0, то одиниця віднімається. Таким чином, лічильник містить середнє значення сигналу в даний момент часу.

Якщо значення лічильника досягає 0 або менше порогового рівня, приймається рішення, що сигнал низького логічного рівня.

Якщо значення лічильника досягає максимального заданого рівня (часу усереднення), приймається рішення, що сигнал високого логічного рівня.

Рис. 2.7. Алгоритм усереднення сигналу кнопки

Такий алгоритм достовірно виділяє сигнал серед регулярних імпульсних завад, але за умови, що середній рівень завад менше середнього рівня сигналу. Часова діаграма роботи даного методу показана на рис. 2.8.

Обидва алгоритми цифрової фільтрації брязкоту вносять затримку в визначення стану сигналу. Час затримки залежить від часу підтвердження або усереднення (встановленого порогового рівня лічильника). Для механічних контактів зазвичай використовується час 5 - 50 мс.

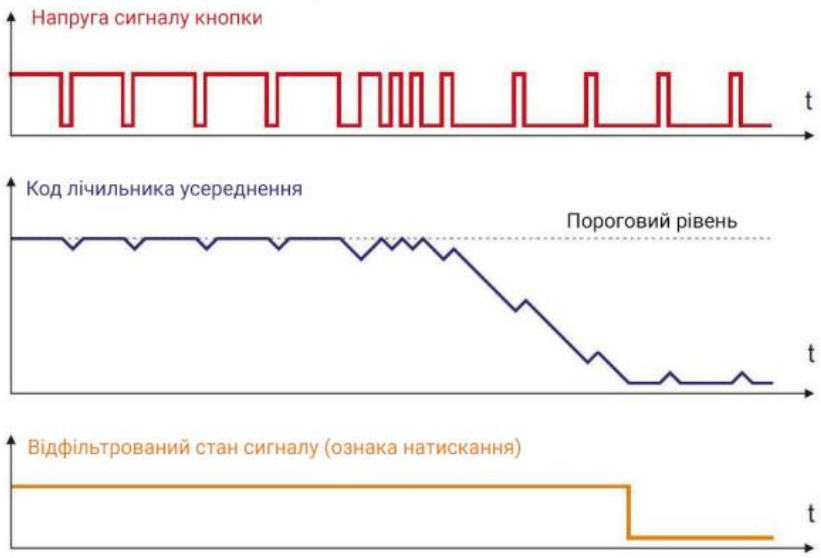

Рис. 2.8. Діаграма усереднення сигналу кнопки

За допомогою часових затримок та на основі модифікації даних алгоритмів можна організувати різні режими натискання кнопок (просте натискання, довге натискання, захист від випадкового натискання тощо). Але варто пам’ятати, що вказана величина затримки є

ефективною лише у разі натискання кнопки людиною або іншим механізмом. Явище брязкоту характерне також і для аналогових компараторів, датчиків з дискретними виходами, зовнішніх дискретних сигналів від інших пристроїв тощо. Для програмної фільтрації такого роду брязкоту можна застосовувати аналогічні підходи, але час затримки буде значно меншим, ніж для випадку механічних контактів.

Хід виконання роботи

1. Виконати попередні налаштування плати «STM32» для керування вбудованим світлодіодом за натисканням кнопки.

1.1. Запустити середовище програмування STM32CubeIDE та створити новий проект.

1.2. Провести початкові налаштування мікроконтролера. Встановити: − Режим відлагодження Debug: «Serial wire»; − Режим тактування від зовнішнього кварцового резонатора High Speed Clock (HSE): «Crystal/Ceramic resonator»; − Вивід PA9 в режимі активного виходу (GPIO mode: «Output Push Pull»), за замовчуванням високий рівень (GPIO output level: «High»), без підтягування (GPIO Pull-up/Pull-down: «No pull-up and no pull-down»), швидкість роботи низька (Maximum output speed: «Low»). − На вкладці Clock Configuration джерело тактування мікроконтролера від зовнішнього кварцового резонатора (HSE), максимальну частоту роботи мікроконтролера – 100 МГц. − На вкладці Project Manager обрати використання бібліотеки HAL та перевірити інші супутні налаштування.

1.3. Активувати вивід PA0 в режимі входу з підтягуванням до низького рівня (Pull-down), як це показано на рис. 2.9. Оскільки в такій конфігурації вивід підтягнуто до землі, низький рівень на вході PA0 буде означати, що кнопка НЕ натиснута. У разі натискання кнопки, вона замкне коло на живлення, і на виводі PA0 з’явиться високий рівень – ознака того, що кнопка натиснута.

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

1.5. Підключити плату STM32 до ПК. Провести компіляцію проекту та завантажити програму до мікроконтролера. Критерієм правильності налаштувань буде вимкнений користувацький світлодіод.

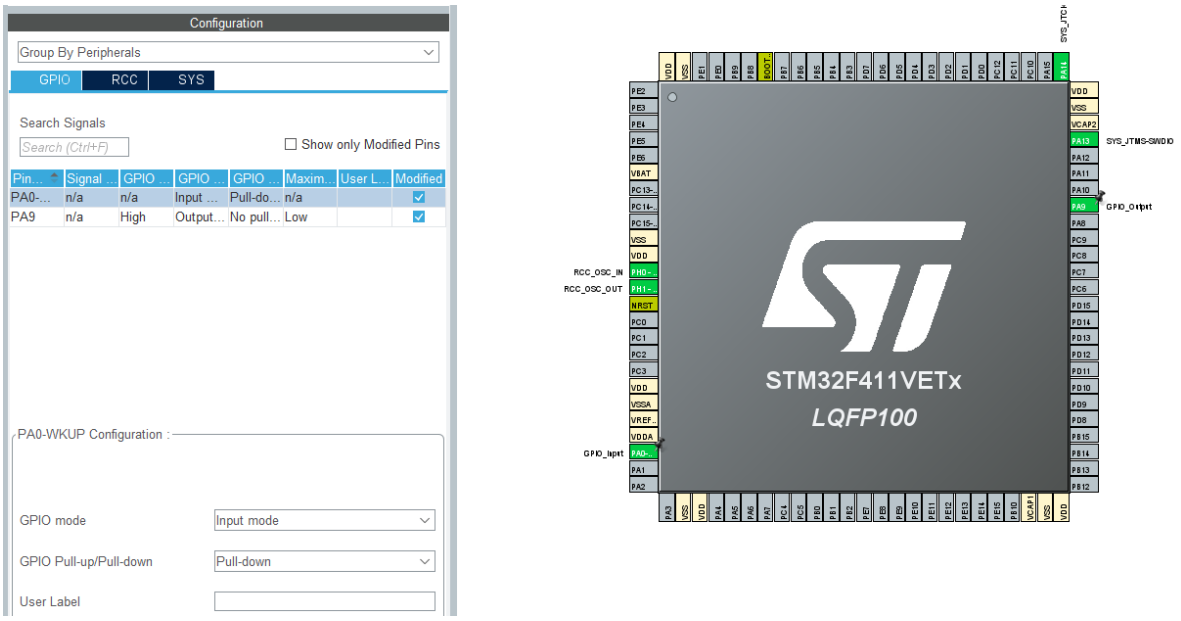

Рис. 2.9. Налаштування виводу PA0 в режимі входу

2. Написати програму для увімкненні світлодіода за натисканням кнопки. Світлодіод має світитися лише тоді, колі кнопка натиснута.

2.1. Для зчитування стану на певному виводі використовується функція HAL_GPIO_ReadPin(). Інформація про функцію:
- Формат: GPIO_PinState HAL_GPIO_ReadPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)
- Аргументи:

GPIOx – вибір порту (GPIOA, GPIOB, GPIOA …);
Pin – номер виводу (GPIO_PIN_0 … GPIO_PIN_15).
- Функція повертає значення:

GPIO_PIN_SET – високий рівень на виводі;
GPIO_PIN_RESET – низький рівень на виводі.

2.2. Написати програмний код, в якому за допомогою HAL_GPIO_ReadPin() зчитується стан на виводі PA0. Якщо на PA0 низький рівень, на виводі PA9 встановлюється логічна 1 (світлодіод не світиться). Якщо навпаки – на PA9 встановлюється логічний 0 (світлодіод світиться).

```
while (1)
  {
   /* USER CODE END WHILE */
   /* USER CODE BEGIN 3 */
  if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0))
  {
   HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_SET);
  }
  else
  {
   HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_RESET);
  }
  }
```

2.3. Завантажити програму до мікроконтролера. Критерієм правильності буде запалення світлодіода під час утримання натиснутої кнопки.

3. Дослідити явище брязкоту механічних контактів.

3.1. Створити нові змінні, в яких буде зберігатись поточний стан кнопки та лічильник кількості натискань на кнопку. Для цього в секції Private variables прописати наступний код:

```
/* Private variables ---------------------------------------------*/
/* USER CODE BEGIN PV */
uint8_t btn_state = GPIO_PIN_SET; //поточний стан кнопки
uint8_t pressed = 0; //лічильник кількості натискань
/* USER CODE END PV */
```

3.2. Модифікувати код програми наступним чином:

```
/* USER CODE BEGIN WHILE */
  while (1)
  {
   /* USER CODE END WHILE */
   /* USER CODE BEGIN 3 */
   if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0))
   {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_SET);
   }

   else
   {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_RESET);
```

```
    }
    // Якщо стан кнопки змінився, збільшуємо кількість натискань на 1
    // та фіксуємо новий стан
    if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0) != btn_state)
    {
      pressed += 1;
      btn_state = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0);
    }
  }
  /* USER CODE END 3 */
```

Запустити режим відлагодження  Debug . У правій частині головної робочої області обрати вкладку Live Expressions (змінні в реальному часі) та додати до списку відслідковування створені змінні, натиснувши на напис Add new expression (рис.2.10).

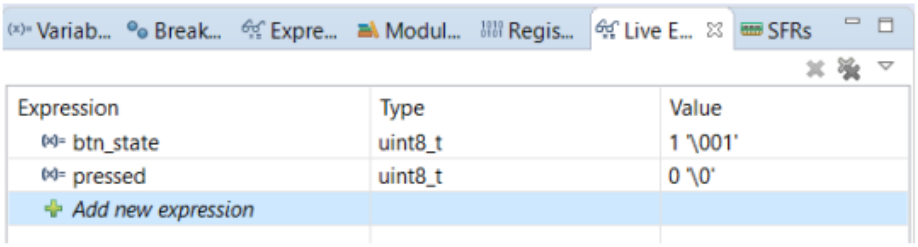

Рис. 2.10. Відстеження змінних

3.4 Запустити мікроконтролер в режимі відлагодження, натиснувши на піктограму (Resume) або клавішу F8. Почергово натискати та відпускати кнопку та спостерігати за значеннями змінних. Якщо значення лічильника pressed буде відрізнятись від реальної кількості натискань/відпускань, то у кнопки присутнє явище брязкоту.

3.5. Зупинити режим відлагодження.

3.6. Модифікувати програму з метою реалізації алгоритму усереднення сигналу для фільтрації брязкоту. Для цього створити змінну count, в якій буде зберігатись значення лічильника усереднення, а також змінну flag, яка буде приймати значення «1» у випадку досягнення лічильником порогу натискання. Код програми для реалізації фільтрації брязкоту:

```
/* USER CODE BEGIN PV */
  uint8_t btn_state = GPIO_PIN_SET; //поточний стан кнопки
  uint8_t pressed = 0; //лічильник кількості натискань
  uint16_t count = 0;    //лічильник для усереднення
  uint8_t flag = 0;    //ознака натиснутої/відпущеної кнопки
  /* USER CODE END PV */
/* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */
          count = 1;
          while(count)
          {
                  if(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0)) count++;

                  else count--;

          if(count == 500)
```

```
        {
            flag = 1;
            count = 0;
        }
        else if (count == 0) flag = 0;
        }

        if (flag==1) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_SET);

        else HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_RESET);

        if (flag == 1 && btn_state == 0) pressed++;
        btn_state = flag;

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

3.7. Запустити мікроконтролер в режимі відлагодження та протестувати роботу алгоритму, додавши створені змінні до блоку відстеження Live Expressions. Зробити висновки.

4. На основі реалізованого алгоритму створити функцію для фільтрації брязкоту. Переписати код програми з використанням створеної функції.

4.1. Для створення функцій користувача у файлі main.c, необхідно розмістити оголошення функції всередині блоку /* USER CODE BEGIN 4 */ … /* USER CODE END 4 */ . Код функції button_pressed() для реалізації фільтрації брязкоту можна записати наступним чином:

```
/* USER CODE BEGIN 4 */
 int button_pressed (void)
 {
 uint16_t count = 0;    //лічильник для усереднення
 uint8_t flag = 0;    //ознака натиснутої/відпущеної кнопки
 count = 1;
 while(count)
 {
 if(!HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0))
 {
    count++;
 }
 else
 {
      count--;
 }
 if(count == 50)
 {
      flag = 1; //Or action
      count = 0;
 }
    else if (count == 0) flag = 0;
    }
  return flag;
 }
 /* USER CODE END 4 */
```

4.1. Прототипи створених функції розміщуються в секції Private function prototypes у блоці /* USER CODE BEGIN PFP */… /* USER CODE END PFP*/. Код прототипу функції button_pressed():

```
/* USER CODE BEGIN PFP */
int button_pressed (void);
/* USER CODE END PFP */
```

4.2. Видалити оголошення змінних count та flag із секції Private variables, оскільки тепер вони є локальними змінними функції button_pressed().

4.3. Переписати основний цикл програми із урахуванням всіх змін:

```
  /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1)
    {
      /* USER CODE END WHILE */
      /* USER CODE BEGIN 3 */
        if (button_pressed())
        {
          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_RESET);
        }
        else
        {
          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_SET);
        }
    }
    /* USER CODE END 3 */
```

4.4. Запустити програму та перевірити правильність її роботи.

5. Модифікувати функцію button_pressed(), додавши дві різні дії – реакції на коротке та довге натискання кнопки. У випадку короткого натискання світлодіод має змінити стан на протилежний. У випадку довгого натискання світлодіод повинен блимнути три рази з періодом 500 мс.

5.1. Для реалізації описаного алгоритму, змінна flag повинна приймати не лише логічні значення, а і додаткове. Наприклад, 0 (кнопка не натиснута), 1 (коротке натискання кнопки) та 2 (довге натискання кнопки).

5.2. Для переключення логічного стану на вихідному виводі мікроконтролера на протилежний (операція інверсії) використовується функція HAL_GPIO_TogglePin(). Інформація про функцію:

Формат: void HAL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)

Аргументи:

- GPIOx – вибір порту (GPIOA, GPIOB, GPIOA …);
- Pin – номер виводу (GPIO_PIN_0 … GPIO_PIN_15).

5.3. Переписати функцію button_pressed() з урахуванням описаного алгоритму:

```
/* USER CODE BEGIN 4 */
 int button_pressed (void)
 {
 uint32_t count = 0;
 uint8_t flag = 0;
```

```
  count = 1;
  while(count)
  {
     if(!HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0))
     {
      count++;
     }
     else
     {
      count--;
     }
     switch (count)
     {
      case 100:
        flag = 1; //Коротке натискання
        break;
      case 300000:
        flag = 2; //Довге натискання
        count = 0;
        break;
     }
  }
     return flag;
  }
  /* USER CODE END 4 */
```

5.4. Створити нову змінну action, яка буде містити ознаку дії у випадку натискання кнопки:

```
/* USER CODE BEGIN PV */

  uint8_t action = 0;
  /* USER CODE END PV */
```

5.5. Модифікувати код основної програми наступним чином:

```
/* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1)
    {
      /* USER CODE END WHILE */
      /* USER CODE BEGIN 3 */
        action = button_pressed();
      if (action == 1)
      {
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_9);
        HAL_Delay(100);
      }
      else if (action == 2)
      {
        for (int i=5; i>=0; i--)
        {
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_9);
        HAL_Delay(500);
        }
      }
    }
    /* USER CODE END 3 */
```

5.6. Завантажити програму до мікроконтролера та перевірити правильність виконання алгоритму. (При короткому натиснені змінюється стан світлодіода, а при довгому натисканні він починає блимати)

Індивідуальні завдання

1. Модифікувати програму таким чином, щоб користувацький світлодіод миготів з інтервалом 1 с лише у випадку, коли кнопка натиснута і утримується. У разі відпускання кнопки світлодіод має швидко блимнути 5 разів та вимкнутись.

2. Написати програму, в якій у випадку натискання кнопки користувацький світлодіод розпочинає миготіти з частотою 2 Гц. При утримані кнопки в натиснутому стані кожних 2 секунди частота блимання повинна збільшуватись в два рази, максимум до 64 Гц. Після чого світлодіод повинен загаснути.

Зміст звіту

1. Тема та мета роботи;
2. Виконання вказівок з ходу виконання роботи;
3. Скріншот файлу .ioc з конфігурацією усіх задіяних пінів контролера(налаштування по кожному піну) для кожного підпункту індивідуального завдання окремо;
4. Фрагмент коду з початку та до SystemClock_Config для кожного підпункту індивідуального завдання окремо;

Контрольні запитання

1. Опишіть структуру та принципи роботи вхідного драйверу виводу порту GPIO.
2. Поясніть призначення тригера Шмітта.
3. Що таке підтягуючі резистори? Наведіть приклади використання.
4. Наведіть схему підключення кнопки до виводів мікроконтролера. Опишіть два варіанти підключення.
5. Поясніть суть та причини виникнення явища брязкоту механічних контактів.
6. Опишіть метод апаратного усунення брязкоту. Як розрахувати номінали елементів, що використовуються в схемі апаратного усунення брязкоту?
7. Поясніть два програмних алгоритми усунення брязкоту. В чому їх переваги та недоліки?
8. Опишіть послідовність початкових налаштувань мікроконтролера для підключення кнопки.
9. Опишіть призначення, аргументи та вихідні дані функцій бібліотеки HAL, які використовуються для управління виводами портів GPIO в режимі входів.