НАВЧАЛЬНО-МЕТОДИЧНА ЛІТЕРАТУРА

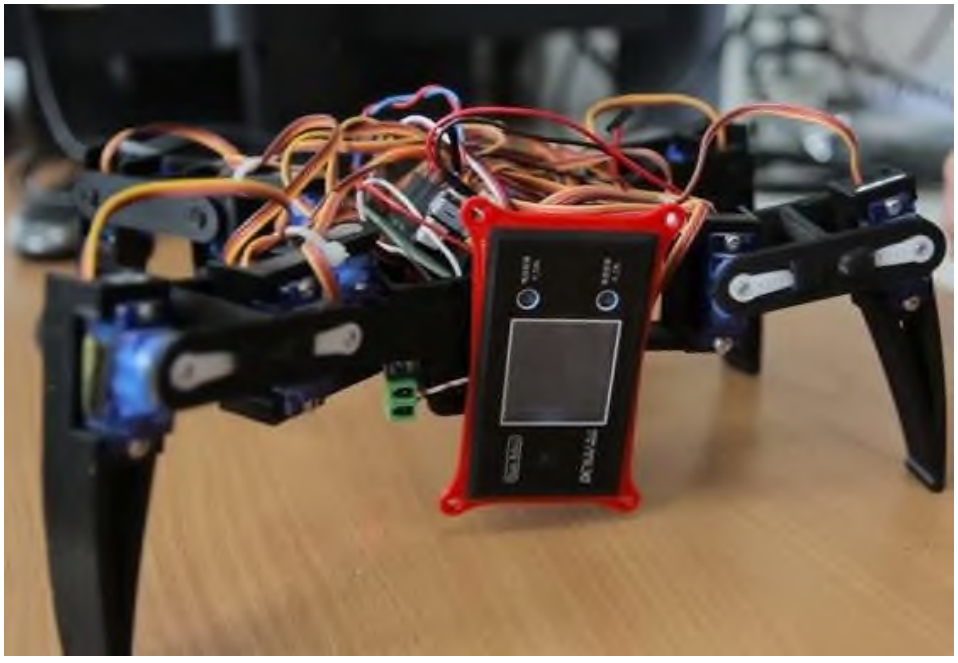
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ
УНІВЕРСИТЕТ ІМЕНІ ІВАНА ПУЛЮЯ

Кафедра приладів і контрольно-вимірювальних системи

М Е Т О Д И Ч Н І В К А З І В К И
для виконання практичних робіт
з дисципліни

Компоненти мікро- та нанотехніки

для студентів спеціальності
176 «Мікро- та наносистемна техніка»



ТЕРНОПІЛЬ, 2023

Паламар М.І., Стрембіцький М.О. Методичні вказівки для виконання практичниї робіт з дисципліни «Компоненти мікро- та нанотехніки» для студентів спеціальності 176 – Мікро-та наносистемна техніка. Тернопіль: ТНТУ, 2023. 54 с.

Укладачі: д.т.н., Паламар М.І., к.т.н., Стрембіцький М.О.

Відповідальний за випуск: завідувач кафедри приладів і контрольно-вимірювальних систем Паламар М.І.

Розглянуто та затверджено на засіданні приладів і контрольно-вимірювальних систем Тернопільського національного технічного університету імені Івана Пулюя, протокол № 7 від «1» травня 2023 р.

Схвалено та рекомендовано до друку науково-методичною комісією факультету прикладних інформаційних технологій та електроінженерії ТНТУ, протокол № 10 від «5» травня 2023 р.

Методичні вказівки призначені для полегшення засвоєння дисципліни «Компоненти мікро- та нанотехніки» і контролю знань студентів. Складається з урахуванням модульної системи навчання, рекомендацій до самостійної роботи і індивідуальних завдань, тестів, екзаменаційних питань, типової форми та вимог для комплексної перевірки знань з дисципліни.

ЗМІСТ

ПРАКТИЧНА РОБОТА № 1 32-Х РОЗРЯДНА АРХІТЕКТУРА ARM

Мета роботи - вивчення структури процесора ARM.

Загальні відомості

Процесори ARM є ключовим компонентом для великої кількості успішних 32-бітних вбудованих систем. Процесори ARM широко використовуються в мобільних телефонах, планшетах та інших портативних пристроях. ARM засновані на RISC-архітектурі, що дозволяє зменшити споживання енергії процесором і, таким чином, робить їх ідеальним вибором для вбудованих систем.

Хоча ARM засновані на RISC-архітектурі, вони не повністю повторюють принципи побудови таких систем. Для того, щоб зробити ARM більш пристосованими до використання у вбудованих системах, довелося піти на такі відхилення від принципів RISC:

Змінна кількість циклів виконання для простих інструкцій. Прості інструкції ARM можуть виконуватися декілька циклів. Наприклад, виконання інструкцій Load і Save залежить від кількості регістрів, які їм передані.

Можливість поєднувати команди зсуву і обертання з командами обробки інформації.

Умовне виконання - інструкція виконується лише в тому випадку, якщо виконується конкретна умова. Це збільшує продуктивність і дозволяє позбутися від операторів розгалуження.

Поліпшені інструкції - процесори ARM підтримують поліпшені DSP-інструкції для операцій з цифровими сигналами.

Процесори ARM містять до 18 регістрів: 16 регістрів даних і 2 регістра процесів. Усі регістри містять 32 біта і іменуються від R0 до R15. Регістри R13, R14, R15 використовуються для виконання певних специфічних завдань:

R13 використовується в якості покажчика стека;

R14 використовується як зв'язуючий регістр;

R15 грає роль лічильника.

Залежно від контексту ці регістри можуть використовуватися як регістри загального призначення. Також є два програмних регістра, які називаються CPSR (Current Program Status Register) і SPSR (Saved Program Status Register), які використовуються для збереження стану процесора і програми.

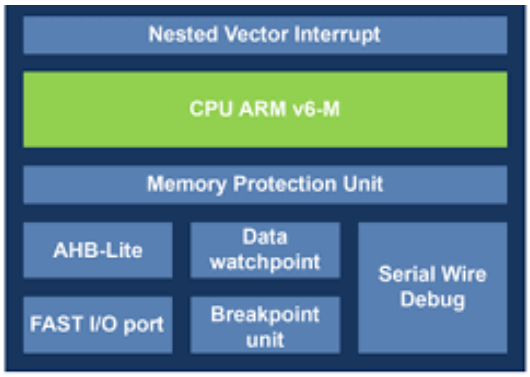


Рисунок 1.1 - Складові модулі процесорів ARM Cortex-M4

Одними з останніх процесорів для вбудованих систем, є процесори, засновані на архітектурі ARM Cortex-M4. Ці процесори призначені для використання в цифровій обробці сигналів (Digital Signal Processing, DSP). У загальному вигляді мікроконтролери, засновані на базі ARM Cortex-M4 мають такі внутрішні модулі (рис. 1.1): Мікроконтролер, встановлений на розглянутій платі, STM32F407VG, в якості основи використовує саме рішення ARM Cortex- M4.

Особливості будови STM32 процесорів

Мікроконтролери STM32 побудовано з використанням 32-розрядний ядра Cortex різних версій (в мікроконтролері, встановленому на платі stm32f4Discovery використовується ядро Cortex-M4). Основні характеристики ядра мікроконтролерів STM32 представлені в таблиці 1.1.

Таблиця 1.1 - Основні характеристики ядра мікроконтролерів STM32

Характеристика	Пояснення
Ширина слів для даних, розрядів	32
Архітектура	Гарвард
Конвеєр виконання інструкцій	3-х ступеневий
Набір інструкцій	RISC
Розрядність пам'яті програм	32
Буфер попередньої вибірки, розрядів	2x64
Середній розмір інструкцій, байтів	2
Тип переривань	Векторизовані
Затримка реакції на переривання	12 циклів
Режими керування енергозабезпеченням Сон	сон по виходу, глибокий сон
Інтерфейс програмування та налагодження	ST-LINK, JTAG

Мікроконтролери STM32Fxxx типу побудовані на гарвардської архітектурі і мають 3-х ступеневий конвеєр, який зменшує час виконання команд. Вони розроблені для побудови систем з максимальною енергоефективністю і мають кілька режимів управління енергоспоживанням. Також використовуються внутрішні інтерфейси пам'яті шириною більше, ніж середня довжина інструкції. Це мінімізує число звернень до шини пам'яті, тому споживання електроенергії, пов'язане з операціями по шині і читанням незалежній пам'яті є мінімальним. Використана технологія обробки переривань з виключенням внутрішніх операцій над стеком (tail chaining), скорочує час реакції на переривання і зменшує кількість необхідних операцій зі стеком.

На рис. 1.2 представлено спрощене уявлення цифрового периферійного пристрою.

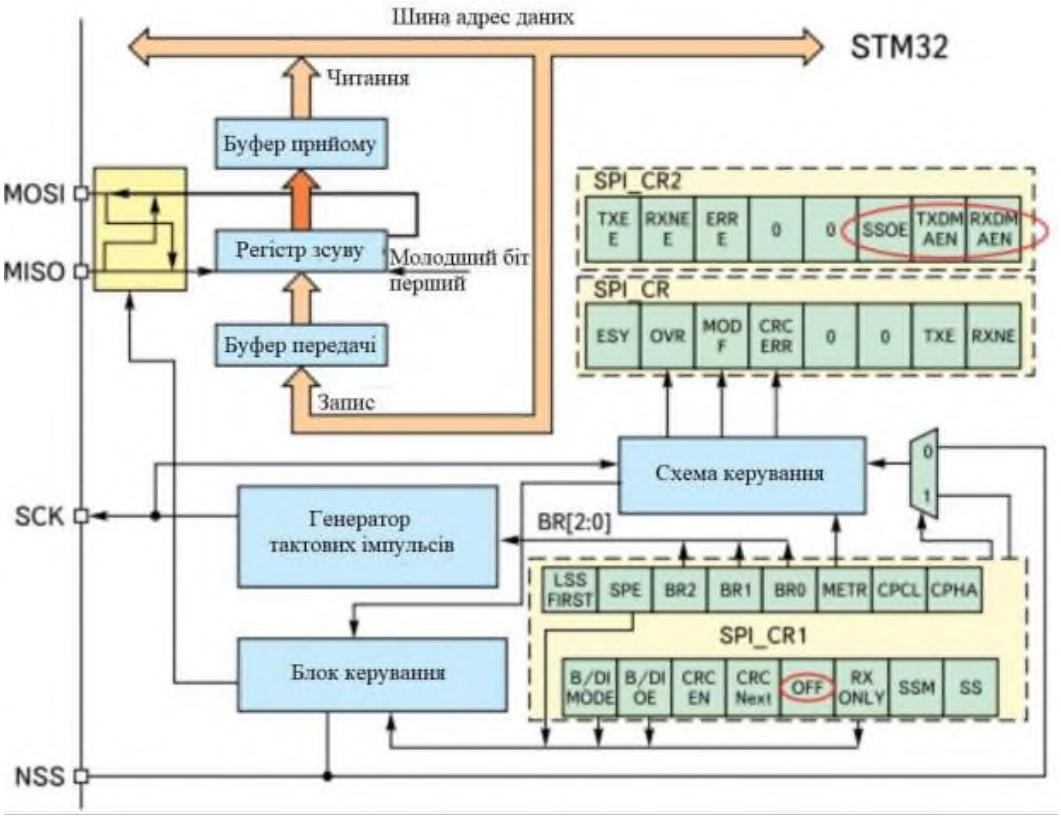


Рисунок 1.2 - Структура периферійного пристрою

Периферійний вузол може бути розділений на два основні блоки. Перший блок - це ядро, яке має архітектуру на скінчених автоматах, лічильники і будь- який вид комбінаторної або послідовної логіки. Вони призначені для виконання завдань, які потребують участі процесора, таких як прості завдання передачі даних, управління аналоговими входами або виконання функцій, прив'язаних до синхросигналів. Ядро периферійного вузла зв'язується із зовнішнім світом через порти введення/виводу МК. Зовнішні з'єднання можуть складатися з декількох сигналів або складних шин. Другий блок - налаштування і управління периферією, які здійснюються програмою через регістри, з'єднані з внутрішньою шиною, яка розділяється з іншими ресурсами МК.

Короткий опис STM32F4 Discovery

Плата STM32F4 Discovery (рис. 1.3) призначена для ознайомлення з можливостями 32-х розрядного МК на основі ARM-архітектури, а також для реалізації власних пристроїв і програм з використанням апаратного забезпечення платою.

Плата STM32F4Discovery забезпечена наступними пристроями:

- Мікроконтролером STM32F407VGT6 з ядром Cortex-M4F тактовою частотою 168 МГц, 1 Мб;
- Flash-пам'яті, 192 кб RAM в корпусі LQFP100;
- налагоджувачем ST-Link / V2 для налагодження та програмування МК;
- живленням плати через USB або від зовнішнього джерела живлення 5В;
- давачем руху ST MEMS LIS302DL і виходами цифрового акселерометра по трьох осях;

- давач звуку ST MEMS MP45DT02;
- звуковим ЦАП CS43L22;
- вісьмома світлодіодами: LD1 (червоний / зелений) для USB-підключення, LD2 (червоний) для живлення 3.3 В, чотири призначені для користувача світлодіода: LD3 (рожевий), LD4 (зелений), LD5 (червоний), LD6 (синій) і два світлодіода для USB LD7 (зелений) і LD8 (червоний);
- двома кнопками (для програмування користувачем і для перезапуску).

Рисунок 1.3 - Зовнішній вигляд плати STM32F4Discovery

Порядок виконання роботи

1. Озайомитися із структурною схемою плати STM32F4Discovery.
2. Створити проект в середовищі програмування ARM.
3. Візкомпілювати проєкт.
4. Представити результати відкомпільованого коду програми.

Зміст звіту

1. Мета роботи.
2. Відомості про порядок створення проекту..
3. Організація інтерфейсу середовища для програмування.
4. Висновки.

Контрольні питання

1. Назвіть характерні особливості RISC–процесорів.
2. Які архітектури процесорів є RISC–подібними?
3. Як розуміти слова «скорочений набір команд» у RISC–процесорів?

4. Дайте визначення архітектурі ARM.

5. Де сьогодні широко використовуються ARM–контролери?

6. Коли відбувся перехід з класичної архітектури фон Неймана на модифіковану Гарвардську архітектуру ARM–контролерів?

7. Чим відрізняються Гарвардська архітектура та архітектури фон Неймана?

8. Які класи процесорів входять до сімейства Cortex ?

9. Охарактеризуйте процесор Cortex–A.

10. Охарактеризуйте процесор Cortex–R.

11. Охарактеризуйте процесор Cortex–M.

12. Чим відрізняються версії ARM–ядер від версій ARM– архітектури?

13. Назвіть відмінності між ARM7 та ARM7TDMI?

ПРАКТИЧНА РОБОТА № 2 РОБОТА З STM32

Мета роботи - вивчення структури процесора ARM.

Загальні відомості

Плата розробника під'єднується до ПК за допомогою кабелю USB з роз'ємами USB типу A та mini-USB типу B. У комплекті такий шнур не надається, тому його потрібно придбати окремо. Зовнішній вигляд конекторів кабелю показано на рис. 2.1.

Рисунок 2.1 - Конектори кабелю для приєднання STM32F4Discovery до ПК

Передбачається, що на комп'ютері встановлена операційна система сімейства Windows (XP, або новіші).

Для початку роботи слід встановити середовище розробки. Далі розглянуто процес установки і початку роботи для одного з найпопулярніших засобів розробки для ARM - Keil. Розглянемо деталі установки та використання середовища розробки Keil uVision.

Завантажити програму можна з сайту виробника - https://www.keil.com/download/product/, вибравши для завантаження пункт MDK-ARM останньої версії, після чого потрапляємо на сторінку, на якій потрібно ввести свої персональні дані (рис 2.2).

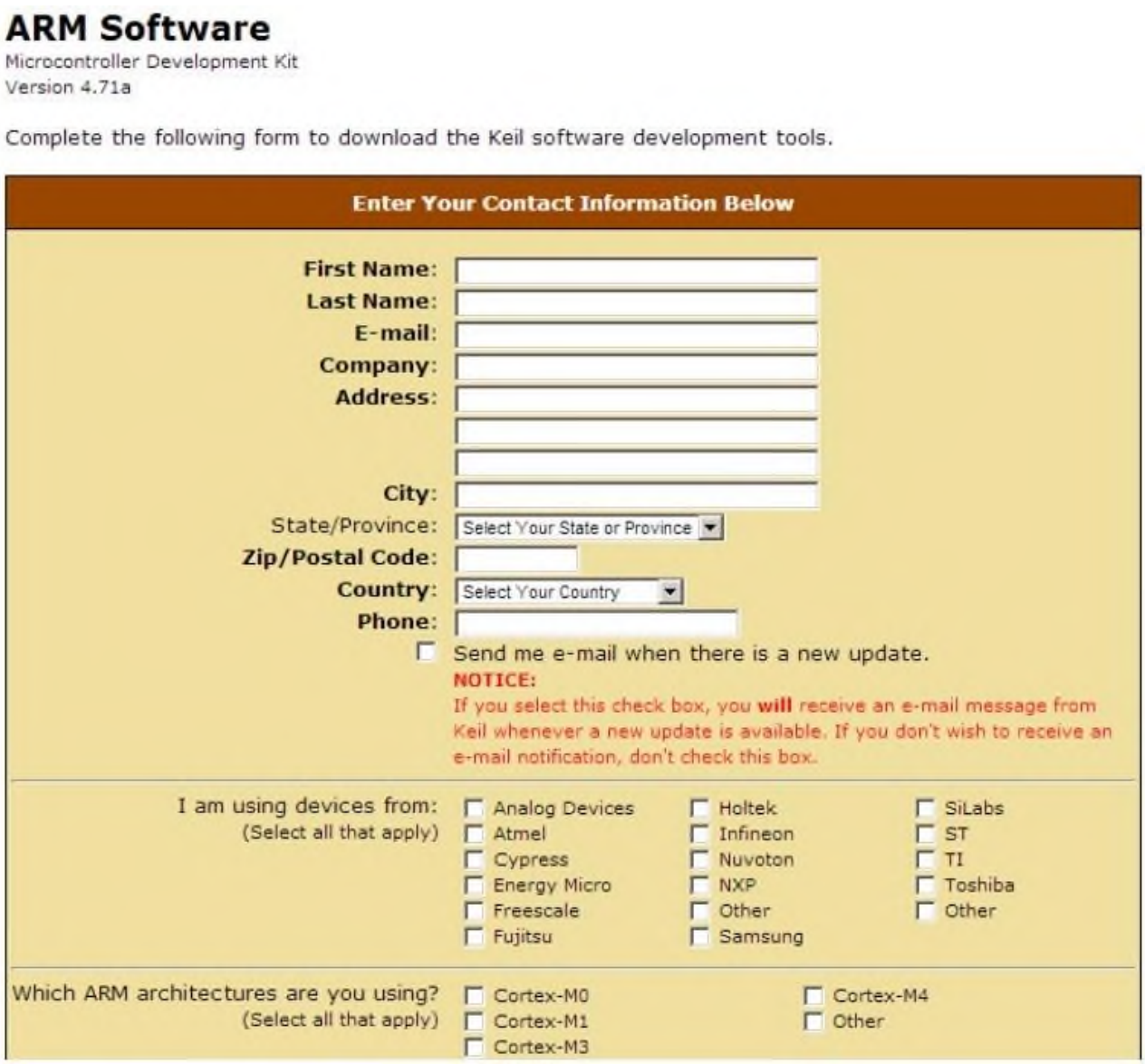


Рисунок 2.2 - Реєстраційна форма для завантаження Keil uVision

Після заповнення форми надається можливість завантажити програмний продукт. Інсталяція потребує досить багато вільного дискового простору (близько 2.5 Гб), тому слід заздалегідь підготувати необхідні ресурси. Після запуску установки відкриється наступне вікно (рис. 2.3).

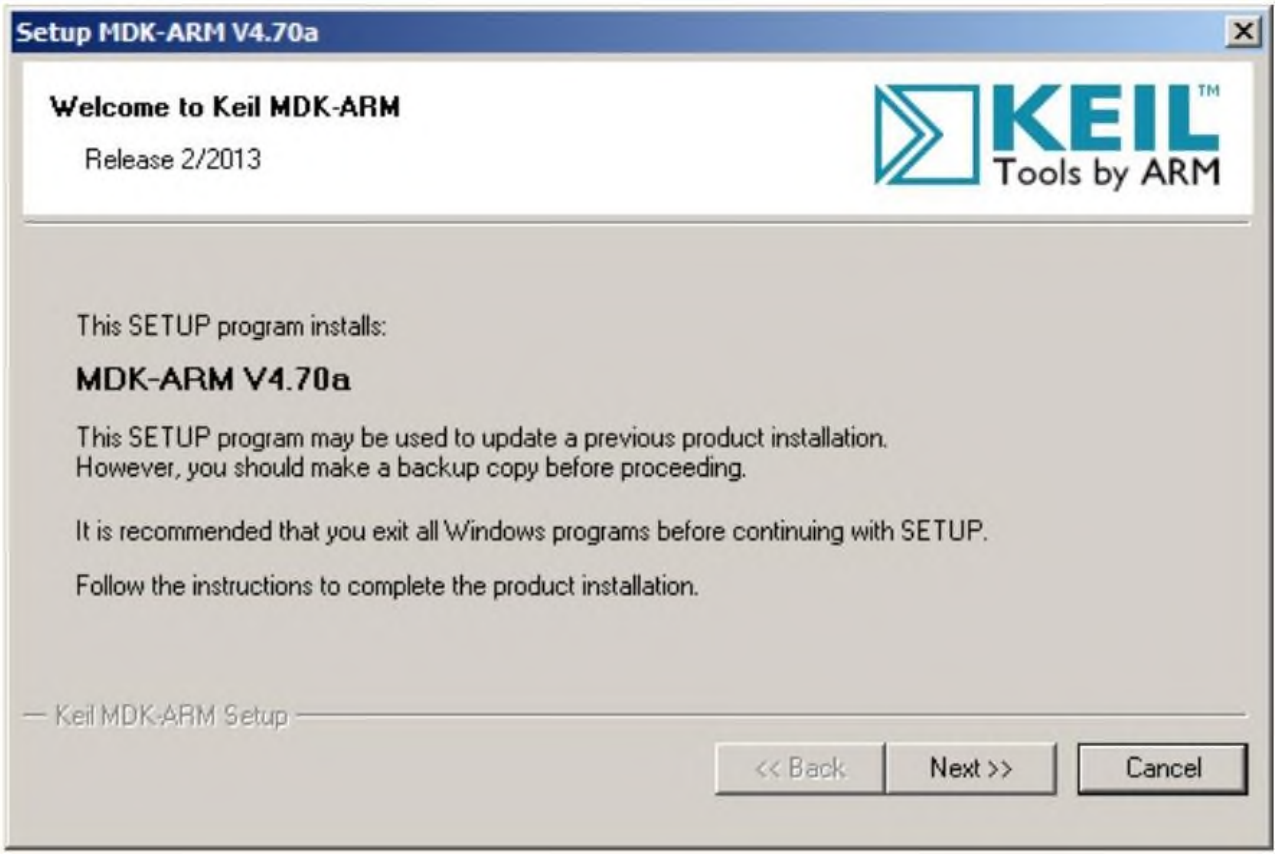


Рисунок 2.3 - Початок інсталювання Keil

Далі необхідно прийняти умови ліцензійної угоди, а також вибрати директорію для інсталяції (рис. 2.4), потім потрібно вказати інформацію про користувача.

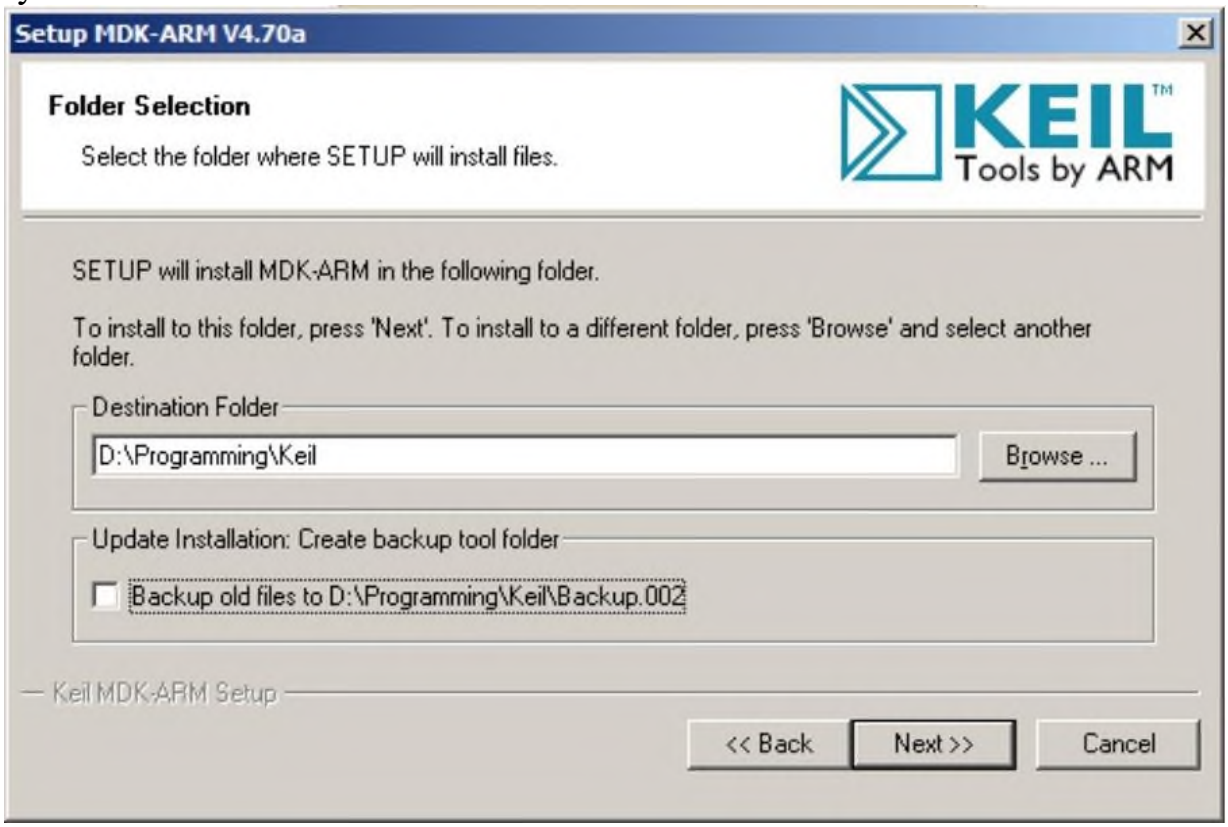


Рисунок 2.4 - Вибір директорії для інсталювання

Після цього почнеться безпосередньо процес інсталювання.

Крім самого середовища розробки для написання програм необхідні також бібліотеки Cortex Microcontroller Software Interface Standart (CMSIS) і Standart Peripheral Library (SPL). Краще за все їх завантажити з сайту виробника st.com.

Бібліотека SPL служить для управління всіма основними пристроями, що входять до складу мікроконтролера: USART, SPI, DMA, ADC, DAC, ...

Основною перевагою бібліотеки є те, що вона робить більш зрозумілим управління мікроконтролером для розробника, в тому числі і початківця, який ще не знає всіх тонкощів налаштувань напряму через регістри. Крім того, разом із самою бібліотекою поширюються і демонстраційні проекти для роботи з основними пристроями, на які можна спиратися при написанні власних проектів.

Після відкриття вікно середовища виглядає наступним чином (рис. 2.5):

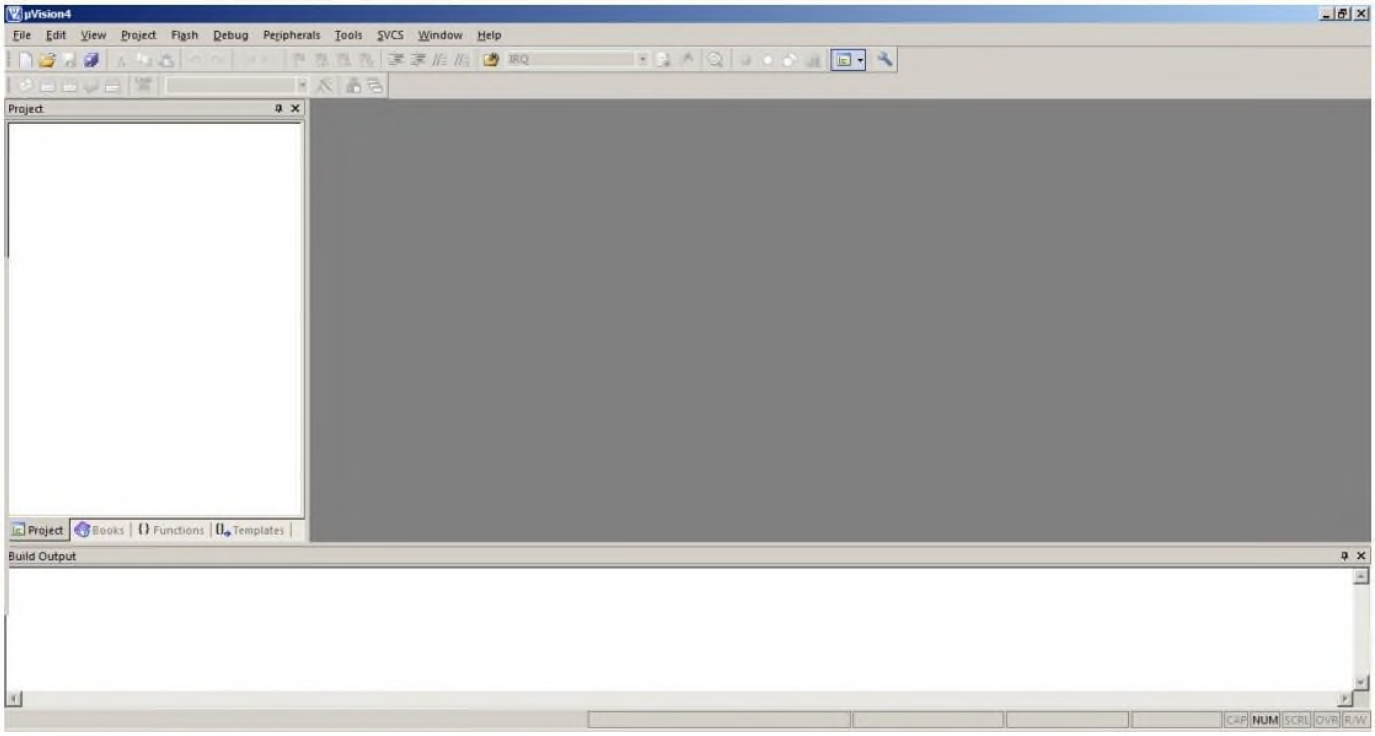


Рисунок 2.5 - Зовнішній вигляд середовища розробки

Тепер створимо проект для stm32F4Discovery. Для цього виконуємо команду меню Project \ New uVision Project. В результаті з'явиться діалогове вікно для вибору теки розташування проекту. Для кожного проекту бажано створювати окрему теку, оскільки проект може розростатися і містити велику кількість файлів, тому тримати 2 проекти в одній теці буде незручно. Після вибору теки слід вибрати мікроконтролер, для якого планується написання програми. Вибираємо виробника (STMicroelectronics) і конкретну модель контролера STM32F407VG (рис. 2.6).

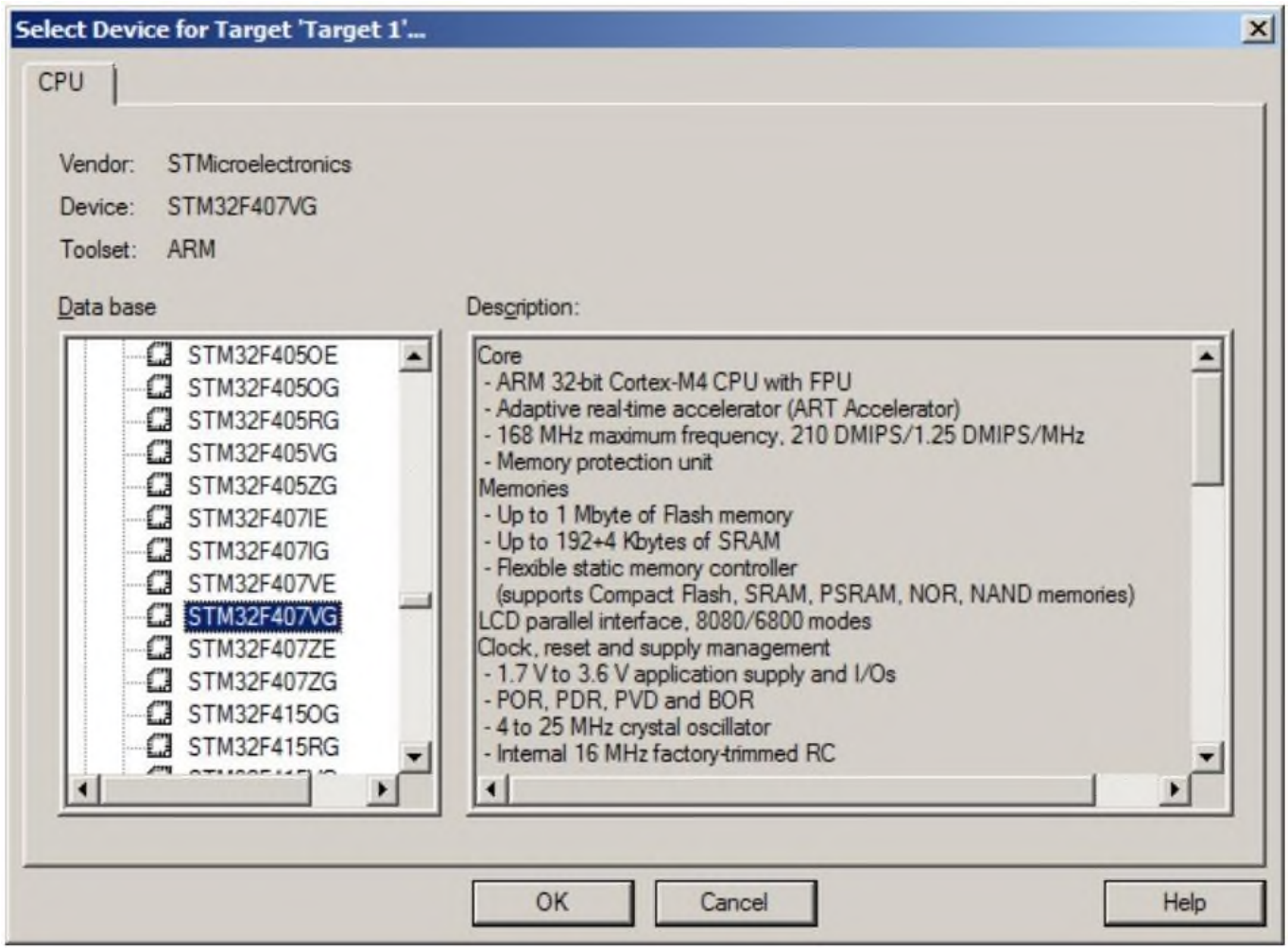


Рисунок 2.6 - Вибір мікроконтролеру для якого писатиметься програма

Далі відкриється діалогове вікно з пропозицією додати до проекту файл запуску startup_stm32f4xx.s, яке потрібно підтвердити. Середовище розробки створить проект, а також створить в ньому групу для вихідних файлів, в якій і буде розташований згаданий вище файл. Групи файлів служать для зручної організації представлення файлів в проекті. В даному прикладі використано 4 групи файлів: Startup, User, CMSIS і SPL.

Для початку перейменуємо створену за замовчуванням групу і назвемо її Startup та додамо ще 3 групи з допомогою контекстного меню проекту в панелі Project, виконавши команду Add Group.

Тепер додамо необхідні нам файли в проект. Перед додаванням файлів бажано скопіювати їх в директорію проекту. З бібліотеки CMSIS нам знадобляться наступні файли:

- stm32f4xx.h;
- system_stm32f4xx.h;
- system_stm32f4xx.c;
- stm32f4xx_conf.h;
- core_cm4.h.

Всі перераховані вище файли знаходяться в директорії, де розпакований архів з бібліотекою. До групи SPL Вам потрібно додати файли з бібліотеки SPL для роботи з пристроями. Необхідно додавати пару *.c (*.cpp) і файлів заголовків *.h (*.hpp).

Наприклад, для роботи з портами введення / виводу слід додати файли stm32f4xx_gpio.c та stm32f4xx_gpio.h.

Обов'язково слід підключити файли для управління пристроєм скидання і тактування, а саме, stm32f4xx_rcc.c та stm32f4xx_rcc.h. Вони містять функції для вмикання тактування периферійних пристроїв.

Після цього залишається додати ще один файл, в якому міститься функція main. Спочатку створимо і збережемо його за допомогою команд меню File, а потім додамо його в групу Users. В результаті отримаємо проект з наступною структурою (рис. 2.7):

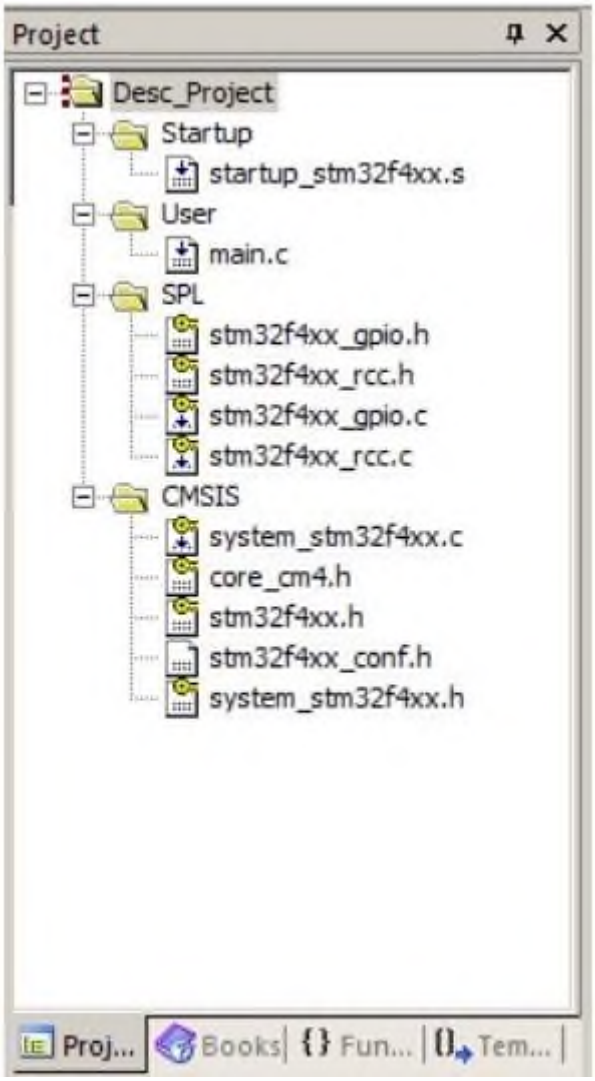


Рисунок 2.7 - Структура типового проекту для мікроконтролеру STM32

Залишилося зробити ще деякі налаштування, які потрібні для компіляції коду та виконання налагодження. Налаштування виробляються у вікні налаштувань, яке відкривається через команду Project / Options for target ... або кнопкою на панелі інструментів, або, використовуючи комбінацію клавішею Alt-F7. У вікні, на вкладці C/C++ задамо визначення USE_STDPERIPH_DRIVER, STM32F4XX в текстовому полі Define. Тут ж можна задати шляху до теки з заголовками файлами.

Налаштування налагодження за замовчуванням дозволяють проводити налагодження в режимі симуляції, проте, якщо пристрій підключено до комп'ютера, то краще проводити налагодження на самому пристрої. Для цього необхідно на

вкладці Debug для пункту Use вказати ST-Link Debugger. Після цього слід натиснути кнопку Settings. На вкладці Debug значення Port встановити як SW. Потім перейти до вкладки Trace Download та додати в список Programming Algorithm пристрій STM32Ff4xx як показано на рис. 2.8.

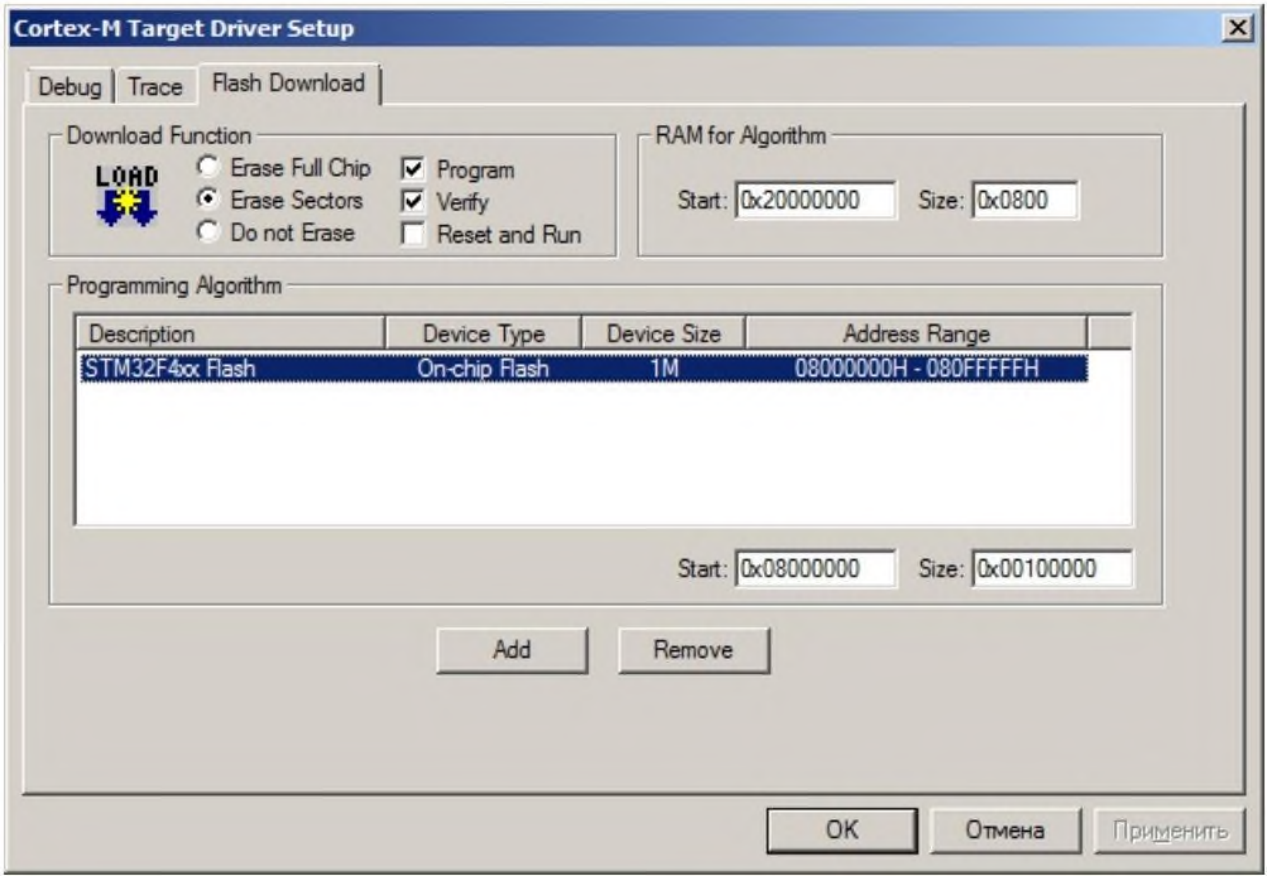


Рисунок 2.8 - Налаштування для налагодження програми на фізичному пристрої

На вкладц > Utilities вибрати варіант > Use > Target > Driver > for > Flash Programming і також встановити значення ST-Link Debugger. Перевірити, що в вікні після натискання кнопки Settings значення збігаються з заданими для налагодження.

Після проведення даних дій можна приступати до написання програми та проведення налагодження на демонстраційній платі. Однак перед цим слід підключити плату до ПК.

Для роботи з платою необхідно, щоб пройшла установка драйверів.

Зазвичай, при установці засобів розробки, наприклад, Keil, відбувається також установка супутнього програмного забезпечення (якщо цього не сталося, тослід знайти в директорії установки необхідні програми і встановити їх самостійно), тому рекомендується проводити такі дії вже після установки засобів розробки. Якщо ж варіант з установкою середовища розробки не підходить, то слід завантажити і встановити програму STM32 ST-Link Utility, яка також розглядається в методичних вказівках.

При підключенні пристрою відразу ж починається установка драйверів для нового пристрою.

Порядок виконання роботи

1. Озайомитися із структурною проєкту в середовищі Keil.
2. Створити проект в середовищі Keil.
3. Візкомпілювати проєкт.
4. Представити результати відкомпільованого коду програми.

Зміст звіту

1. Мета роботи.
2. Відомості про порядок створення проекту..
3. Організація інтерфейсу середовища для програмування.
4. Висновки.

Контрольні питання

1. Від чого походить назва ARM?
2. На чому спеціалізується компанія "Advanced RISC Machines"?
3. Які компанії є клієнтами компанії ARM?
4. Які основні характеристики ядра ARM7?
5. Назвіть основні особливості мікроконтролера LPC2378.
6. Опишіть відмінності мікроконтролера LPC2378 від інших мікроконтролерів ряду LPC23xx.
7. Охарактеризуйте детальніше такі особливості мікроконтролера LPC2378 як: структурна схема; швидка периферія і ядро контролера; зв'язок з периферією; шина AHB1; шина AHB2; міст AHB–to–AHB.
8. Чим займається компанія STMicroelectronics?
9. На основі якого ядра виконані мікроконтролери STM32?
10. Охарактеризуйте склад та основні характеристики мікроконтролера STM32.
11. За якою архітектурою виконано мікроконтролер STM32?
12. Опишіть сімейство STM32.

ПРАКТИЧНА РОБОТА № 3 СТВОРЕННЯ ПРОЕКТУ В СЕРЕДОВИЩІ РОЗРОБКИ. ВИКОРИСТАННЯ ПОРТІВ ВВЕДЕННЯ / ВИВОДУ

Мета роботи - вивчення структури процесора ARM.

Загальні відомості

Контролер STM32F407VG містить п'ять 16-розрядних портів вводу/виводу загального призначення, які позначені як GPIOx, де x може мати значення A, B, C, D, E. Кожен порт GPIO має чотири 32-розрядних регістри конфігурації (GPIOx_MODER, GPIOx_TYPER, GPIOx_SPEEDR, GPIOx_ORD), два 32- розрядних регістри даних (GPIOx_ODR, GPIOx_IDR) і два 32-бітових регістра вибору додаткових функцій (GPIOx_AFRH і GPIOx_AFRL).

Світлодіоди, призначені для програмування, на платі підключені до порту D (рис. 3.1).

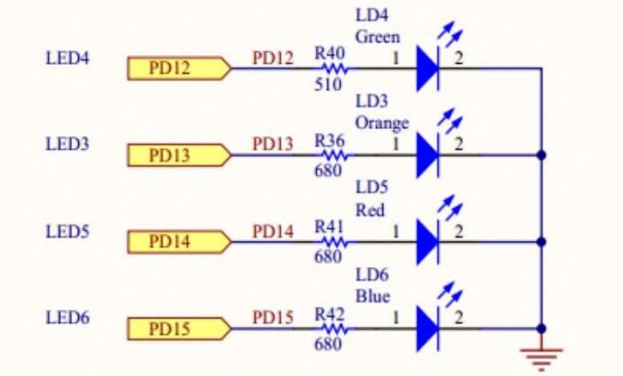


Рисунок 3.1 - Фрагмент принципової електричної схеми платі налагодження

Зі схеми видно, що для світіння світлодіодів потрібно через виходи мікроконтролера PD12-PD15 подавати напругу +, що відповідає логічному стану “1”. При поданні на вихід логічного стану “0” світіння спостерігатися не буде.

Решта чотири світлодіода виконують службові функції індикації та в програмуванні певних дій не використовуються.

Початківцям рекомендується встановити середовище розробки CooCox CoIDE 1.7. Створення проекту виконується за допомогою майстра, який відкривається при виконанні команди меню Project / New Project. Далі слід покроково вказати ім'я проекту і його розташування, виробника і МК, для якого призначена програма. Далі робота з проектом здійснюється за допомогою вікна Repository, яка дозволяє додати необхідні бібліотеки управління периферійними частинами МК, а також вікно Project. Вікно Repository також є майстер, який містить кілька вкладок, основний з яких є вкладка Peripherals (рис. 3.2).

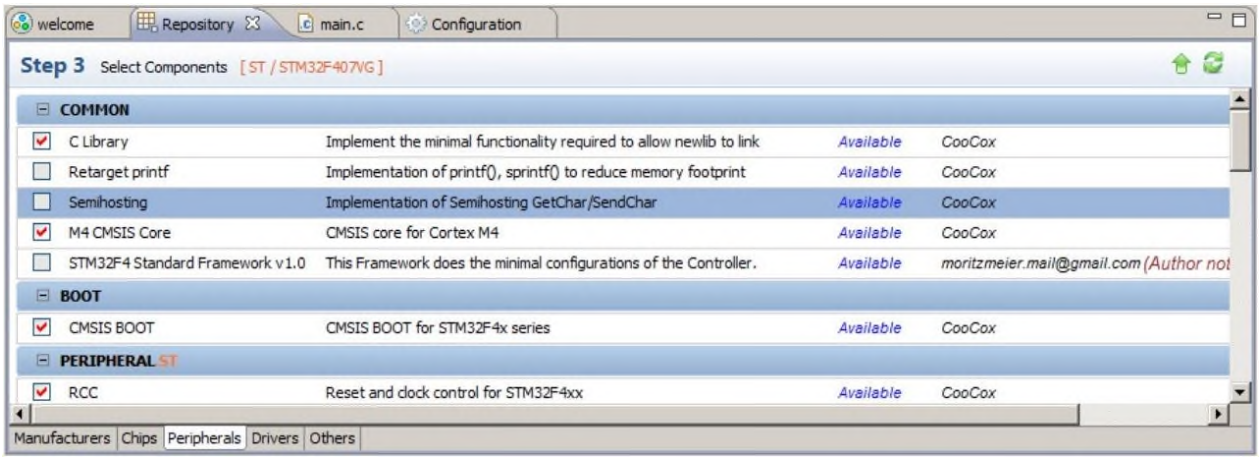


Рисунок 3.2 - Додавання модулів до проекту

Проект програми для плати STM32F4 Discovery має схожу структуру для всіх засобів розробки. Зазвичай він включає в себе два внутрішніх каталоги: один для файлів з бібліотеки CMSIS, а інший для файлів роботи з периферією. Для прикладу представлені типові структури проектів в середовищах розробки CoIDE і Keil (рис. 3.3).

Розглянемо приклад програми, яка демонструє роботу з портами введення/виводу. вона дозволяє перемикати стану світлодіода при натисканні кнопки, яка знаходиться на платі. Кнопка замикає через додатковий опір вихід A0 на джерело живлення, тим самим подає високий логічний рівень в момент утримання кнопки.

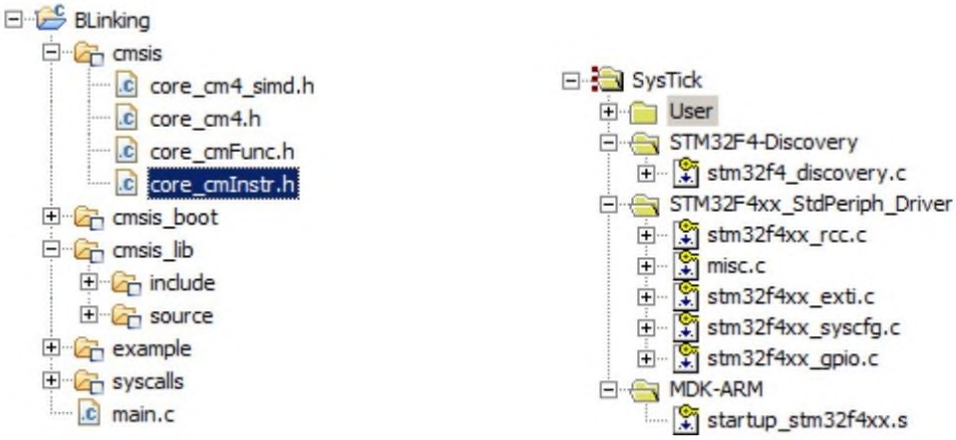


Рисунок 3.3 - Дерево проектів в CoIDE та Keil

Розглянемо детально наступний проект.

На початку файлу main.c оголошуємо використання наступних бібліотек:

#include "stm32f4xx.h"

Бібліотека містить оголошення регістрів конкретного процесору.

#include "stm32f4xx_rcc.h"

Бібліотека містить структури та функції керування тактуванням периферійних пристроїв мікроконтролеру.

#include "stm32f4xx_gpio.h"

Наступна бібліотека містить керування виходами мікросхеми.

const int LED1 = GPIO_Pin_12;

const int LED2 = GPIO_Pin_13;

Для зручності позначимо виходів мікросхеми, які з'єднано до світлодіодів, константами. Це дає переваги при зміні процесору або пристрою, тоді потрібно змінити в програмі значення констант, і програма знову придатна для використання на новому пристрої.

Далі по коду потрібно реалізувати функції, які буде використовувати програма. Реалізуємо організацію пауз для уповільнення роботи окремих ділянок програмного коду, для цього використаємо пустий цикл. Цей цикл змушує процесор працювати “в холосту”, при цьому фактична пауза може

значно різнитися від налаштувань процесора. Слово volatile використано для позначення компілятору, що спрощувати вирази з цією змінною не можна, її значення може змінитися непередбачувано, наприклад, у перериванні. Завдяки цьому при увімкненому оптимізаторі коду, компілятор не замінить цей цикл на просте присвоєння нуля змінній.

```
void Delay(volatile unsigned int nCount)
{
while(nCount--);
}
```

Наступна функція призначена для налаштування виходу з мікросхеми PA0

як приймач інформації. Для цього використана структура GPIO_InitTypeDef, яка містить опис зрозумілою мовою всіх властивостей виходів мікросхеми (але керування альтернативними функціями винесено окремо). Далі за допомогою RCC команди подається тактувальний сигнал до периферійного пристрою GPIOA, після чого його можна використовувати. Розглянемо послідовно властивості виходу мікросхеми.

gpioConf.GPIO_Pin = GPIO_Pin_0 — вказується що тут використовується лише один вихід мікросхеми, а саме за номером нуль.

gpioConf.GPIO_Speed = GPIO_Speed_100MHz — швидкість спрацювання зміни логічного рівня, чим швидше працює вихід, тим більше він споживатиме енергії.

gpioConf.GPIO_PuPd = GPIO_PuPd_DOWN — керує значенням яке читається на вході по умовчанню, коли вихід не приєднано до джерела сигналу. Коли кнопка не натиснута, потрібна гарантія надходження логічного нуля, тому вихід “придушено” на “землю”.

gpioConf.GPIO_OType = GPIO_OType_PP — визначає метод керування

напругою на виході.

gpioConf.GPIO_Mode = GPIO_Mode_IN — задає направлення руху інформації, з кнопки до мікросхеми надходить інформація — лапка є входом інформації.

На останнє, проводиться застосування вказаних налаштувань:

```
void ButtonConfig(){ GPIO_InitTypeDef gpioConf;
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
gpioConf.GPIO_Pin = GPIO_Pin_0;
```

gpioConf.GPIO_Speed = GPIO_Speed_100MHz; gpioConf.GPIO_PuPd = GPIO_PuPd_DOWN; gpioConf.GPIO_OType = GPIO_OType_PP; gpioConf.GPIO_Mode = GPIO_Mode_IN; GPIO_Init(GPIOA, &gpioConf);

}

Наступна функція відповідає на налаштування виходів мікросхеми на керований вивід напруги -керування світлодіодами. Тіло функції є аналогічним, але тут вказується одночасно два виходи, які поєднуються операцією логічного бітового додавання. Також відмінним є вимкнена підтяжка виходів, бо невизначеності читання стану тут немає.

void LedConfig(){ GPIO_InitTypeDef gpioConf;

RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE); gpioConf.GPIO_Pin = LED1 | LED2;

gpioConf.GPIO_Mode = GPIO_Mode_OUT; gpioConf.GPIO_Speed = GPIO_Speed_100MHz; gpioConf.GPIO_OType = GPIO_OType_PP; gpioConf.GPIO_PuPd = GPIO_PuPd_NOPULL; GPIO_Init(GPIOD, &gpioConf);

}

Наступні більшість функцій є короткими та мають призначення в заміні загального коду на змістовні назви. Компілятор визначає складність функцій, і коли це є вигідним, замість виклику функцій буде вставляти сам код. Тому уповільнення роботи програми за черезмірного розбиття на функції немає. В цьому коді проходить звернення до PA0 з визначенням напруги. Якщо на вхід подається напруга, функція вертатиме 1, інакше — 0.

int GetButtonState(){

return GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0);

}

Наступна функція не може завершити цикл доки стан кнопки повертає 1, тобто, вийти з цього циклу програма зможе лише при відпусканні кнопки: void WaiteButtonRelease(){

while(GetButtonState()!=0);

}

Далі функція визначає повний цикл натискання кнопки, натиснення та відпускання. Лише після цих двох подій функція передасть виконання на основну програму:

void WateClickButton(){ while(GetButtonState()==0); Delay(10000); WaiteButtonRelease();

}

Використання команд встановлення напруги на виході є простим, але для світлодіодів, матриць та двигунів така команда виглядає однаково. Тому оформимо команду увімкнення світлодіоду:

void LedOn(int LED){ GPIO_SetBits(GPIOD, LED);

}

Та вимкнення світлодіоду:

void LedOff(int LED){

GPIO_ResetBits(GPIOD, LED);

}

Додатковим бонусом до такого стилю запису команд керування світлодіодами є те, що в деяких схемах світлодіоди світяться при поданні на вихід низького рівня. В такому випадку в програмному коді потрібно переписати лише дві функції, а в противному випадку — код потрібно змінювати в усіх випадках керування світлодіодами по тексту програми.

Розглянемо головну функцію програми:

```
int main(void)
{
SystemInit(); SystemCoreClockUpdate(); ButtonConfig(); LedConfig();
while(1) {
if(GetButtonState()==0){ LedOff(LED1);
}else{
LedOn(LED1);
}
}
}
```

Завдяки описаним вище функціям, програма читається легко. Розглянемо лише дві перші функції. SystemInit() - відповідає за налаштування тактування процесору. Без виклику цієї функції процесор працюватиме на частоті по замовчанню — 16МГц. SystemCoreClockUpdate() - відповідає лише за розрахунок дійсної частоти процесора. Ця константа може бути використана для розрахунку частот інтерфейсів та таймерів, щоб програма рахувала час при різних налаштуваннях правильно.

В основному циклі програми відбувається зчитування стану кнопки, іякщо кнопка натиснена, вмикається світлодіод.

Наступний головний цикл перемикає світлодіоди по натисканні кнопки:

```
while(1) {
LedOn(LED1);
LedOff(LED2); WateClickButton(); LedOff(LED1); LedOn(LED2);
WateClickButton();
}
```

Порядок виконання роботи

1. Озайомитися із структурною проєкту що приведений вище.
2. На основі отриманих даних виконайте самостійне завдання, коли при натиснканні користувацької кнопки, програма буде перемикати світлодіод.
3. Задійте всі чотири світлодіоди.
4. Представити результати відкомпільованого коду програми.

Зміст звіту

1. Мета роботи.
2. Відомості про порядок створення проекту..
3. Організація інтерфейсу середовища для програмування.
4. Висновки.

Контрольні питання

1. Які модулі відносяться до системної периферії?

2. Що представляє собою пам'ять кожного з пристроїв сімейства LPC2300 з точки зору програміста?

3. Які шини використовуються в мікроконтролерах сімейства LPC2300 для взаємодії ядра та периферійних пристоїв?

4. Які системні тактові сигнали використовуються в мікроконтролерах сімейства LPC2300 та як вони формуються?

5. Назвіть режими зниженого енергоспоживання в мікроконтролерах сімейства LPC2300.

6. Дайте характеристику режиму зниженого енергоспоживання Idle.

7. Дайте характеристику режиму зниженого енергоспоживання Power Down.

8. Дайте характеристику режиму зниженого енергоспоживання Deep Power Down.

9. Дайте характеристику режиму зниженого енергоспоживання Sleep.

10. В якому стані знаходяться периферійні пристрої після скидання?

11. Який регістр служить для керування включенням/відключенням тактових сигналів периферійних пристроїв?

12. Які периферійні пристрої за замовчуванням після скидання знаходяться в вимкненому стані?

13. В якому режимі зниженого енергоспоживання досягається найменше енергоспоживання мікроконтролера?

14. Як може відбуватися вихід з режиму Sleep?

15. Які регістри використовуються для керування системним тактовим сигналом?

16. До якої шини підключено ядро ARM7?

ПРАКТИЧНА РОБОТА № 4 ПЕРЕРИВАННЯ ТА ВИКОРИСТАННЯ ТАЙМЕРІВ

Мета роботи - вивчення переривання та використання таймерів.

Загальні відомості

Переривання - механізм, який дозволяє апаратному забезпеченню повідомляти про настання подій у своїй роботі. У момент, коли відбувається переривання, процесор перемикається з виконання основної програми на виконання відповідного обробника переривань. Як тільки виконання обробника завершено, триває виконання основної програми з місця, в якому вона була перервана.

Для використання переривань необхідно спочатку налаштувати регістр, який називається Nested Vector Interrupt Controller (NVIC), вбудований в контролер контролер переривань. Цей регістр є стандартною частиною архітектури ARM і зустрічається на всіх процесорах, незалежно від виробника. NVIC розроблений таким чином, що затримка переривання мінімальна. NVIC підтримує вбудоване переривання з 16-ма рівнями пріоритету.

Мікроконтролер STM32F407VG містить 14 таймерів. У зяагальному вигляді схема управління рахування імпульсів може бути представлена наступним чином (рис. 4.1):

Виробник розділяє всі таймери на три типи:

1) з розширеними можливостями;
2) загального призначення;
3) базові.

Кожен таймер може мати до 4 ліній захоплення / порівняння (саме вони використовуються в режимі генерації ШІМ).

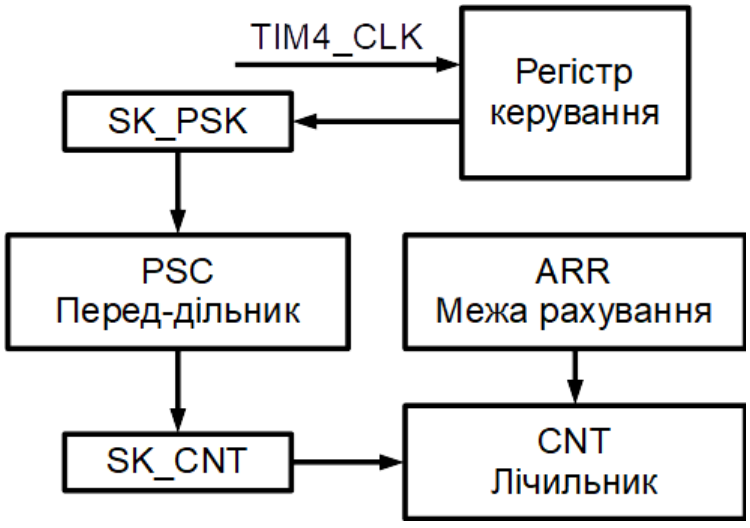


Рисунок 4.1 - Схема керування тактуванням таймеру

Наступна програма демонструє роботу з перериваннями і з таймерами. У ній реалізовано перемикання світлодіода, який підключений до порту введення/

виведення, через кожну секунду. Для цього попередню програму доповнимо функціями:

```
void LedToggle(int LEDS){ GPIO_ToggleBits(GPIOD, LEDS);
}
```

Функція виконує перемикання стану виходу на протилежне значення,

якщо на вихід подавалася напруга то вона вимкнеться, якщо ж напруги не було - напруга буде подана.

```
void TIM2_IRQHandler(void) {
if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET) {
TIM_ClearITPendingBit(TIM2, TIM_IT_Update); LedToggle(LED2);
}
}
```

Наведена функція має фіксовану назву обробника переривання. Всі доступні обробники переривань можна переглянути у файлі startup секції. Для того щоб визначити власний обробник переривання, достатньо створити функцію з відповідною назвою. Також для багатьох пристроїв переривання може викликатися по різним подіям, тому в обробнику передбачено умовний оператор в якому перевіряється факт переповнення лічильнику, лише після цього становиться ознака, що переривання оброблено, а потім змінюється стан другого світлодіоду.

Наступна функція складається з двох логічних частин, налаштування дозволу на переривання та налаштування таймеру:

```
void INTTIM_Config(void)
{
NVIC_InitTypeDef nvic_struct; nvic_struct.NVIC_IRQChannel =
TIM2_IRQn; nvic_struct.NVIC_IRQChannelPreemptionPriority = 0;
nvic_struct.NVIC_IRQChannelSubPriority = 1;
nvic_struct.NVIC_IRQChannelCmd = ENABLE; NVIC_Init(&nvic_struct);

TIM_TimeBaseInitTypeDef tim_struct;
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
tim_struct.TIM_Period = 10000 - 1; tim_struct.TIM_Prescaler =
SystemCoreClock/10000 - 1; tim_struct.TIM_ClockDivision =
TIM_CKD_DIV1; tim_struct.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM2, &tim_struct); TIM_ITConfig(TIM2,
TIM_IT_Update, ENABLE); TIM_Cmd(TIM2, ENABLE);
}
```

Налаштування дозволу переривання відбувається викликом функції NVIC_Init, для якої підготовлені налаштування в структурі NVIC_InitTypeDef. Для по використанню в лістингу функції, вказується вектор переривання, пріоритет, підпріоритет та команда задіяння переривання.

Наступний блок відповідає за роботу таймеру. Після увімкнення тактування таймеру заповнюється структура в якій зазначено, що період лічильника є 10000-1=9999, це означає доступність значення лічильника в межах 0..9999 - десять тисяч циклів (бо нульовий стан теж потрібно враховувати). Наступним параметром налаштовується передлічильник, який фактично виступає в ролі дільника частоти.

По досягнені попереднього лічильника вказаного значення, його таймер скидає на нуль та на лічильник відправляє імпульс. Тут SystemCoreClock/10000-1 гарантує, що 10000 імпульсів, які складають період, вкладатимуться в одну секунду. Це означає щосекундний виклик обробника переривань (якщо частота процесору налаштовано правильно).

Далі застосовуються налаштування, вмикається генерування події переповнення лічильника та подається команда на увімкнення таймеру. Саме з цього моменту починається рахування часу та виклик обробника переривання.

Тіло програми тепер матиме вигляд:

```
int main(void)
{
SystemInit(); SystemCoreClockUpdate(); LedsInit(LED1 | LED2 | LED3 | LED4); LedOff(LED1 | LED2 | LED3 | LED4);
INTTIM_Config();
while(1){
LedOn(LED1); LedOff(LED3); Delay(500000); LedOn(LED3); LedOff(LED1); Delay(500000);
}
}
```

Ця програма мало відрізняється від попередньої, додано лише виклик налаштування обробника переривання, після якого відбувається увімкнення світлодіоду кожні дві секунди.

Порядок виконання роботи

1. Озайомитися із структурною проєкту що приведений вище.
2. На основі отриманих даних виконайте самостійне завдання, додайте виклик налаштування обробника переривання, після якого відбувається увімкнення світлодіоду кожні дві секунди..
3. Задійте всі чотири світлодіоди.
4. Представити результати відкомпільованого коду програми.

Зміст звіту

1. Мета роботи.
2. Відомості про порядок створення проекту.
3. Налаштування таймера на розрахований час переривання.
4. Висновки.

Контрольні питання

1. Скільки таймерів є у складі мікроконтролерів сімейства LPC2300?
2. Опишіть спрощену структуру таймера загального призначення.
3. Від чого залежить швидкість лічби таймера?
4. Скільки каналів захоплення має кожен таймер?
5. Опишіть роботу таймера в режимі захоплення.
6. Опишіть роботу таймера в режимі лічильника зовнішніх подій.
7. Опишіть роботу таймера в режимі збігу.

8. Скільки каналів збігу має кожен таймер?

9. Назвіть та опишіть керуючі регістри модуля таймерів.

10. Дайте опис бітів регістрів переривань.

11. Дайте опис бітів регістрів керування таймерами.

12. Дайте опис бітів регістрів керування підрахунком.

13. Дайте опис бітів регістрів таймерів/лічильників.

14. Дайте опис бітів регістрів масштабування та лічильників масштабування.

15. Дайте опис бітів регістрів збігу та керування збігом.

16. Дайте опис бітів регістрів захоплення та керування захопленням.

17. Дайте опис бітів регістрів зовнішнього збігу.

18. Опишіть структуру модуля таймерів/лічильників.

19. За яким фронтом зовнішнього сигналу може відбуватися подія захоплення?

20. Як може змінюватися сигнал на відповідному виході при настанні події збігу?

21. Які прапорці переривань можуть використовуватись при програмуванні роботи таймерів?

ПРАКТИЧНА РОБОТА № 5 ГЕНЕРУВАННЯ СИГНАЛУ ШИРОТНО-ІМПУЛЬСНОЇ МОДУЛЯЦІЇ

Мета роботи - вивчення генерування сигналу широтно-імпульсної модуляції.

Загальні відомості

Широтно-імпульсна модуляція (ШІМ) - спосіб управління середнім значенням напруги на навантаженні шляхом зміни ширини імпульсів. В основному, мікроконтролери дозволяють генерувати цифровий ШІМ різної частоти.

Мікроконтролер призначений для керування зовнішніми пристроями за алгоритмами, які вираховують сигнали керування за отриманими з датчиків даними. Деякі пристрої вимагають для керування формування сигналів широтної імпульсної модуляції (ШІМ). Зміст цього сигналу показано на наступному малюнку:

ШІМ сигнал формується одиничним імпульсом напруги з певною паузою.

При цьому час T=const, період сигналу є величиною постійною, а за цей час розподіл T_1, T_0 може змінюватися. Та завжди є справедливою сума $T=T_1+T_0$.

Такий сигнал може використовуватися для регулювання світимості світлодіоду або потужності двигуна постійного струму. Відомо, що світлодіод має порогову напругу живлення, після якої він починає майже одразу світитися на повну потужність, і з метою запобігання перегорання, струм живлення та яскравість світіння обмежують додатковим опором. Для регулювання світності світлодіоду потрібно використати змінний опір, але опір мікроконтролером регулювати є нетривіальною задачею, тому світлодіод використовують в режимі максимальної потужності, але живлять його пульсуючим струмом ШІМ сигналу. Фактично за час T світлодіод на повну потужність світить час T_1, тобто $100T_1/T$ % часу світлодіод є джерелом світла. Для малого періоду T, коли за одну секунду відбувається більше 1000 спалахів, людське око сприймає такий режим як плавну зміну яскравості світлодіоду. Ще однією з переваг такої схеми живлення є більш високе ККД використання електричної енергії, зменшивши її витрачання на додаткових опорах.

Перед підключенням пристрою відомі такі параметри ШІМ, як частота і коефіцієнт заповнення. Для їх розрахунку в контролерах STM32 необхідно визначити значення преддільника і автоматично завантажувати значення в регістрі ARR (Auto-Reload Register). Розрахунок значення, яке слід записати в переддільник виконується наступним чином:

$$PSC = \frac{TIMxCLK}{TIMxCNT} - 1,$$

де *PSC* – значення переддільника;

TIMxCLR – вихідна частота роботи таймеру;

TIMxCNT – частота лічильника.

Для отримання необхідної вихідної частоти слід записати значень в регістр ARR, яке повчає з наступного співвідношення:

$$ARR\_VAL = \frac{TIMxCNT}{TIMx\_вихідна\_частота} - 1,$$

де *ARR_VAL* – значення для запису в регістр *ARR*; *TIMx_вихідна_частота* – вихідна частота роботи таймеру; *TIMxCNT* – частота лічильника.

Останнім етапом є завдання потрібного коефіцієнта заповнення, що забезпечить потрібне заповнення імпульсу. Ця процедура проводиться за допомогою регістра захоплення/порівняння (Capture / compare register, CCRx), виходячи з наступного співвідношення:

$$Z = \frac{CCRx\_VAL}{ARR\_VAL} 100\%,$$

де *Z* – коефіцієнт заповнення, *CCRxVAL* – значення регістру *CCRx*.

Особливістю даних мікроконтролерів є те, що в переддільник і інші регістри можна записати будь-яке значення, яке можна описати з допомогою відведеного кількості розрядів. Видача сигналу ШІМ на вихід не є основним режимом роботи висновків порту, а відноситься до додаткових (альтернативним) режимам. Тому попередньо потрібно задати потрібний режим в налаштуваннях порту. Для підключення альтернативних функцій до портів введення / виводу необхідно:

1. Підключити вихід до альтернативної функції відповідного периферійного пристрою за допомогою функції GPIO_PinAFConfig.
2. Конфігурувати вихід в режим виконання альтернативної функції - GPIO_Mode_AF.
3. Виконати інші налаштування.
4. Викликати GPIO_Init для застосування зазначених налаштувань.

Розглянемо програмну реалізацію:

```
#include "stm32f4xx.h" #include "stm32f4xx_rcc.h" #include "stm32f4xx_gpio.h" #include "stm32f4xx_tim.h" #include "misc.h"
```

Наступна функція відповідає за налаштування четвертого таймеру на генерування ШІМ по чотирьом каналам:

```
void PWM_Init(){
TIM_TimeBaseInitTypeDef time_init;
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE); uint16_t PrescalerValue = (uint16_t)((SystemCoreClock / 2) / 21000000);
time_init.TIM_Period = 665;
time_init.TIM_Prescaler = PrescalerValue;
time_init.TIM_ClockDivision = TIM_CKD_DIV1;
time_init.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM4, &time_init);

TIM_OCInitTypeDef oc_init; oc_init.TIM_OCMode = TIM_OCMode_PWM1;
oc_init.TIM_OutputState = TIM_OutputState_Enable;
oc_init.TIM_Pulse = 0;
oc_init.TIM_OCPolarity = TIM_OCPolarity_High;
TIM_OC1Init(TIM4, &oc_init); TIM_OC1PreloadConfig(TIM4, TIM_OCPreload_Enable); TIM_OC2Init(TIM4, &oc_init);
```

```
TIM_OC2PreloadConfig(TIM4, TIM_OCPreload_Enable);
TIM_OC3Init(TIM4, &oc_init); TIM_OC3PreloadConfig(TIM4,
TIM_OCPreload_Enable); TIM_OC4Init(TIM4, &oc_init);
TIM_OC4PreloadConfig(TIM4, TIM_OCPreload_Enable);
TIM_ARRPreloadConfig(TIM4, ENABLE); TIM_Cmd(TIM4, ENABLE);
    }
```

В першу чергу командою RCC_APB1PeriphClockCmd подається тактування на четвертий таймер. Далі розраховується значення передлічильника PrescalerValue так, щоб частота ШІМ імпульсів не залежала від тактової частоти процесора. Далі встановлюємо періодичність імпульсів в 665 тактових імпульси з передлічильника. Інші налаштування є аналогічними що до налаштування таймеру 2, який розглянуто в попередньому прикладі.

Далі в функції PWM_Init проводиться налаштування виходів ШІМ, окремо для кожного з каналів. Для цього проводиться заповнення структури TIM_OCInitTypeDef. IM_OCMode приймає значення TIM_OCMode_PWM1, бо саме цей режим відповідає за генерування неперервних імпульсів заданої ширини.

TIM_OutputState приймає значення TIM_OutputState_Enable для увімкнення подання сигналу на вихід мікроконтролера. TIM_Pulse = 0 визначає початкову ширину сигналу, а TIM_OCPolarity = TIM_OCPolarity_High визначає, що регулювання відбувається шириною саме початкового одиничного імпульсу, а до кінця періоду буде виводитися нульовий рівень. Після цього застосовуються налаштування на кожен з каналів. Завершує функцію вмикання четвертого таймеру, після чого починається генерування сигналу.

Налаштування виходів до світлодіодів відрізняється від простого використання GPIO:

```
    void LedsPWMInit(){ GPIO_InitTypeDef gpio_init;
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
gpio_init.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 |
    GPIO_Pin_15;
    gpio_init.GPIO_Mode = GPIO_Mode_AF; gpio_init.GPIO_Speed =
GPIO_Speed_100MHz; gpio_init.GPIO_OType = GPIO_OType_PP;
gpio_init.GPIO_PuPd = GPIO_PuPd_UP; GPIO_Init(GPIOD, &gpio_init);
    GPIO_PinAFConfig(GPIOD,    GPIO_PinSource12,    GPIO_AF_TIM4);
GPIO_PinAFConfig(GPIOD,      GPIO_PinSource13,      GPIO_AF_TIM4);
GPIO_PinAFConfig(GPIOD,      GPIO_PinSource14,      GPIO_AF_TIM4);
GPIO_PinAFConfig(GPIOD, GPIO_PinSource15, GPIO_AF_TIM4);
    }
```

Тут режим роботи обрано як GPIO_Mode_AF, що означає використання виходу в одному з альтернативних режимів. Після застосування налаштувань виходів до світлодіодів, вказується для кожного з виходів, яку саме альтернативну функцію вони будуть використовувати.

Наступна змінна показую фазу розподілу яскравості між світлодіодами:

```
    volatile int faze = 0;
```

Наступна функція є обробником переривання від другого таймеру, вона використана для зміни фази світіння світлодіодів та запису ширини імпульсів у

регістри CCRx таймеру, завдяки цьому ширина імпульсів по чотирьом каналам є різною:

```
void TIM2_IRQHandler(void) {
if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET) {
TIM_ClearITPendingBit(TIM2, TIM_IT_Update); faze = (faze+1)%665;
TIM4->CCR1 = (faze+665*0/4)%665; TIM4->CCR2 = (faze+665*1/4)%665; TIM4->CCR3 = (faze+665*2/4)%665; TIM4->CCR4 = (faze+665*3/4)%665;
}
}
```

Налаштування переривання INTTIM_Config є аналогічним до попереднього прикладу та відрізняється лише частотою спрацювання:

```
void INTTIM_Config(void)
{
NVIC_InitTypeDef nvic_struct; nvic_struct.NVIC_IRQChannel = TIM2_IRQn; nvic_struct.NVIC_IRQChannelPreemptionPriority = 0;
nvic_struct.NVIC_IRQChannelSubPriority = 1;
nvic_struct.NVIC_IRQChannelCmd = ENABLE; NVIC_Init(&nvic_struct);

TIM_TimeBaseInitTypeDef tim_struct;
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
tim_struct.TIM_Period = 400 - 1; tim_struct.TIM_Prescaler = SystemCoreClock/100000 - 1; tim_struct.TIM_ClockDivision = TIM_CKD_DIV1; tim_struct.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM2, &tim_struct); TIM_ARRPreloadConfig(TIM2, ENABLE); TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE); TIM_Cmd(TIM2, ENABLE);
}
```

В результаті головна функція містить лише налаштування ШІМ сигналу та переривання для змін яскравості. Всі події генеруються апаратно та за допомогою переривання, в результаті цикл можна зайняти іншими корисними задачами:

```
int main(void)
{
SystemInit(); SystemCoreClockUpdate(); LedsPWMInit();
PWM_Init(); INTTIM_Config(); while(1){

}
}
```

Порядок виконання роботи

1. Озайомитися із структурною проєкту що приведений вище.
2. На основі отриманих даних виконайте самостійне завдання, додайте налаштування параметрів генерування ШІМ-сигналу.
3. Задійте світлодіоди для відображення ШІМ-сигналу.

4. Представити результати відкомпільованого коду програми.

Зміст звіту

1. Мета роботи.
2. Відомості про порядок створення проекту.
3. Налаштування таймера на розрахований час генерації ШІМ-сигналу.
4. Висновки.

Контрольні питання

1. Дайте визначення широтно–імпульсній модуляції.
2. Як розраховується шпаруватість ШІМ–сигналу?
3. Які ШІМ–сигнали може формувати модуль ШІМ?
4. Який таймер загального призначення виконує функцію модуля ШІМ?
5. Який механізм введено в модуль ШІМ з метою недопущення спотворення ШІМ–сигналу?
6. В які моменти періоду формування ШІМ–сигналу вступають в силу нові значення при оновленні відповідного каналу?
7. Який регістр та як використовується при реалізації механізму недопущення спотворення ШІМ–сигналу?
8. Які відмінності існують у функціонуванні каналів збігу модуля ШІМ крім наявності тіньових регістрів–защіпок?
9. Чим відрізняється схема керування виводами модуля ШІМ від звичайних таймерів?
10. Поясніть роботу схеми керування виводами в модулі ШІМ.
11. Поясніть особливості широтно–імпульсної модуляції за одним фронтом.
12. Поясніть особливості широтно–імпульсної модуляції за обома фронтами.
13. Які регістри використовуються для керуванням модулем ШІМ?
14. Поясніть використання модуля ШІМ в режимі лічильника.
15. Яка тактова частота використовується модулем ШІМ?
16. Скільки регістрів збігу має модуль ШІМ?
17. Як можуть змінюватись фронти вихідних імпульсів в модулі ШІМ при настанні події збігу?
18. Який регістр використовується для керування шпаруватістю?
19. Який регістр використовується для керування положенням фронту PWM?
20. Який регістр використовується для керування частотою ШІМ–сигналу?

.

ПРАКТИЧНА РОБОТА № 6 ВИКОРИСТАННЯ АНАЛОГОВО-ЦИФРОВОГО ПЕРЕТВОРЮВАЧА

Мета роботи - вивчення використання аналогово-цифрового перетворювача.

Загальні відомості

Мікроконтролер STM32F407VG включає в себе три АЦП. Розрядність всіх АЦП становить 12 біт. Кожен перетворювач здатний приймати сигнал з шістнадцяти зовнішніх каналів. Крім того, до складу контролера входить датчик температури. Діапазон вхідної напруги від датчику температури становить 1.8...3.6В. Датчик температури підключений до вхідного каналу ADC_IN16, який використовується для того, щоб перетворити вихідну напругу сенсора в цифрове значення. Внутрішній датчик температури призначений лише для

відстеження зміни температури, а не для її вимірювання, оскільки зсув показників датчика може змінюватися в ході змін параметрів процесу. Тому, якщо необхідно точне вимірювання абсолютних значень температури, то для цього краще використовувати зовнішній датчик, який призначений саме для точного вимірювання температур.

Для того, щоб правильно визначити подану напругу, необхідно провести додаткові вимірювання. Для цього за допомогою вольтметра визначаємо напругу, яка відповідає 0b111111111111=4095, підключивши вольтметер до відповідних виходів подачі живлення на процесор. Наприклад, якщо вольтметр показав, що в напруга дорівнює 2.96В, записуємо даний коефіцієнт безпосередньо в код програми.

Розрахунок напруги здійснюється за наступною формулою:

$$U = \frac{U_{ref} \cdot ADC}{ADC_{\max}},$$

де U_{ref} – виміряне значення напруги живлення;

ADC – показник аналогово-цифрового перетворювача;

ADC_{max} – максимально можливий показник аналогово-цифрового перетворювача, в нашому випадку для процесорів STM32F… має значення 4095.

Наступний програмний код демонструє використання цифрового вольтметру в режимі вимірювання по запиту. Звісно, існують режими автоматичного запису виміряних значень з кількох каналів по DMA до оперативної пам'яті в фоновому режимі, однак одноразове вимірювання дозволяє використати вимірювання в той час коли воно саме потрібно з більш простими налаштуваннями та використанням меншої кількості бібліотек:

```
#include <stm32f4xx_rcc.h>
#include <stm32f4xx_gpio.h>
#include <stm32f4xx_adc.h>

void Led_init(){
GPIO_InitTypeDef gpio;
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
gpio.GPIO_OType = GPIO_OType_PP;
```

```
gpio.GPIO_PuPd = GPIO_PuPd_NOPULL; gpio.GPIO_Mode =
GPIO_Mode_OUT; gpio.GPIO_Pin = GPIO_Pin_12; gpio.GPIO_Speed =
GPIO_Speed_100MHz; GPIO_Init(GPIOD, &gpio);
}

void Led_on(){
GPIO_SetBits(GPIOD, GPIO_Pin_12);
}

void Led_off(){
GPIO_ResetBits(GPIOD, GPIO_Pin_12);
}
```

Перші три наведені функції знайомі по попереднім прикладам і нічого нового не містять в своєму коді. Новою є наступна функція:

```
void Adc_gpio_init() { GPIO_InitTypeDef gpio;
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
GPIO_StructInit(&gpio);
gpio.GPIO_Pin = GPIO_Pin_0;
gpio.GPIO_Mode = GPIO_Mode_AIN;//GPIO_Mode_AN
GPIO_Init(GPIOA, &gpio);
}
```

Тут проводиться налаштування виводу мікросхеми на роботу як аналоговий вхід. Від відомих до цього налаштувань тут використано функцію заповнення структури налаштування на значення по замовчанню GPIO_StructInit, далі новим є лише режим роботи лапки мікросхеми як аналоговий вхід GPIO_Mode_AIN.

Наступна функція відповідає безпосередньо за налаштування цифрового вольтметру:

```
void Adc_init() {
ADC_InitTypeDef adc; ADC_CommonInitTypeDef adc_init;
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
ADC_DeInit();

ADC_StructInit(&adc);
adc_init.ADC_Mode = ADC_Mode_Independent;
adc_init.ADC_Prescaler = ADC_Prescaler_Div8;

adc.ADC_ScanConvMode = ENABLE;
adc.ADC_ContinuousConvMode = ENABLE; adc.ADC_ExternalTrigConv
= ADC_ExternalTrigConvEdge_None; adc.ADC_DataAlign =
ADC_DataAlign_Right; adc.ADC_Resolution = ADC_Resolution_12b;

ADC_CommonInit(&adc_init); ADC_Init(ADC1, &adc);
ADC_Cmd(ADC1, ENABLE);
}
```

Наведена функція використовує дві структури налаштувань ADC_InitTypeDef та ADC_CommonInitTypeDef. Далі розглянемо використані налаштування:

ADC_Mode = ADC_Mode_Independent – режим незалежних вимірювань.

Перелік можливих режимів можна переглянути у файлі «stm32f4xx_adc.h» в розділі defgroup ADC_Common_mode.

ADC_Prescaler = ADC_Prescaler_Div8 – визначає тактову частоту надану всім АЦП, дільник частоти може мати значення 2, 4, 6 та 8.

ADC_DMAAccessMode – відповідає за використання одного з каналів DMA, в нашому випадку DMA не використовується, тому залишається нульове значення.

ADC_TwoSamplingDelay – цей параметр відповідає за паузу між послідовними вимірами в автоматичному режимі, тут використовуються окремі вимірювання, тому значення обрано по замовчанню ADC_TwoSamplingDelay_5Cycles=0.

ADC_Resolution = ADC_Resolution_12b – відповідає за налаштування режиму роздільної здатності АЦП. Цей параметр може приймати значення 6, 8, 10 та 12 бітів.

ADC_ScanConvMode = ENABLE або DISABLE – вказує, чи конвертація виконується в багатоканальному чи одноканальному режимі.

ADC_ContinuousConvMode = ENABLE або DISABLE – вказує чи виконується вимірювання в режимі безперервного або одиночного режиму.

ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None – визначає що зовнішні синхроімпульси вимірювання не використовуються.

ADC_ExternalTrigConv – відповідає за вибір джерела зовнішніх синхроімпульсів, джерелом можуть працювати таймери або зовнішній сигнал, який подається до мікроконтролеру.

ADC_DataAlign = ADC_DataAlign_Right – задає положення отриманих значень праворуч, тобто виміряне значення належить проміжку 0..4095.

ADC_NbrOfConversion – вказує кількість вимірювань АЦП які буде зроблено для регулярної групи каналів. Цей параметр має коливатися від 1 до 16.

Останні команди задіяють обрані налаштування та вмикають ADC1.

Наступна функція виконує безпосереднє вимірювання напруги на обраному каналі. До цього відповідний вихід мікросхеми повинен бути налаштований як аналоговий вхід:

```
u16 ReadADC1(u8 channel) {
ADC_RegularChannelConfig(ADC1, channel, 1,
ADC_SampleTime_3Cycles);
ADC_SoftwareStartConv(ADC1);
while (ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
return ADC_GetConversionValue(ADC1);
}
```

Функція містить вибір каналу для вимірювання де вказано, що вимірювання здійснює ADC1, вказується канал а потім їх кількість, в разі групового вимірювання кількість буде більшою за одиницю. Останнім параметром вказано час вимірювання в циклах (деякі значення з проміжку від 3 до 480).

Наступною командою проводиться запуск вимірювання. Далі в циклі очікується прапор завершення вимірювання, після чого виміряне значення повертається з функції.

Наступна функція вже відома за минулими прикладами:

```
void Delay(volatile int Val) {
while(Val--);
}
```

Тіло програми містить функції налаштувань, після яких в головному

циклі проводиться вимірювання напруги. Виміряне значення використане для задання шпаруватості ШІМ сигналу, який подано на вбудований на плату зелений світлодіод – яскравість світлодіоду пропорційна поданій на вхід напрузі:

```
int main(void) {
Adc_gpio_init();
Adc_init();
Led_init();
unsigned int bin_code=0; float voltage=0.0f; while(1){
bin_code = ReadADC1(ADC_Channel_0); voltage = bin_code * 2.96
/ 0b111111111111;
/* voltage debug control only */ Led_on();
Delay(bin_code);
Led_off();
Delay(0b111111111111 - bin_code);
}
}
```

Порядок виконання роботи

1. Озайомитися із структурною проєкту що приведений вище.
2. На основі отриманих даних виконайте самостійне завдання, додайте вимірювання значення аналогової величини та передачу параметрів на генерування ШІМ-сигналу.
3. Задійте світлодіоди для відображення ШІМ-сигналу та аналогового значення.
4. Представити результати відкомпільованого коду програми.

Зміст звіту

1. Мета роботи.
2. Відомості про порядок створення проекту.
3. Налаштування блоку АЦП.
4. Висновки.

Контрольні питання

1. Опишіть призначення та принцип роботи АЦП.
2. Як розраховується абсолютна та відносна похибки АЦП?
3. Як робиться вибір величини кроку квантування за часом АЦП?

4. Для чого в модулях АЦП використовується пристрій вибірки –зберігання?

5. Поясніть роботу АЦП послідовного наближення.

6. Поясніть особливості АЦП у складі моделей мікроконтролерів сімейства LPC23XX.

7. Як формується тактовий сигнал модуля АЦП?

8. Які дії треба виконати у програмі керування АЦП перед його запуском?

9. Як можна змінювати розрядність результату перетворення?

10. Що відбувається при завершенні перетворення?

11. Де та в якому вигляді зберігається результат перетворення?

12. Поясніть режими функціонування АЦП.

13. Назвіть та поясніть регістри керування АЦП.

14. Опишіть біти регістра керування модулем.

15. Опишіть біти глобального регістра даних.

16. Опишіть біти регістра статусу.

17. Опишіть біти регістра дозволу переривання.

18. Опишіть біти регістрів даних.

19. Яким виразом визначається результат перетворення АЦП?

20. Наведіть та поясніть характеристику перетворення АЦП.

21. Назвіть основні параметри АЦП.

22. Назвіть та поясніть способи запуску АЦП.

23. Поясніть як саме відбувається перетворення вхідного сигналу у двійковий код.

24. Яку роль виконує мультиплексор в схемі АЦП?

25. Наведіть короткий опис кожного з контактів АЦП.

ПРАКТИЧНА РОБОТА № 7 ВИКОРИСТАННЯ УНІВЕРСАЛЬНОГО АСИНХРОННО-СИНХРОННОГО ПРИЙМАЧА- ПЕРЕДАТЧИКУ USART

Мета роботи - вивчення використання універсального асинхронно-синхронного приймача- передатчику USART.

Загальні відомості

USART – універсальний асинхронно-синхронного приймач-передатчик, пристрій для передачі інформації через послідовний порт, коли біти числа передаються по черзі по одній лінії. Мікроконтролер використовує два виходи, один на приймання інформації, другий – передачу. Кожен пристрій на лінії передачі утримує високий рівень. За цією ознакою пристрій може визначати, чи активний пристрій на іншій стороні та відстежити момент його підключення.

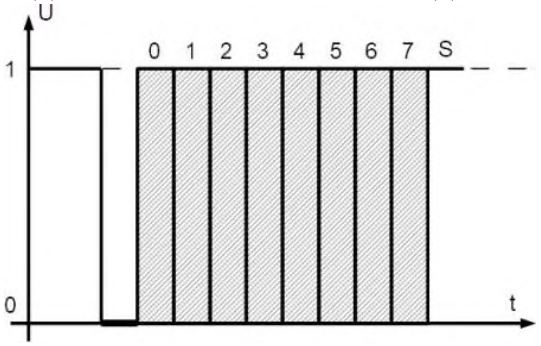


Рисунок 7.1 - Передача восьми розрядного байту з одним стоп-бітом без контролю парності

Діаграма передачі байту показана на рис. 7.1. З діаграми видно, для того щоб приймач зрозумів, що почалася передача байту, передавач повинен видати низький рівень на час передачі одного біту. Далі передається байт від молодшого до старшого біту, але порядок може бути й зворотнім. В останню чергу передається біт високого рівня для гарантії, що стартовий біт наступного байту не буде пропущений приймачем.

На діаграмі показано мінімальна конфігурація пакету сигналів для передачі байту. Для більш надійного зв’язку можна використовувати перед стоп- бітом ще один додатковий біт парності. За домовленістю цей біт парності гарантує парну або непарну кількість одиничних бітів в пакеті. Тим самим система може перевіряти випадки випадкового помилкового прийняття біту, при цьому парність бітів в доповненому байті порушиться і приймач матиме ознаку прийняття інформації з помилкою. Також деякі повільні пристрої вимагають більш довгого стоп-біту довжиною 1,5 або 2 від довжини імпульсу на один біт. Алгоритми передачі та прийому байту показано на рис. 2 та 3.

За логікою цей протокол повністю відповідає комп’ютерному COM порту, тому за цим протоколом через мікросхему (наприклад max232) перетворювача рівнів, мікросхему можна під’єднати до комп’ютеру.

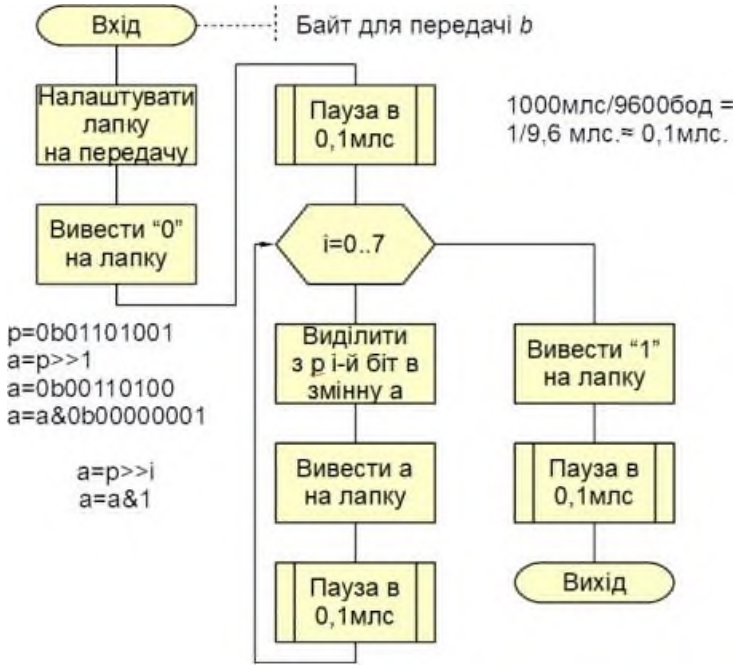


Рисунок 7.2 - Алгоритм передачі байту в послідовний USART порт

Рисунок 7.3 - Алгоритм прийому байту з послідовного USART порту

В мікроконтролері реалізована апаратна реалізація послідовного синхронно-асинхронного інтерфейсу. Розглянемо програму, яка реалізує приймання та передачу байтів по USART інтерфейсу.

Для використання апаратного USART інтерфейсу будуть потрібні наступні бібліотеки:

```
#include <stm32f4xx.h>
#include <stm32f4xx_rcc.h>
#include <stm32f4xx_gpio.h>
#include <stm32f4xx_usart.h>
#include <misc.h>
```

Більшість з них відомі, новою тут буде бібліотека обслуговування

послідовного порту USART. Далі йде функція для налаштування виходів мікросхеми на альтернативні функції. З документації можна визначити, що USART1 використовує 9 та 10 піни порту А:

```
void Usart_gpio_init(){ GPIO_InitTypeDef gpio;
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
gpio.GPIO_Pin = GPIO_Pin_9 | GPIO_Pin_10; gpio.GPIO_Mode =
GPIO_Mode_AF; gpio.GPIO_Speed = GPIO_Speed_50MHz;
gpio.GPIO_OType = GPIO_OType_PP; gpio.GPIO_PuPd =
GPIO_PuPd_UP; GPIO_Init(GPIOA, &gpio);
GPIO_PinAFConfig(GPIOA, GPIO_PinSource9, GPIO_AF_USART1);
GPIO_PinAFConfig(GPIOA, GPIO_PinSource10, GPIO_AF_USART1);
}
```

Наступним кроком є налаштування самого інтерфейсу USART1 на приймання-передачу байтів:

```
void Usart_init(){ USART_InitTypeDef usart;
usart.USART_BaudRate = 9600;
usart.USART_WordLength = USART_WordLength_8b;
usart.USART_StopBits = USART_StopBits_1; usart.USART_Parity =
USART_Parity_No;
usart.USART_HardwareFlowControl =
USART_HardwareFlowControl_None; usart.USART_Mode =
USART_Mode_Rx | USART_Mode_Tx; USART_Init(USART1, &usart);
USART_Cmd(USART1, ENABLE);
}
```

Тут використано структуру USART_InitTypeDef. Розглянемо використані параметри. USART_BaudRate – вказується швидкість передачі в бодах (кількість бітів за секунду), тут обрано стандартне значення 9600 бодів, що відповідає налаштуванню по замовчанню Bluetooth модуля HC-06, який авторами був використаний для пробного зв'язку з комп'ютером. Далі USART_WordLength відповідає за довжину байту, який передається. При передачі між контролерами це поле може набувати значень USART_WordLength_8b та USART_WordLength_9b. Однак при обміні дев'ятирозрядні байти при зв'язку з комп'ютером не використовуються.

USART_StopBits – може набувати значення USART_StopBits_0_5, USART_StopBits_1, USART_StopBits_1_5 та USART_StopBits_2, що відповідає проміжку між байтами в 0.5, 1, 1.5 та 2 одиничних біти. Ці проміжки використовують для гарантії часового проміжку між двома бітами, також довгі проміжки 1.5 та 2 біти можуть бути задіяні при обміні інформацією з повільними пристроями – за цей час вони повинні встигнути обробити прийнятий щойно байт та підготуватися до приймання наступного.

USART_Parity – використовується для контролю правильності приймання байту. Значення USART_Parity_Even означає використання додаткового біту, який гарантує парну кількість одиничних бітів, якщо буде помилка прийнятті байту, парність бітів порушиться; USART_Parity_Odd – кількість одиничних бітів є

непарною; USART_Parity_No – біт парності не використовується. Біт парності не використовують, коли помилка в передачі допустима або контролюється правильність передачі іншими програмними засобами, наприклад пакет байтів доповнюють контрольною сумою та перевіряють прийняту контрольну суму та розраховану самостійно.

USART_HardwareFlowControl – апаратний контроль дозволу на приймання та передачі байтів. Може набувати наступних значень: USART_HardwareFlowControl_None, керування дозволом на приймання та передачу не використовується і передача/прийом може відбуватися в довільний момент часу; USART_HardwareFlowControl_RTS, використовується окремий вихід мікросхеми, на якому надається сигнал пристрою-передатчику, що данні можна/заборонено надсилати; USART_HardwareFlowControl_CTS, використовується окремий вхід мікросхеми, на якому приймається сигнал пристрою-приймача, що данні можна/заборонено надсилати; USART_HardwareFlowControl_RTS_CTS – використано двонаправлене апаратне керування дозволом на передачу та прийом даних.

Код USART_Mode = USART_Mode_Rx | USART_Mode_Tx відповідає за налаштування USART в режим приймання та передачі інформації. Також можливі варіанти, коли мікроконтролер лише приймає або лише передає інформацію.

Наступна функція відповідає за налаштування переривання подій від USART пристрою:

```
void Usart_irq_init(){ NVIC_InitTypeDef irqconf;
USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
irqconf.NVIC_IRQChannel = USART1_IRQn;
irqconf.NVIC_IRQChannelPreemptionPriority = 0;
irqconf.NVIC_IRQChannelSubPriority = 0;
irqconf.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&irqconf);
}
```

Цей код мало чим відрізняється від попереднього налаштування переривання від таймеру, а наступний код вам відомий по керуванню світлодіодами:

```
void Led_init(){
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
GPIO_InitTypeDef gpio;
gpio.GPIO_Mode = GPIO_Mode_OUT; gpio.GPIO_OType =
GPIO_OType_PP; gpio.GPIO_Pin = GPIO_Pin_12; gpio.GPIO_PuPd =
GPIO_PuPd_NOPULL; gpio.GPIO_Speed = GPIO_Speed_2MHz;
GPIO_Init(GPIOD, &gpio);
}
void Led_on(){
GPIO_SetBits(GPIOD, GPIO_Pin_12);
}
void Led_off(){
GPIO_ResetBits(GPIOD, GPIO_Pin_12);
```

```
}
```

Наступна функція є обробником переривання подій від USART пристрою,

при чому для всіх подій призначений лише один обробник. Тому, якщо в налаштуванні переривань вказано кілька подій в якості джерела переривання, функція викликатиметься для кожної з подій. Для визначення події, яка саме стала причиною переривання потрібно перевіряти регістр стану USART пристрою за допомогою функції USART_GetITStatus. В нашому випадку використано подію наявності нового прийнятого байту USART_IT_RXNE:

```
volatile unsigned char lastByte = 0;
void USART1_IRQHandler(void){
if( USART_GetITStatus(USART1, USART_IT_RXNE) ){
USART_ClearITPendingBit(USART1, USART_IT_RXNE); lastByte =
USART_ReceiveData(USART1); USART_SendData(USART1, lastByte);
//echo
}
}
```

По визначенню події потрібно обов'язково очистити відповідний біт-

ознаку події, інакше система вважатиме, що подія не оброблена і виклику переривання після надходження наступного байту не буде, а наступні байти взагалі будуть проігноровані.

Наступна функція використовується для генерування пауз:

```
void Delay(volatile int Val) {
while(Val--);
}
```

Головна ж функція викликає процедури налаштувань, при цьому світлодіод за програмно генерованим ШІМ сигналом, змінює яскравість світіння:

```
int main(void) {
SystemInit(); SystemCoreClockUpdate(); Led_init();
Usart_gpio_init(); Usart_irq_init(); Usart_init();
while(1){
Led_on(LED0); Delay(0x0100*lastByte); Led_off(LED0);
Delay(0x0100*(255-lastByte));
}
}
```

Порядок виконання роботи

1. Озайомитися із структурною проєкту що приведений вище.
2. На основі отриманих даних виконайте самостійне завдання, реалізувати зміну яскравості світіння чотирьох світлодіодів, де кожен з світлодіодів отримує власні параметри світіння..
3. Задійте світлодіоди для відображення рівня світіння прийнятого від передавача.
4. Представити результати відкомпільованого коду програми.

Зміст звіту

1. Мета роботи.
2. Відомості про порядок створення проекту.
3. Налаштування блоку USART.
4. Висновки.

Контрольні питання

1. Опишіть структурну схему сполучення МП/МК з модемом за допомогою інтерфейсу RS – 232C.
2. Скільки модулів УАПП мають мікроконтролери сімейства LPC2300?
3. Які особливості мають модулі UART1 та UART3?
4. Назвіть основні керуючі регістри модуля UART.
5. Як виконується ініціалізація модуля UART?
6. Як формується тактовий сигнал модуля UART?
7. Як здійснюється керування функцією автоматичного визначення швидкості передачі пакету даних, який приймається?
8. В який регістр модуля UART записуються дані для передачі?
9. Прапорці якого регістра відображають стан модуля UART?
10. Як програмується потрібна швидкість обміну в модулі UART?
11. Які переривання може генерувати модуль UART?
12. Поясніть особливості організації обміну за протоколом IrDA.
13. Опишіть призначення бітів регістра U0RBR.
14. Опишіть призначення бітів регістрів U0DLL та U0DLM.
15. Опишіть призначення бітів регістрів U0IER та U0IIR.
16. Опишіть призначення бітів регістрів U0LCR та U0LSR.
17. Опишіть призначення бітів регістрів U1MCR та U1MSR.
18. Поясніть призначення основних контактів роз’єму RS–232C.
19. При якому значенні біта DLAB регістра LCR виконується програмування швидкості обміну?
20. Опишіть призначення та програмування FIF0–буферів передачі та прийому.
21. Які помилки прийому може виявляти модуль UART?
22. Як біти керуванням модемом можуть використовуватися для синхронізації обміну між мікропроцесором та терміналом (модемом)?

ПРАКТИЧНА РОБОТА № 8 РОБОТА З SPI

Мета роботи - вивчення використання SPI.

Загальні відомості

Serial Peripheral Interface (SPI) - популярний інтерфейс для послідовного обміну даними між мікросхемами. Шина SPI організована за принципом "майстер-підлеглий". В якості майстра шини зазвичай виступає мікроконтролер, але їм також може бути програмована логіка, DSP-контролер або спеціалізована інтегральна схема. До одного керуючого пристрою паралельно включають ланцюг підлеглих пристроїв (рис. 8.2).

Головним складовим блоком інтерфейсу SPI є звичайний регістр зсуву, сигнали синхронізації і введення/виводу бітового потоку з якого і утворюють інтерфейсні сигнали. Таким чином, протокол SPI правильніше що назвати непротоколом передачі даних, а протоколом обміну даними між двома зсувними регістрами, кожен з яких одночасно виконує і функцію приймача, і функцію передавача. Неодмінною умовою передачі даних по шині SPI є генерація сигналу синхронізації шини. Цей сигнал має право генерувати лише майстер шини і від цього сигналу повністю залежить робота всіх підлеглих пристроїв на шині.

Приклад найбільш простого підключення по шині SPI показаний на рис. 8.1 та 8.2, де однойменні висновки з'єднуються між собою. Присутні два канали передачі даних (MOSI і MISO) один канал для подачі тактових імпульсів (SCLK), а також лінія включення підлеглого пристрою (SS).

Підлеглий стає активним при подачі низького рівня по лінії SS.

Плата STM32F4Discovery має в своєму складі мікросхемний акселерометр (рис. 3.11), який може працювати в режимах обміну даними I2C та SPI. Використовуватимемо останній, бо розводка плати виконана саме під цей режим.

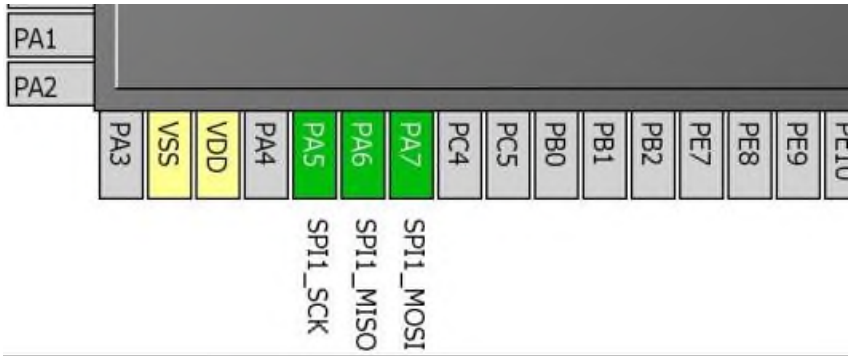


Рисунок 8.1 - Позначення виходів SPI1 в STM32CubeMX

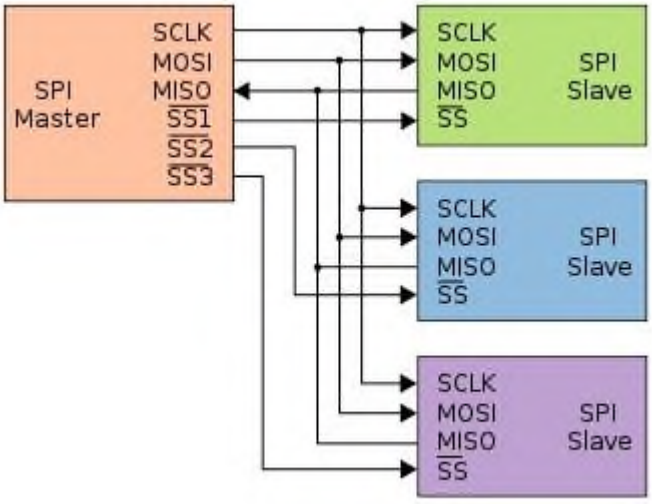


Рисунок 8.2 - Схема включення «один ведучий – кілька незалежних підлеглих»

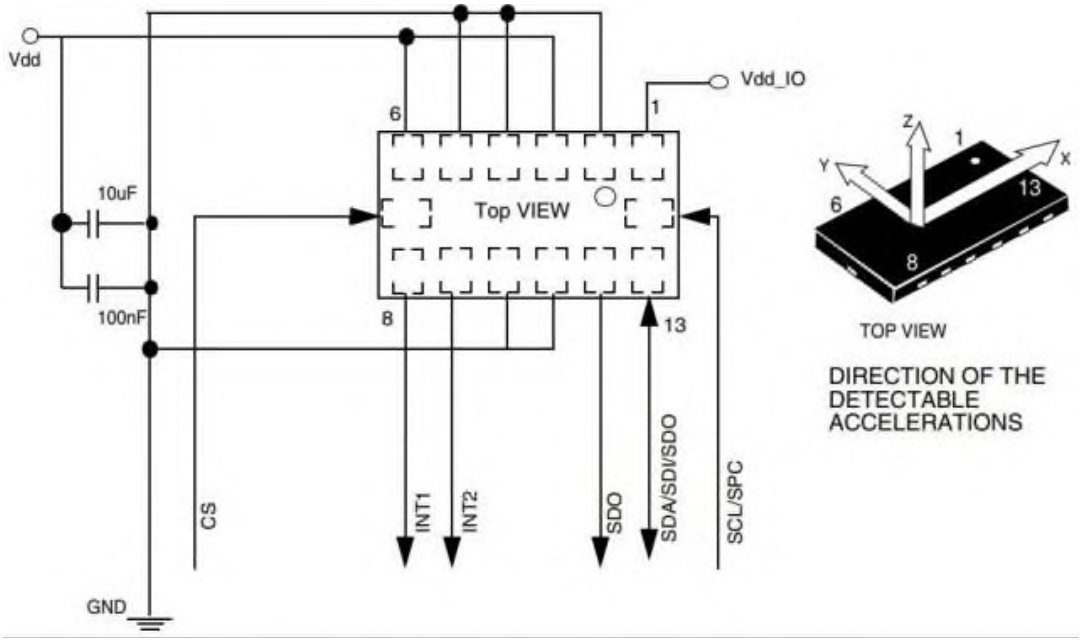


Рисунок 8.3 - Схема під’єднання акселерометру з офіційної документації

На рис. 8.3 показано контакти мікросхеми акселерометру.

З причини використання SPI, виходи мікросхеми мають наступну відповідність: SDA=MOSI, SCL=SCK, SDO=MISO.

Вибір інтерфейсу вибирається на вході мікросхеми CS. Якщо на нозі логічний 1, то акселерометр використовує I2C, в іншому випадку – SPI. Також з документації маємо наступні характеристики мікросхеми:

1) Напруга живлення 2.16-3.6 В.
2) Вимірювання прискорення по трьох осях.
3) Два діапазону вимірювання 2G / 8G.
4) Два настроюються виходу для переривань.
5) Самодіагностика.

6) Виявлення кліків (постукувань).

7) Вбудований фільтр.

Розглянемо порядок роботи інтерфейсу.

Взаємодія з акселерометром здійснюється через його регістри. Щоб прочитати або щось записати в них дані, потрібно відправити їх через SPI посилку певного формату. В документації наведено датаграми обміну інформацією (рис. 8.4).

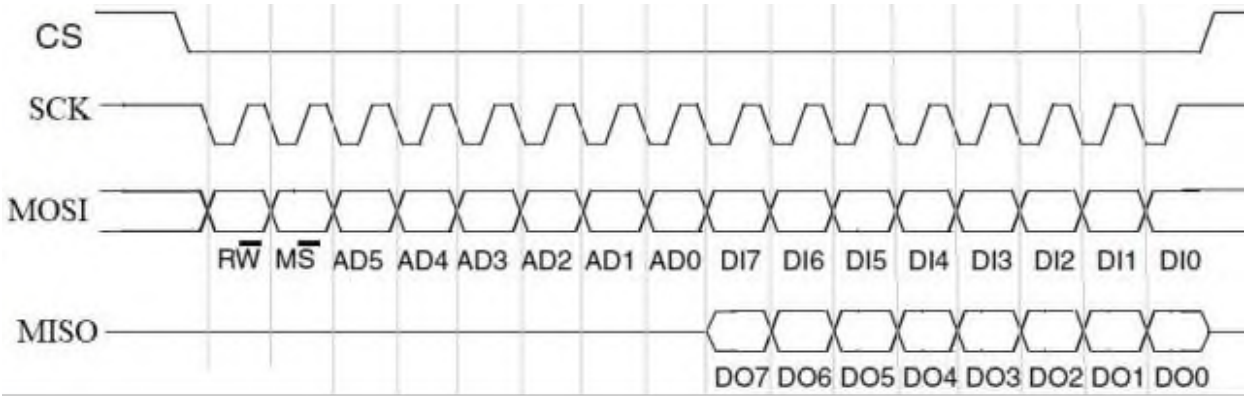


Рисунок 8.4 - Датаграма процесу обміном байтом майстра з підлеглим пристроєм

DO7..DO0 – байт даних відправлений акселерометром в мікроконтролер; DI7..DI0 – байт даних переданий мікро контролером в акселерометр;

AD5 .. AD0 – адреса яка записується/зчитується до/з регістра;

RW – якщо біт = 0 то байт даних DI7..DI0 буде записаний в регістр за адресою AD5..AD0, в іншому випадку (RW=1) байт даних DO7..DO0 буде прочитаний з регістру за адресою AD5..AD0.

MS - використовується якщо ми хочемо прочитати/записати кілька регістрів поспіль. Якщо біт скинутий (=0), то після передачі адреси ми будемо зчитувати/записувати один і той же регістр незалежно від того, скільки разів буде зроблена спроба читання/запису. Якщо ж цей біт встановлений (=1), то адреса буде автоматично збільшуватися на одиницю після кожного запису або читання, це зручно використовувати для запису/зчитування кілька байтів поспіль.

Для того щоб прочитати вміст одного регістра (наприклад з адресою 0x0F) потрібно виконати наступну послідовність дій:

1) Вихід CS притиснути до землі (=0).

2) Передати перший байт (0x8F) який містить адресу регістра який потрібно зчитати; при цьому скинутий біт MS (так як ми читаємо тільки один регістр) і встановлений біт RW (адже ми читаємо дані), що й спричинило доповнення адреси до значення 0x8F.

3) Передати довільний байт. Адже інтерфейс SPI так влаштований, що підлеглий пристрій не може передавати синхронізуючі імпульси. Тому акселерометр почне передавати значення з регістру лише в момент передачі мікроконтролером другого байту (який по суті буде просто проігноровано).

4) Прочитати значення з регістра даних SPI (те, що прийшло від акселерометра).

5) Встановити високий логічний рівень на виводі CS.

Якщо потрібно швидко читати один і той самий регістр, то можна постійно повторювати кроки 3 і 4.

Запис одного регістра не складніше за читання, спробуємо записати байт 0x47 в регістр за адресою 0x20. Для цього потрібно:

1) Вихід CS притиснути до землі.

2) Передати перший байт (0x20) який містить адресу регістра для запису, скинутих біт MS=0 (так записується лише один регістр) і скинутий біт RW=0 (пишемо дані).

3) Передати нове значення регістра (0x47)

4) Встановити високий логічний рівень на виводі CS

Якщо є потреба швидко писати будь-які значення в один і той же регістр, то можна постійно повторювати 3-й крок.

Читання кількох регістрів виконують за наступним алгоритмом:

1) Вихід CS притиснути до землі.

2) Передати перший байт (0xCF) який містить адресу початкового регістру для читання, та у якому встановлений біт MS (читання кількох регістрів), та встановлений біт RW (ознака читання даних).

3) Передати довільний байт (який по суті буде просто проігноровано).

4) Прочитати значення з регістра даних SPI (відповідь від акселерометра).

5) Повторювати кроки 3 і 4 поки не прочитаємо потрібну кількість регістрів. Адреса читання буде збільшуватися на одиницю самостійно.

6) Встановити високий логічний рівень на виводі CS.

І нарешті, запис кількох регістрів виконують за наступним алгоритмом:

1) Вихід CS притиснути до землі.

2) Передати перший байт (0x4F) який містить адресу початкового регістру для читання, та у якому встановлений біт MS (читання/запис кількох регістрів), та скинутий біт RW (ознака запису даних).

3) Передати значення до регістру 0x0F.

4) Передати значення до регістру 0x10.

5) Передати значення до регістру 0x11. 6) …

7) Встановити високий логічний рівень на виводі CS.

Залишається з’ясувати, які саме регістри читати та записувати при роботі з пристроями. Цю інформацію можна отримати для конкретного пристрою з його документації. Для акселерометру розглянемо документовані регістри та параметри.

Who_Am_I (0x0F) – регістр містить ідентифікатор акселерометру LIS302DL. Значення можна лише прочитати, прочитане значення повинне бути рівним 0x3b. За цим значенням можна перевіряти наявність зв’язку з акселерометром.

CTRL_REG1 (0x20) – перший регістр стану акселерометру. Його біти відображені нижче, де в дужках наведене значення по замовчанню:

DR (0), PD (0), FS (0), STP (0), STM (0), Zen (1), Yen (1), Xen (1).

DR – біт налаштовує частоту вибірки. Якщо він встановлений, то частота 400Гц, якщо скинуто то 100Гц. Впливає на точність вимірювань.

PD – біт керування живленням. Біт скинутий – живлення вимкнене і навпаки. Поки не встановлено цей біт вимірювання не проводяться.

FS – біт вибору діапазону вимірювань. Біт скинутий – діапазон ±2g, якщо ж встановлений, то діапазон вимірювань складає ±8g.

STP і STM – біти керування режимом самодіагностики.

Zen, Yen, Xen – біти дозволу генерування сигналу готовності даних для осі Z, Y та X. Якщо скинути біт, прискорення по цій осі буде вимірюватися, але прапор готовності даних (в регістрі STATUS_REG) встановлено не буде.

CTRL_REG2 (0x21) – другий регістр налаштування:

SIM (0), BOOT (0), - , FDS (0), HP_FF_W-U2 (0), HP_FF_W-U1 (0), HP_coeff2 (0), HP_coeff1 (0).

SIM – біт вибору режиму SPI. Якщо встановлено, то акселерометр переключиться в трьохпровідний режим SPI (прийом і передача буде відбуватися по одному дроту). За замовчуванням він скинутий і SPI працює в звичайному 4-х дротовому режимі.

BOOT – при запису одиниці в цей біт, відбувається скидання всіх налаштувань акселерометра за замовчуванням. Регістри статусу не скидаються.

FDS – включити/вимкнути фільтр. Якщо біт встановлено, то тяжіння землі не робить вплив на вимірювання і навпаки. Простіше кажучи, якщо фільтр включений, то поки до акселерометру не надаватимуть прискорення, в регістрах даних X, Y, Z будуть нулі (або майже нулі). Якщо ж вимкнути фільтр, то дані в регістрах X, Y, Z будуть залежати від орієнтації акселерометра в просторі.

HP_FF_WU2 і HP_FF_WU1 – включають/вимикають фільтр верхніх частот для двох модулів генеруючих переривання (FF_WU1 і FF_WU2). Частота зрізу фільтра налаштовується за допомогою бітів HP_ coeff2 і HP_ coeff1:

Таблиця 8.1 – Вибір частоти зрізу для фільтру

HP_ coeff2	HP_ coeff1	Частота зрізу (Біт DR = 0)	Частота зрізу (Біт DR = 1)
0	0	2 Гц	8 Гц
0	1	1 Гц	4 Гц
1	0	0.5 Гц	2 Гц
1	1	0.25 Гц	1 Гц

CTRL_REG3 (0x21) – Регістр налаштування переривань:

IHL (0), PP_OD (0), I2CFG2 (0), I2CFG1 (0), I2CFG0 (0), I1CFG2 (0), I1CFG1 (0), I1CFG0 (0).

IHL – якщо біт скинутий, то при виникненні переривання, на ногах INT1 і INT2 з’явиться логічна одиниця. Якщо біт встановлено, то поки переривання не сталося на висновках INT1 і INT2 присутня логічна одиниця, а в момент виникнення переривання, на вихід подається логічний нуль.

PP_ OD – тип виходів INT1 та INT2. 0 – підтяжка до нуля, 1 – відкритий колектор.

За допомогою бітів I2 CFG [2..0] і I1 CFG [2..0] можна вибрати джерело викликів переривання INT1 і INT2:

Таблиця 8.2 – Вибір джерела переривання

I1 (2) _CFG2	I1 (2) _CFG1	I1 (2) _CFG0	Джерело переривання
0	0	0	Переривання вимкнені, виходи підтягнені до землі.
0	0	1	FF_WU_1
0	1	0	FF_WU_2
0	1	1	FF_WU_1 та FF_WU_2
1	0	0	Дані готові.
1	1	1	Переривання від «кліка».

FF_WU_1 і FF_WU_2 це два незалежних блоку призначених для генерації переривань в разі якщо прискорення по одній або декількох осях, вийде за заданий поріг.

HP_FILTER_RESET (0x23) –Якщо буде включений фільтр верхніх частот, після читання вмісту цього регістра в регістри Out_X, Out_Y і Out_Z запишуться нулі. Тут важливий сам факт читання регістра, а не ті дані, які прочиталися з нього.

STATUS_REG (0x27) – регістр статусу, кожен біт якого є ознакою певної події: ZXYOR (0), ZOR (0), YOR (0), XOR (0), ZYXDA (0), ZDA (0), YDA (0),

ZDA (0).

ZXYOR - Дані в регістрах Out_X, Out_Y і Out_Z були переписані новими значеннями, а їх попередній вміст не встигли прочитати.

ZOR - Дані в регістрі Out_Z переписано новими, до того як були прочитані попередні.

YOR - Дані в регістрі Out_Y переписано новими, до того як були прочитані попередні.

XOR - Дані в регістрі Out_X переписано новими, до того як були прочитані попередні.

ZYXDA - Доступні нові дані в регістрах Out_X, Out_Y і Out_Z. ZDA - Доступні нові дані в регістрі Out_Z.

YDA - Доступні нові дані в регістрі Out_Y. XDA - Доступні нові дані в регістрі Out_X.

OUT_X (0x29) – восьмирозрядний регістр даних, в який записується прискорення по осі X.

OUT_Y (0x2b) – восьмирозрядний регістр даних, в який записується прискорення по осі Y.

OUT_Z (0x2d) – восьмирозрядний регістр даних, в який записується прискорення по осі Z

У акселерометра є два незалежних модуля, які вміють самостійно читати дані з регістрів Out_X , Out_Y , Out_Z і на їх основі генерувати переривання. Для налаштування кожного модуля використовуються 4 регістра які будуть описані нижче. Для зручності налагодження можна причепити до ніг INT1/INT2 по світлодіоду через резистор. Тоді якщо переривання трапиться це буде відразу видно.

FF_WU_CFG_1 (0x30), FF_WU_CFG_2 (0x34) – регістр налаштовує умови виникнення переривання.

AOI (0), LIR (0), ZHIE (0), ZLIE (0), YHIE (0), YLIE (0), XHIE (0), XLIE (0)

AOI – біт визначає в коли згенерувати переривання. Якщо він скинутий, то переривання виникне, коли відбудуться всі події обраних бітами XHIE/ YHIE/ ZHIE/ XLIE/ YLIE/ ZLIE.

LIR – якщо біт варто, то коли ми будемо читати регістр FF_WU_SRC_1 (2), його вміст автоматично скидатиметься.

XHIE / YHIE / ZHIE - якщо біт виставлений, то акселерометр починає стежити за станом відповідного регістра даних, і коли значення прискорення за відповідною осі перевищить граничне – буде виставлений прапор в регістрі FF_WU_SRC_1 (2).

XLIE / YLIE / ZLIE - якщо біт виставлений, то акселерометр починає стежити за станом відповідного регістра даних, і коли значення прискорення за відповідною осі стане менше порогового – буде виставлений прапор в регістрі FF_WU_SRC_1 (2).

FF_WU_SRC_1 (0x31), FF_WU_SRC_2 (0x35) – регістр прапорів. Якщо біт LIR (в регістрі FF_WU_CFG_X) встановлено, то при зчитуванні цього регістра, він скидається. Якщо до цього відбулося переривання, то відразу після читання, обидві ноги INT1 / INT2 перемикаються в початковий стан.

IA (0), ZH (0), ZL (0), YH (0), YL (0), XH (0), XL (0).

IA - якщо біт встановлений, то переривання сталося.

ZH / YH / XH - якщо біт встановлений то прискорення за відповідною осі перевищило поріг.

ZL / YL / XL - якщо біт встановлений то прискорення за відповідною осі стало менше порога.

Природно, перші шість біт будуть встановлюватися лише в тому випадку, якщо встановлено біти XHIE / YHIE / ZHIE / XLIE / YLIE / ZLIE в регістрі FF_WU_CFG_1 (2).

Налаштування для генерування «кліків / постукувань» можна прочитати в документації до акселерометра.

На платі розробника акселерометр підключено за наступною схемою (рис.8.5):

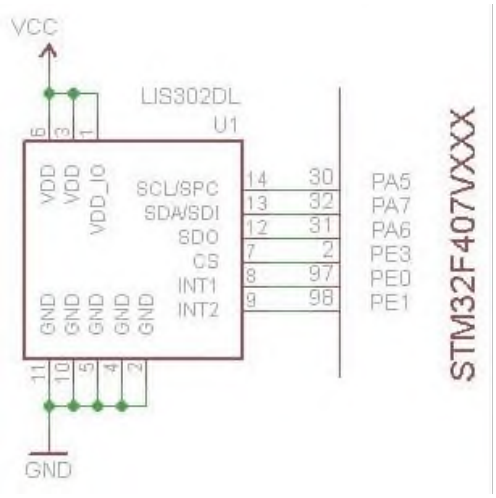


Рисунок 8.5 - Підключення акселерометру до мікроконтролера по шині SPI

В нашому випадку переривання використовуватися не будуть, задача полягає в опитуванні показань прискорення та засвічування найвищого світлодіоду. Для цього потрібно в циклі читати регістр стану акселерометру STATUS_REG (0x27) і, при наявності встановленого прапору ZYXDA прочитати показання OUT_X (0x29) та OUT_Y (0x2b) – значення прискорення по двом напрямкам. Максимальне по модулю значення вказує пару протилежних світлодіодів, а знак – який саме світлодіод з пари потрібно включити.

Почнемо з ініціалізації SPI інтерфейсу:

```
void SpiInit() {
GPIO_InitTypeDef GPIO_InitStructure; SPI_InitTypeDef SPI_InitStructure;

// Тактування SPI1 та порту A
RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1,
ENABLE);
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA | RCC_AHB1Periph_GPIOE,
ENABLE);

// Налаштування виходів для роботи в SPI1 GPIO_PinAFConfig(GPIOA,
GPIO_PinSource7,
GPIO_AF_SPI1);
GPIO_PinAFConfig(GPIOA, GPIO_PinSource5, GPIO_AF_SPI1);
GPIO_PinAFConfig(GPIOA, GPIO_PinSource6, GPIO_AF_SPI1);
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7 | GPIO_Pin_6 | GPIO_Pin_5;
GPIO_Init(GPIOA, &GPIO_InitStructure); GPIO_InitStructure.GPIO_Mode =
GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_Speed =
GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP; GPIO_InitStructure.GPIO_Pin =
GPIO_Pin_3; GPIO_Init(GPIOE, &GPIO_InitStructure);
GPIO_SetBits(GPIOE,GPIO_Pin_3); //Вихід CS

//Заповнюємо структуру з налаштуванням SPI SPI_InitStructure.SPI_Direction =
SPI_Direction_2Lines_FullDuplex; //Двонаправлений обмін
SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b; //Восьмирозрядний режим
SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low; SPI_InitStructure.SPI_CPHA =
SPI_CPHA_1Edge;
//Біт повинен прийматися в момент переходу сигналу тактування від низького
до високого рівня (рис.8.4)
```

```
SPI_InitStructure.SPI_NSS = SPI_NSS_Soft; //
Керування NSS сигналом відбувається програмно
SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_32;
//Преддільник SCK
SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB; // В байті може передаватися
першим старший або молодший біт, правильний напрям визначається з
документації пристрою, тут встановлена передача зі старшого біту
SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
// Режим - майстер
SPI_Init(SPI1, &SPI_InitStructure);
//Застосовуємо налаштування SPI1 SPI_Cmd(SPI1, ENABLE); // Вмикаємо модуль
// Сигнал NSS контролюється програмно, потрібно встановити його в
одиницю,якщо ж його скинути в нуль, то SPI перейде в мультимайстерну топологію
(сигнал скасування повноважень майстра).
SPI_NSSInternalSoftwareConfig(SPI1, SPI_NSSInternalSoft_Set);
}
```

Обмін інформацією будемо використовувати лише по одному байту.

Передача/прийом буферу читач може реалізувати самостійно. Першочергово створимо функцію запису й одночасного приймання байту:

```
int WriteSPIData(uint8_t data) {
while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_BSY) == SET); //Чекаємо
звільнення пристрою
SPI_I2S_SendData(SPI1, data); //Ініціюємо передачу байту
while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_BSY) == SET); //Чекаємо
завершення передачі
return SPI_I2S_ReceiveData(SPI1); //Повертаємо вхідний байт
}
```

Тепер розглянемо запис байту до регістру за розглянутим до цього алгоритмом:

```
void SetReg(uint8_t address, uint8_t value) {
GPIO_ResetBits(GPIOE,GPIO_Pin_3); //SS
встановимо в 0
WriteSPIData(address); //Пишемо адресу регістра
WriteSPIData(value); //Відправляємо значення до регістру
GPIO_SetBits(GPIOE, GPIO_Pin_3); //SS
встановлюємо в 1 – обмін завершено
}
```

Тепер читання значення регістру:

```
int GetReg(uint8_t address) {
int data=0; //Змінна для збереження відповіді
address|=(1<<7); //Встановлюємо прапор RW=1 –
режим читання однобайтного читання
GPIO_ResetBits(GPIOE,GPIO_Pin_3); //SS
```

```
встановимо в 0 – початок обміну
WriteSPIData(address); //Пишемо адресу регістра
data = WriteSPIData(0x00); //Відправляємо довільне значення до регістру, автоматично отримаємо відповідь мікроконтролера
GPIO_SetBits(GPIOE,GPIO_Pin_3); //SS
встановлюємо в 1 – обмін завершено
return data; //Повертаємо отриманий результат
}
```

Тепер наведемо реалізацію головної функції програми:

```
int main(void)
{
SystemInit();// Налаштування таймінгу
SystemCoreClockUpdate();// Оновлення фактичної частоти процесора
LedConfig();//Ініціюємо світлодіоди
SpiInit();//Ініціюємо обмін з акселерометром

unsigned char idf = GetReg(0x0F); //Читаємо ідентифікаційний байт пристрою
if(idf!=0x3B){
//Не наш пристрій, покажемо це світлодіодами
LedOn(LED1 | LED2 | LED3 | LED4);
while(1); //Працювати нема з чим
}
SetReg(0x20, 0b11000000);//DR=1 (400Hz), PD=1
(живлення увімкнене), ініціювання акселерометру
while(1){ //Робочий цикл:
int x = GetReg(0x29); //Читаємо прискорення по X
int y = GetReg(0x2B); // - по Y
if( Abs(x)>Abs(y) ){ //Обираємо світлодіод
if(x>0){
LedOff(LED2 | LED3 | LED4); LedOn(LED1);
}else{
LedOff(LED2 | LED1 | LED4); LedOn(LED3);
}
}else{
if(y>0){
LedOff(LED1 | LED3 | LED4); LedOn(LED2);
}else{
LedOff(LED2 | LED1 | LED3); LedOn(LED4);
}
}
}
}
```

Порядок виконання роботи

1. Озайомитися із структурною проєкту що приведений вище.

2. На основі отриманих даних виконайте самостійне завдання, реалізувати виведення даних на семисегментний чотирьох позиційний індиктор, який підєднаний до регустру зсуву.

2. Представити результати відкомпільованого коду програми.

Зміст звіту

1. Мета роботи.
2. Відомості про порядок створення проекту.
3. Налаштування блоку SPI.
4. Висновки.

Контрольні питання

1. На яку відстань і з якою швидкістю можна здійснювати обмін даними через інтерфейс SPI?
2. У яких режимах може працювати мікроконтролер при обміні даними інтерфейсом SPI?
3. Скільки та які виводи мікроконтролера використовує модуль SPI?
4. Який регістр призначений для керування модулем SPI?
5. Який регістр використовується для контролю стану модуля?
6. Як задати режим роботи мікроконтролера в режимі «MASTER»?
7 .Як визначити кінець передачі байта?
8. За якої умови, при передачі даних від ведучого до веденого, можлива передача у зворотному напрямі?
9. Що означає те, що у модулі реалізована одинарна буферизація при передачі та подвійна при прийомі?
10. Чи можна до інтерфейсу підключати декілька периферійних пристроїв?
11. Назвіть кількість режимів передачі даних та принцип їх роботи.
12. Що є джерелом тактового сигналу при роботі модуля?
13. Наведіть опис бітів регістра керування.
14. Наведіть опис бітів регістра стану.
15. Наведіть опис бітів регістра прапорців переривання.
16. Опишіть передачу даних в режимі ведучого.
17. Опишіть передачу даних в режимі веденого.
18. Опишіть переривання від модуля SPI.
19. Опишіть часові діаграми роботи SPI–інтерфейсу.
20. Опишіть схему з’єднання вузлів SPI двох різних пристроїв.
21. Як компенсуються часові затримки при передачі сигналів між двома пристроями?
22. Наведіть та опишіть приклад мережі SPI.
23. За якою формулою розраховується тактова частота модуля SPI?

СПИСОК ЛІТЕРАТУРИ

1. Прищепа М.М., Погребняк В.П. Мікроелектроніка. Частина І. Елементи електроніки. – Київ: Вища школа, 2004. – 431 с.

2. Закалик Л.У., Ткачук Р.А. Основи мікроелектроніки. - Тернопіль, 1998. - 380 с.

3. Хоружний В.А., Письмецький В.О. Функціональна мікроелектроніка, опто- та акустоелектроніка. - Харків, 1995. - 186 с.

4. Сенько В.І., Панасенко М.В., Сенько Є.В. Електроніка і мікросхемотехніка. - Т.1. Елементна база електронних пристроїв. - Київ: Обереги, 2000. - 300 с.

5. Стахів П.Г., Коруд В.І., Гамола О.Є. Основи електроніки: функціональні елементи та їх застосування. - Львів: Новий світ-2000, 2003. - 128 с.

6. Радіотехніка: Енциклопедичний навчальний довідник: Навч. Посіб. / за ред. Ю.Л.Мазора, Є.А.Мачуського, В.І. Правди. – Київ: Вища школа, 1999. – 838 с.

7. Мікроелектроніка і наноелектроніка. Вступ до спеціальності / Ю. М. Поплавко, О. В. Борисов, В. І. Ільченко та ін. – Київ: НТУУ «КПІ», 2010. – 160 с